Verification Continuum™

VC Verification IP
PCIe
UVM User Guide

Version Q-2020.06, June 2020

Contents

Preface

About This Manual

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

Manual Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Introduction", introduces the VC PCIe and its features.
- Chapter 2, "Installation and Setup", describes system requirements and provides instructions on how to install, configure, and begin using the VC PCIe.
- Chapter 3, "General Concepts", introduces the PCIe VIP within the UVM environment and describes the data objects and components that comprise the VIP.
- Chapter 4, "Verification Features", describes the verification features supported by PCIe VIP such as, Verification Planner and Protocol Analyzer.
- Chapter 5, "General VIP Protocol Features", describes the general VIP protocol features available with the Synopsys PCIe Verification IP.
- Chapter 6, "Gen4 Features", describes the Gen4 features available with the Synopsys PCIe Verification IP.
- Chapter 7, "Gen5 Features", describes the Gen5 features available with the Synopsys PCIe Verification IP.
- Chapter 8, "PIPE Features", describes the PIPE feature support available with the Synopsys PCIe Verification IP.
- Chapter 9, "PIPE5 Features", describes the PIPE5 feature support available with the Synopsys PCIe Verification IP.
- Chapter 10, "PCIe Verification Topologies", describes various ways the PCIe VIP can be connected with other components.

- Chapter 11, “Using the PCIe Verification IP”, shows how to install and run a getting started example.
- Chapter 12, “PCIe Device Agent”, provides an overview on PCIe Device Agent and how to use them.
- Chapter 13, “PCIe Agent”, provides an overview on PCIe Agent and how to use them.
- Chapter 14, “Using the Transaction Layer”, describes features of the Transaction Layer and how to use them.
- Chapter 15, “Data Link Layer Features and Classes”, describes features of the Data Link Layer and how to use them.
- Chapter 16, “PHY Layer Features and Classes”, describes features of the PHY Layer and how to use them.
- Chapter 17, “Using the Driver Application”, describes the procedure for using the driver application.
- Chapter 18, “Functional Coverage”, provides the overview of PCIe functional coverage features, usage mechanism, coverage generation and coverage back annotation process.
- Chapter 19, “Using Callbacks”, describes the basic usage of Callbacks, provides some examples, and gives tips for debugging them.
- Chapter 20, “Passive Monitor”, describes the use of PCIe Passive Monitor VIP in testbenches that comply with the SystemVerilog Universal Verification Methodology (UVM).
- Chapter 21, “Support for CCIX”, describes the CCIX features available with the Synopsys PCIe SVT Verification IP.
- Appendix A, “PCIe PIPE Interface”, provides the detailed list of all the signal widths and associated macros present in PIPE and PIPE5 interface.
- Appendix B, “Legacy Testbench”, provides the information on using the instantiation models for legacy testbench.
- Appendix C, “Functional Coverage”, describes how to enable and use the functional coverage features.
- Appendix D, “Partition Compile and Precompiled IP”, describes the PC and PIP features in Verification Compiler to optimize the compilation performance.
- Appendix E, “Protocol Checks”, outlines the process for working through and reporting VC PCIe issues.
- Appendix F, “PCIe PIE-8 Interface”, describes the PIE-8 specification in PCIe VIP.
- Appendix G, “PCIe Compile-Time Parameters”, contains a table of parameters that can only be set before compilation, not at runtime.
- Appendix H, “Verilog Task/Parameter to SVT Class Mapping”, contains tables that show the correspondence between SVT classes and Verilog tasks or parameters.
- Appendix I, “SolvNet PCIe VIP Articles”, provides the lists and links to all the technical articles published on the PCIe VIP.
- Appendix J, “Reporting Problems”, outlines the process for working through and reporting VC PCIe issues.

Web Resources

- Documentation through SolvNetPlus: https://solvnetplus.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:

- Go to https://solvnetplus.synopsys.com and open a case.
    - Enter the information according to your environment and your issue.
    - If applicable, provide the information noted in Appendix J, "Reporting Problems".
- Send an e-mail message to support_center@synopsys.com
    - Include the Product name, Sub Product name, and Product version for which you want to register the problem.
    - If applicable, provide the information noted in Appendix J, "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

This chapter gives a basic introduction, overview and features of the PCIe UVM Verification IP.

This chapter discusses the following topics:

1.1 Introduction

The VC PCIe Verification IP supports verification of SoC designs that include interfaces implementing the PCIe Specification. This document describes the use of this VIP in testbenches that comply with the SystemVerilog Universal Verification Methodology (UVM). This approach leverages advanced verification technologies and tools that provide:

- Protocol functionality and abstraction
- Constrained random verification
- Functional coverage
- Rapid creation of complex tests
- Modular testbench architecture that provides maximum reuse, scalability and modularity
- Proven verification approach and methodology
- Transaction-level models
- Self-checking tests
- Object oriented interface that allows OOP techniques

1.2 Prerequisites

Familiarity with PCIe, object oriented programming, SystemVerilog, and the current version of UVM.

1.3 Online Class Reference HTML Help

For more information on PCIe Verification IP, refer to the Class Reference for Synopsys Verification IP for PCIe, which you can access by opening the following file in a browser:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/index.html

Please note that the search available in the PCIe Class Reference does not support multiple words, Boolean expressions, or wild card searches. Use only single word searches.

1.4 Product Overview

The PCIe UVM VIP is a suite of UVM-based verification components that are compatible for use with SystemVerilog-Compliant testbenches. The Synopsys PCIe VIP suite simulates PCIe transactions using an active agent as defined by the PCIe specification.

The VIP provides a system environment that contains an active device and MAC agent. The agent supports all the functionality normally associated with active UVM components, including the creation of transactions, checking and reporting the protocol correctness, transaction logging and functional coverage.

1.5 Key Features

Following are the main key architectural features of the PCIe VIP.

- Emulates Root Complex and Endpoint
- Full protocol stack:

  Application, Transaction, Link, and Phy layers
- Checkers verify handshakes and functional accuracy at each layer
- Interfaces to capture sent and received packets for external scoreboard
- Error Injection at all levels
- Protocol checks integrated w/ UVM report object
- Extensive debug Aids
  - All states and Primitives available as ascii strings in waveform viewer
  - All signals named as close to the standard as possible
  - Log file similar to common trace formats from Bus Analyzers
  - Integrated with Protocol Analyzer
  - Symbol logger
- Verilog and UVM APIs
- Pre-configured instances
- Few parameters required to initiate transactions
- Full controllability for complex configurations
- Software Applications

    - Built-in Scoreboard
    - Shadow Memory for self-checking
    - Driver
    - Standard PCIe Bus Transactions
    - Requester
    - Background traffic to various memory ranges
    - Completer
        - Automatically handles completions to mem/cfg/io writes & read
- Error Injections
    - Built in, Exceptions provide automated injection, checking, and recovery
    - User defined injections
        - Per transaction
        - Per symbol
- Debug
    - Grey box SystemVerilog Model
    - Key internal signals can be viewed in waveform viewer
    - ASCII string values for internal states
    - Multiple log options
    - UVM reporter
    - Transaction log
    - Symbol log

1.6 Other Supported Features

1.6.1 Requester, Driver, and Completer Applications

PCIe VIP currently supports the following verification functions:

- Functional coverage
- Protocol checking
- Control on delays and timeouts
- Built-in completer memory
- Verification Planner
- Protocol Analyzer
- Shadow memory with application-level scoreboard.

  Note: Shadow memory is limited to default behavior; settings are not changeable.
- Requester and driver applications

1.6.2 Methodology Features

PCIe VIP currently supports the following methodology functions:

- VIP organized as a set of agents and applications
- Analysis ports for connecting the agent to the scoreboard, or any other component
- Error injections via Factory, callback, or transaction item

1.6.3 ECN Support

Engineering Change Notices (ECNs) that have bee implemented in the PCIe VIP are listed in Table 1-1.

Table 1-1 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

2 Installation and Setup

This section leads you through installing and setting up the VC PCIe. When you complete this checklist, the provided example testbench will be operational and the VC PCIe will be ready to use.

The checklist consists of the following major steps:

This chapter contains the following additional topics:

Note If you encounter any problems with installing the VC PCIe, see "Customer Support" on page 18.

2.1 Verifying the Hardware Requirements

The PCIe Verification IP requires a Solaris or Linux workstation configured as follows:

- 1200 MB available disk space for installation
- 16 GB Virtual memory (recommended)
- FTP anonymous access to ftp.synopsys.com (optional)

2.2 Verifying Software Requirements

The VC PCIe is qualified for use with certain versions of platforms and simulators. This section lists software that the VC PCIe requires.

2.2.1 Platform/OS and Simulator Software

- **Platform/OS and VCS**: You need versions of your platform/OS and simulator that have been qualified for use. To see which platform/OS and simulator versions are qualified for use with the PCIe VIP, check the support matrix for "SVT-based" VIP in the following document:

  Support Matrix for SVT-Based Synopsys PCIe VIP is in:

  Synopsys PCIe Release Notes

2.2.2 Synopsys Common Licensing (SCL) Software

- The SCL software provides the licensing function for the VC PCIe. Acquiring the SCL software is covered here in the installation instructions in “Licensing Information” on page 26.

2.2.3 Other Third Party Software

- **Adobe Acrobat**: VC PCIe documents are available in Acrobat PDF files. You can get Adobe Acrobat Reader for free from http://www.adobe.com.
- **HTML browser:** VC PCIe 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)

2.3 Preparing for Installation

1. Set DESIGNWARE_HOME to the absolute path where Synopsys PCIe VIP is to be installed:

   ```
   setenv DESIGNWARE_HOME absolute_path_to_designware_home
   ```

2. Ensure that your environment and PATH variables are set correctly, including:
   - DESIGNWARE_HOME/bin – The absolute path as described in the previous step.
   - LM_LICENSE_FILE – The absolute path to a file that contains the license keys for your third-party tools. Also, include the absolute path to the third party executable in your PATH variable.

     % setenv LM_LICENSE_FILE <my_license_file|port@host>
   - SNPSLMD_LICENSE_FILE – The absolute path to a file that contains the license keys for Vera and Synopsys Common Licensing software or the *port@host* reference to this file.

     % setenv SNPSLMD_LICENSE_FILE $LM_LICENSE_FILE

2.4 Downloading and Installing

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 available, 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

2.4.1 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 and Password.
3. Click "Sign In" button.
4. Choose "Verification Compiler Verification IP" from the list of available products under "My Product Releases"
5. Select the Product Version from the list of available versions.
6. Click the "Download Here" button for HTTPS download.
7. After reading the legal page, click on "Yes, I agree to the above terms".
8. Click the download button(s) next to the file name(s) of the file(s) you wish to download.
9. Follow browser prompts to select a destination location.
10. You may download multiple files simultaneously.

Note
The Protocol Analyzer is not included in the PCIe VIP download. It is a separate download, which you can get using the procedure above and selecting the Protocol Analyzer file, vip_pa_*version*_run, in step 8.

2.4.2 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.

If you are unable to download the Verification IP using above instructions, refer to “Customer Support” section to obtain support for download and installation.

2.5 What’s Next?

The remainder of this chapter describes the details of the different steps you performed during installation and setup, and consists of the following sections:

- "Licensing Information" on page 26
- "Environment Variable and Path Settings" on page 27
- "Determining Your Model Version" on page 27
- "Integrating a VC VIP into Your Testbench" on page 28

2.6 Licensing Information

The PCIe uses the Synopsys Common Licensing (SCL) software to control its usage. You can find general SCL information at:

http://www.synopsys.com/keys

Attention Licensing is required if the VIP component classes are instantiated in the design. This includes envs, agents, drivers, monitors, sequencers, and components in UVM and OVM. This includes groups, subenvs, and transactors in VMM.

The Synopsys PCIe VIP uses a licensing mechanism that is enabled by the following license features which includes Gen3, Gen4 and Gen5.

- Gen1–Gen2: `VIP-PCIE-SVT, VIP-SOC-LIBRARY-SVT`
- Gen1–Gen3: `VIP-PCIE-SVT + VIP-PCIE-G3-OPT-SVT, VIP-SOC-LIBRARY-SVT`
- Gen1–Gen4: `VIP-PCIE-G4-SVT,` Gen1–Gen5: `VIP-PCIE-G5-SVT, VIP-PCIE-G5-BETA-SVT`

Only one license is consumed per simulation, regardless of how many PCIe VIP models are instantiated in the design. Each of the above features can also be enabled by VIP Library license. For more details, see *VC VIP Library Release Notes*

Note the following:

- When G3 is being used, the VIP will consume the `VIP-PCIE-G3-OPT-SVT` license key in addition to `VIP-PCIe-SVT` or `VIP-LIBRARY-SVT + DesignWare-Regression`
- When G4 is being used, the VIP will consume `VIP-PCIE-G4-SVT` key or `VIP-LIBRARY-SVT + DesignWare-Regression`
- When G5 is being used, the VIP will consume the `VIP-PCIE-G5-SVT` or `VIP-LIBRARY-SVT + DesignWare-Regression + VIP-PCIE-G5-BETA-SVT`

The licensing key must reside in files that are indicated by specific environment variables. For information about setting these licensing environment variables, refer to "Environment Variable and Path Settings" on page 27.

2.6.1 Controlling License Usage

Using the DW_LICENSE_OVERRIDE environment variable, you can control which license is used as follows.

To use only DesignWare-Regression and VIP-LIBRARY-SVT licenses, set DW_LICENSE_OVERRIDE to:

DesignWare-Regression VIP-LIBRARY-SVT

To use only a VIP-PCIE-SVT license, set DW_LICENSE_OVERRIDE to:

VIP-PCIE-SVT

If DW_LICENSE_OVERRIDE is set to any value and the corresponding feature is not available, a license error message is issued.

2.6.1.1 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.

2.6.1.2 Simulation License Suspension

All Synopsys Verification IP 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.

2.7 Environment Variable and Path Settings

The following are environment variables and path settings required by the PCIe verification models:

- DESIGNWARE_HOME – The absolute path to where the VIP is installed.
- SNPSLMD_LICENSE_FILE – The absolute path to a file that contains the license keys for Synopsys Common Licensing software or the *port@host* reference to this file.
- LM_LICENSE_FILE – The absolute path to a file that contains the license keys for your third-party tools. Also, include the absolute path to the third party executable in your PATH variable.

2.7.1 Simulator-Specific Settings

Your simulation environment and PATH variables must be set as required to support your simulator.

2.8 Determining Your Model Version

The following steps tell you how to check the version of the models you are using.

Note: Verification IP products are released and versioned by the suite and not by individual model. The version number of a model indicates the suite version.

- To determine the versions of VIP models installed in your $DESIGNWARE_HOME tree, use the setup utility as follows:

  ```
  % $DESIGNWARE_HOME/bin/dw_vip_setup -i home
  ```

- To determine the versions of VIP models in your design directory, use the setup utility as follows:

  ```
  % $DESIGNWARE_HOME/bin/dw_vip_setup -p design_dir_path -i design
  ```

2.9 Integrating a VC VIP into Your Testbench

After you have installed the VIP, you must set up the VIP for project and testbench use. All Verification Compiler VIP suites contain various components such as transceivers, masters, slaves, and monitors depending on the protocol. The setup process gathers together all the required component files you need to incorporate into your testbench and simulation runs.

You have the choice to set up all of them, or only specific ones. For example, the PCIe VIP contains the following components.

- pcie_device_agent_svt: This is the name used for the entire set of sub-models.
- pcie_global_shadow_svt
- pcie_cfg_database_svt
- pcie_driver_app_svt
- pcie_io_target_svt
- pcie_mac_agent_svt
- pcie_mem_target_svt
- pcie_requester_app_svt
- pcie_target_app_svt
- pcie_tl_svt
- pcie_dl_svt
- pcie_pl_svt

You can set up either an individual component, or the entire set of components within one protocol suite. Use the Synopsys provided tool called dw_vip_setup for these tasks. It resides in $DESIGNWARE_HOME/bin. To get help on dw_vip_setup, invoke the following:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup --help
```

Notes for UVM Users

1. UVM users must set the value of the UVM_PACKER_MAX_BYTES macro to 8192 to ensure that a large enough internal buffer is set up to pack and unpack the data structure, since the default buffer size is only 4Kb. This is can be done in a run script or on the command line, as follows:

   ```
   +define+UVM_PACKER_MAX_BYTES=8192
   ```

   This sets the internal buffer size to the minimum size needed for the PCIe VIP suite. If other suites are being used which require a larger buffer, then a larger size must be specified.
2. If you are using UVM version 1.1b, 1.1c, or 1.1d, then you must define the UVM_DISABLE_AUTO_ITEM_RECORDING macro. Since PCIe is a pipelined protocol (that is, the previous transaction does not necessarily complete before another transaction is started), the PCIe VIP handles triggering of the begin and end events and transaction recording. The PCIe VIP does not use the UVM automatic transaction begin and end event triggering feature. If UVM_DISABLE_AUTO_ITEM_RECORDING is not defined the VIP issues a fatal error.

   If you are using UVM version 1.2 or newer, you do not need to set the macro.

   When the UVM_DISABLE_AUTO_ITEM_RECORDING macro is set, recording is disabled for all VIPs in the design.

The following command adds the full model to the design_dir directory.

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -path /tmp/design_dir -add pcie_device_agent_svt
```

This command sets up all the required files in /tmp/design_dir. The dw_vip_setup utility creates three directories under design_dir which contain all the necessary model files. Files for every VIP are included in these three directories.

- **examples**. Each VIP includes example testbenches. The dw_vip_setup utility adds them in this directory, along with a script for simulation. If an example testbench is specified on the command line, this directory contains all files required for model, suite, and system testbenches.
- **include**. Language-specific include files that contain critical information for VC models. This directory "include/sverilog" is specified in simulator commands to locate model files.
- **src**. Synopsys-specific include files This directory "src/sverilog/vcs" must be included in the simulator command to locate model files.

Note that some components are “top level” and will setup the entire suite. You have the choice to set up the entire suite, or just one component such as a monitor.

Attention There *must* be only one design_dir installation per simulation, regardless of the number of Synopsys Verification and Implementation VIPs you have in your project. Create this directory in $DESIGNWARE_HOME.

2.9.1 Installing and Setting Up More than One VIP Protocol Suite

All VIPs for a particular project must be set up in a single common directory once you execute the *.run file. You may have different projects. In this case, the projects can use their own VIP setup directory. However, all the VIPS used by that specific project must reside in a common directory.

The examples in this chapter call that directory design_dir but you can use any name. In this example, assume you will use the pcie_svt and AXI VIP suites in the design. Your $DESIGNWARE_HOME contains both pcie_svt and AXI VIPs.

First, install a pcie_svt example into the design_dir directory. After the pcie_svt example has been installed, the VIP suite must be set up in and located in the same design_dir directory as the pcie_svt VIP. Use the following commands to perform those steps:

```
// First install the pcie_svt intermediate UVM example:
% $DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -e
      pcie_svt/tb_pcie_svt_uvm_unified_vip_sys -svtb

// Add AXI to the same design_dir as the pcie_svt:
% $DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -add axi_system_env_svt -svlog
```

By default, all of the VIPs use the latest installed version of SVT. Synopsys maintains backward compatibility with previous versions of SVT. As a result, you may mix and match models using previous versions of SVT.

2.9.1.1 Getting Help on Example Run/make Scripts

You can get help on the generated make/run scripts in the following ways:

1. Invoke the run script with no switches, as in:

```
run_pcie_svt_uvm_basic_sys
  usage:  run_pcie_svt_uvm_basic_sys [-32] [-verbose] [-debug] [-waves] [-clean] [-
nobuild] [-norun] [-pa] <scenario> <simulator>
  where   <scenario> is one of:   all base_completer_callback_pipe_test
base_completer_callback_pma_test base_completer_callback_serdes_test base_pipe_test
base_pma_test base_serdes_test directed_pipe_test directed_pma_test
directed_serdes_test requester_traffic_pipe_test
<simulator> is one of:  vcsvlog vcsmxvlog mtivlog vcsmxpcvlog vcsmxpipvlog ncvlog
vcspcvlog
          -32       forces 32-bit mode on 64-bit machines
          -verbose  enable verbose mode during compilation
          -debug    enable debug mode for SVT simulations
         -waves    [fsdb|verdi|dump] enables waves dump and optionally opens viewer (VCS
only)
          -clean    clean simulator generated files
          -nobuild  skip simulator compilation
          -norun    exit after simulator compilation
          -pa       invoke PA after execution
```

2. Invoke the make file with help switch as in:

```
gmake help
Usage: gmake USE_SIMULATOR=<simulator> [VERBOSE=1] [DEBUG=1] [FORCE_32BIT=1]
[WAVES=fsdb|verdi|dump] [NOBUILD=1] [PA=1] [<scenario> ...]
Valid simulators are: vcsvlog vcsmxvlog mtivlog vcsmxpcvlog vcsmxpipvlog ncvlog
vcspcvlog
Valid scenarios are:  all base_completer_callback_pipe_test
base_completer_callback_pma_test base_completer_callback_serdes_test base_pipe_test
base_pma_test base_serdes_test directed_pipe_test directed_pma_test
directed_serdes_test requester_traffic_pipe_test
```

Note: you must have PA installed if you use the -pa or PA=1 switches.

2.9.2 Updating an Existing Model

To add an update an existing model, do the following:

1. Install the model to the same location as your other VIPs by setting the $DESIGNWARE_HOME environment variable.
2. Issue the following command using design_dir as the location for your project directory.

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -path  /tmp/design_dir
     -add  <model-name>_svt -svlog
```

3. You can also update your design_dir by specifying the version number of the model.

```
% dw_vip_setup -path design_dir -add <model-name>_svt -v 3.50a model_vmt -v 3.50a
```

2.10 Including and Importing Model Files into Your Testbench

After you set up the models, you must include and import various files into your top testbench files to use the VIP. You must include the file, `import_pcie_svt_uvm_pkgs.svi` that internally imports the existing PCIe UVM top package (`svt_pcie_uvm_pkg`) and individual sub-packages for partition compile at user testbench. Following is a code list of the includes and imports for PCIe:

```
/* include uvm package before VIP includes,  If not included elsewhere*/
`include "svt_pcie.uvm_pkg"

/** Include the PCIE SVT UVM package */
`include "svt_pcie.uvm.pkg"

/** Defines required for PCIE SVC */
`define EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH  test_top.global_shadow0

/** Import UVM Package */
import uvm_pkg::*;
`include "uvm_macros.svh"

/** Imports SVT PCIe Packages */
`include "import_pcie_svt_uvm_pkgs.svi"

/** Include Util parms */
`include "svc_util_parms.v"

/** Include specific pcie_svt files
`include "svt_pcie_defines.svi"
`include "svt_pcie_device_configuration.sv"
`include "svt_pcie_device_agent.sv"
```

You must also include various VIP directories on the simulator command line. Add the following switches and directories to all compile scripts:

- +incdir+<design_dir>/include/verilog
- +incdir+<design_dir>/include/sverilog
- +incdir+<design_dir>/src/verilog/<vendor>
- +incdir+<design_dir>/src/sverilog/<vendor>

Supported vendors are vcs, mti and ncv. For example:

```
+incdir+<design_dir>/src/sverilog/vcs
```

Using the previous examples, the directory `<design_dir>` would be /tmp/design_dir.

2.11 Compile-time and Runtime Options

Every Synopsys provided example has ASCII files containing compile and run time options. The examples for the model are located in:

$DESIGNWARE_HOME/vip/svt/pcie/latest/examples/sverilog/<*test_name*>

The files containing the options are:

- sim_build_options (also vcs_build_options)
- sim_run_options (also vcs_run_options)

These files contain both optional and required switches. For PCIe, following are the contents of each file, listing optional and required switches:

vcs_build_options

Required: +define+UVM_PACKER_MAX_BYTES=8192
Optional: -timescale=1ns/1ps
Required: +define+SVT_PCIE_INCLUDE_USER_DEFINES
Required: +define+SYNOPSYS_SV

Required (for partition compile): +define+SVT_PCIE_OPTIMIZED_COMPILE

Note
`UVM_PACKER_MAX_BYTES` define needs to be set to maximum value as required by each VIP title in your testbench. For example, if VIP title 1 needs `UVM_PACKER_MAX_BYTES` to be set to 8192, and VIP title 2 needs `UVM_PACKER_MAX_BYTES` to be set to 500000, you need to set `UVM_PACKER_MAX_BYTES` to 500000.

vcs_run_options

Required: +UVM_TESTNAME=$scenario
Note: “scenario” is the uvm test name you pass to VCS

3 General Concepts

This chapter describes the usage of the PCIe VIP in a UVM environment, and its user interface. This chapter discusses the following topics:

- "Introduction to UVM" on page 33
- "PCIe UVM Interface" on page 33
- "SVT Service Sequence/Sequencer" on page 37

3.1 Introduction to UVM

UVM is an object-oriented approach. It provides a blueprint for building testbenches using constrained random verification. The resulting structure also supports directed testing.

This chapter describes the usage of the PCIe VIP in UVM environment, and its user interface. Refer to the Class Reference HTML for a description of attributes and properties of the objects mentioned in this chapter.

This chapter assumes that you are familiar with SystemVerilog and UVM. For more information:

- For the IEEE SystemVerilog standard, see:
  - IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language
- For essential guides describing UVM as it is represented in SystemVerilog, along with a Class Reference, see www.accellera.org.

3.2 PCIe UVM Interface

The Verification Compiler UVM VIP for PCIe is a suite of advanced verification components and data objects based on SystemVerilog UVM-compliant technology. The Verification Compiler UVM PCIe VIP is based on the following UVM agent architecture and data objects.

svt_pcie_device_agent

The **svt_pcie_device_agent** object defines a UVM agent that contains the following uvm_components for PCIe applications:

- Driver

- Target
- Requester
- IO target
- Memory target
- Configuration database
- Global shadow

The **svt_pcie_device_agent** object contains a UVM agent named svt_pcie_agent. The PCIe UVM subenvironment contains active PCIe applications to send and receive PCIe packets as well as UVM sequencers and sequences. Your testbench will interact mainly with UVM sequencers, which can use either Verification Compiler-provided sequences or your own sequences.

svt_pcie_agent

The **svt_pcie_agent** object defines a UVM agent that contain a layered stack of uvm_components for the Physical, Link, and Transaction layers of the PCIe protocol. The PCIe UVM agent contains active PCIe Physical, Link, and Transaction uvm_components, as well as UVM sequencers and sequences. Your testbench will interact mainly with UVM sequencers which can use either Verification Compiler-provided sequences, or your own sequences.

3.2.1 UVM Components of the PCIe Device Subenvironment

- **svt_pcie_driver_app** – A uvm_component object that implements the PCIe Driver application, which transmits PCIe packets to PCIe transaction layer.
- **svt_pcie_requester**_app – A uvm_component object that implements the PCIe Requester application, which supports generating Memory reads and writes towards the programmed Memory segment.
- **svt_pcie_target**_app – A uvm_component object that implements the PCIe Target application, which is responsible for responding to received requests by generating completion packets.
- **svt_pcie_io_target** – A uvm_component object that supports the IO segment of the PCIe system.
- **svt_pcie_mem_target** – A uvm_component object that supports the Memory segment of the PCIe system.
- **svt_pcie_cfg_database** – A uvm_component object that supports the Configuration space of the PCIe system.
- **svt_pcie_global_shadow** – A uvm_component object that implements the PCIe Device system IO, Memory and Configuration spaces.

3.2.2 UVM Components of the PCIe MAC Agent

- **svt_pcie_tl** – A uvm_component object that implements the PCIe Transaction Layer.
- **svt_pcie_dl** – A uvm_component object that implements the PCIe Link Layer.
- **svt_pcie_pl_phy** – A uvm_component object that implements the PCIe Physical Layer.

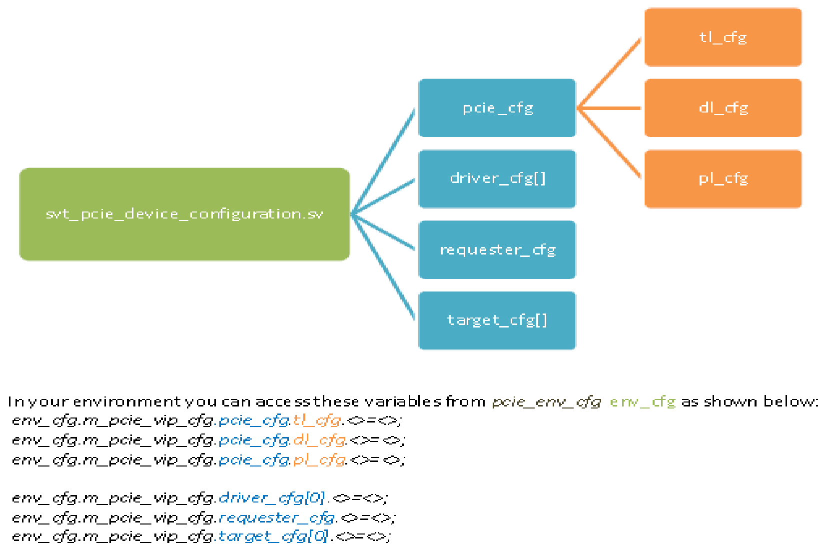
3.2.3 Configuration Data Objects

Configuration data objects are abstracted data objects that represent the content of PCIe VIP configuration data and protocol transactions. The top-level configuration data objects are:

- svt_pcie_device_configuration
    - svt_pcie_driver_app_configuration
    - svt_pcie_requester_app_status
    - svt_pcie_target_app_configuration
    - svt_pcie_configuration
        - svt_pcie_tl_configuration
        - svt_pcie_dl_configuration
        - svt_pcie_pl_configuration

The following illustration shows the inheritance diagram for all the configuration objects.

Figure 3-1 Inheritance Diagram for All Configuration Objects



3.2.4 Status Data Objects

Status data objects are abstracted data objects that represent the content of PCIe VIP statistics. Registered status objects are updated in realtime. Separate statistics are kept per layer/application. Status objects are useful for:

- functional coverage
- reporting testcase progress
- debug

The top-level status data objects are:

- svt_pcie_device_status
    - svt_pcie_requester_app_status
    - svt_pcie_target_app_status
    - svt_pcie_io_target_status
    - svt_pcie_mem_target_status

- svt_pcie_status
  - svt_pcie_tl_status
  - svt_pcie_dl_status
  - svt_pcie_pl_status

Following are some examples of the type of status data you can obtain.

- Target Application

  # bytes received

  # bytes sent

  # msg cpl sent

  # num tlps received
- Phy Layer

  #LTSSM state

  # hot resets initialized
- Link Layer

  # ack received

  # bad tlp sent

  # EI RX TLP withhold ack / nack
- Transaction Layer

  # bad TLPs received

  # TC5 TLPs sent

An example of waiting for link activation:

```
svt_pcie_device_status stat;
stat = svt_pcie_device_status::type_id::create("stat");

// wait for link activation
wait(stat.port_status.pl_status.link_up == 1'b1);
```

3.2.5 Sequence Item Data Objects

The VIP supports extending UVM sequence item data classes for customizing randomization constraints. This allows you to disable some reasonable_* constraints and replace them with constraints appropriate to your system. Individual reasonable_* constraints map to independent fields, each of which can be disabled. The following are the sequence data item classes:

- svt_pcie_driver_app_transaction
- svt_pcie_io_target_service
- svt_pcie_mem_target_service
- svt_pcie_cfg_database_service
- svt_pcie_global_shadow_service
- svt_pcie_driver_app_service

- svt_pcie_requester_app_service
- svt_pcie_target_app_service
- svt_pcie_tl_service
- svt_pcie_dl_service
- svt_pcie_pl_service

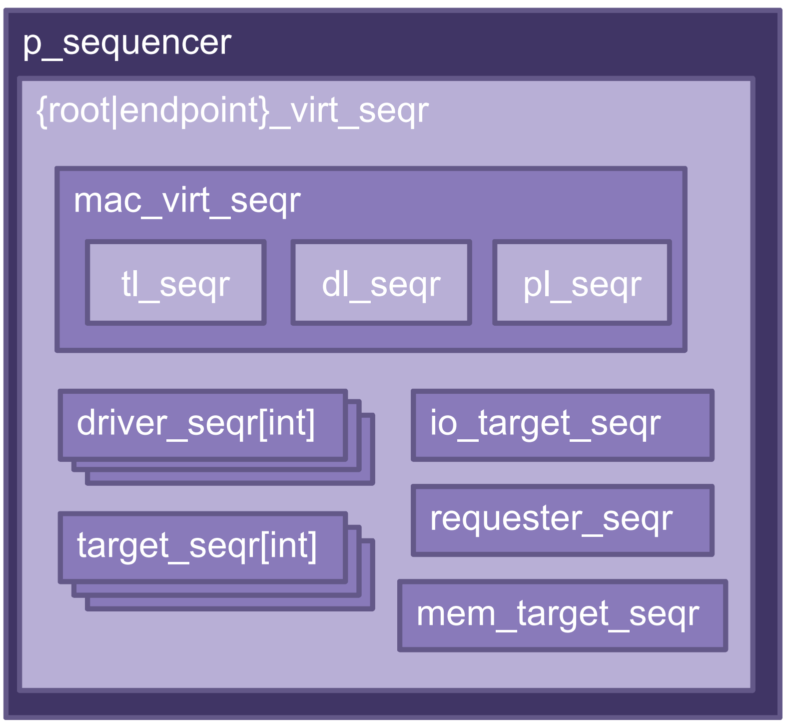
3.3 SVT Service Sequence/Sequencer

Each component in the agent has its own service sequencer. All SVT sequences are derived from uvm_sequence. Sequences or individual sequence items are executed on the appropriate sequencer.

A *p_sequencer* is instantiated with a macro in the top sequence as shown below:

```
`uvm_declare_p_sequencer
 (svt_pcie_device_system_virtual_sequencer
```

The following figure shows the elements of the `p_sequencer`:



4 Verification Features

This chapter describes the various verification features available with the Synopsys PCIe Verification IP. This chapter discusses the following topics:

4.1 The Transaction Logger

All inbound and outbound transactions to or 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, use the enable_transaction_logging member of the svt_pcie_configuration class, as shown in the following example:

```
class example extends uvm_test;
. . .
  virtual function void task build_phase(uvm_phase phase);
    super.build_phase(phase);
    . . .
endpoint_cfg.port_cfg.enable_transaction_logging = 1;
 endpoint_cfg.port_cfg.transaction_log_filename = "pcie_trans.log";
 . . .
endfunction
endclass
```

The transaction log filename will be appended to the full hierarchical name of the port0 instance generating it.

4.1.1 Printing TLP Payload Data to a Transaction Log File

The Synopsys provided intermediate example shows how to print TLP payload data to the Transaction log. You must first enable transaction logging. By default it is off. Also, set the filename of the transaction log.

```
class pcie_shared_cfg extends uvm_object;
...
  /** Setup the PCIE device system default values */
  function void setup_pcie_device_system_defaults();
    begin
...
      root_cfg.pcie_cfg.enable_transaction_logging = 1'b1;
      root_cfg.pcie_cfg.transaction_log_filename = "transaction.log";

      root_cfg.pcie_cfg.enable_symbol_logging = 1'b1;
      root_cfg.pcie_cfg.symbol_log_filename = "symbol.log";
```

Next, set how many dwords of the payload you want the model to write into the transaction log file.

```
class pcie_device_base_test extends uvm_test;
...
  virtual function void build_phase(uvm_phase phase);
    `uvm_info("build_phase", "Entered...", UVM_LOW)
    super.build_phase(phase);

...
    /** Set the payload display limit */
    svt_pcie_dl_disp_pattern::default_max_payload_print_dwords = 1024;
```

Note the default is zero. In the example, it has been set to 1024.

4.1.2 Fields of the Transaction Log Header

The fields of the transaction log header are described in this section. These fields are listed from left to right as they appear on the header.

Field: Reporter

Description:

This field represents an instance of the VIP in the test environment. The transaction log information is reported for this VIP instance.

Field: Start Time (ns)

Description:

This field represents the simulation time in ns when the transaction starts.

Field: End Time (ns)

Description:

This field represents the simulation time in ns when the transaction ends.

Field: Dir

Description:

This field represents the direction of the transaction from the VIP instance. "T" represents a transmit transaction. "R" represents a receive transaction.

Field: TLP Type

DLLP Type

Description:

This field represents the type of TLP (Transaction Layer Packet) or DLLP (Data Link Layer Packet) as defined in Table 2-3 and Table 3-1 of the PCIe Specification respectively. For TLP memory read (MRd) and memory write (MWr) packets, a "32" or "64" is appended to the type. This number represents a 32-bit or 64-bit memory addressing.

For example:

MRd32

MWr64

Field: R_ID /Tag | ST

Description:

This field has 2 different representations for a TLP transaction. The Requester ID and Tag are displayed, or the Steering Tag is displayed. This field is blank for DLLP transactions.

TLP Field	Description
R_ID/Tag	Requester ID/Tag
ST	Steering tag

For example:

TLP

Reporter	Start Time (ns)	End Time (ns)	Dir	TLP Type DLLP Type	R_ID/Tag | ST
vip0	133328.00	133340.00	T	**CfgRd0**	**0x0001/13**
vip0	159140.00	159160.00	T	**MRd32**	**0x0001/1f**

DLLP

Reporter	Start Time (ns)	End Time (ns)	Dir	TLP Type DLLP Type	R_ID/Tag | ST

vip0	133328.00	133328.00	R	**INITFC2_P_VC0**
vip0	133772.00	133772.00	R	**ACK**

Field: Seq Num

Description:

This field represents the sequence number of the transaction.

Field: TC VC

Description:

This field represents the value of the Traffic Class field of the TLP.

Field: TH

Description:

This field represents the 1-bit TH field of the common TLP packet header. The TH field is an indication of the TLP Processing Hints (TPH) and the Optional TPH TLP Prefix when applicable presented in the TLP header.

Field: PH

Description:

This field represents processing hint.

Field: IDO RO

Description:

These 2 fields represent the Ordering Attribute Bits as defined in Table 2-10 of the PCIe Specification. The table is shown below.

Attribute Bit [2] (IDO)	Attribute Bit [1] (RO)	Ordering Type	Ordering Model
0	0	Default Ordering	PCI Strongly Ordered Model
0	1	Relaxed Ordering	PCI-X Relaxed Ordering Model
1	0	ID-Based Ordering	Independent ordering based on Requester/Completer ID
1	1	Relaxed Ordering plus ID-Based Ordering	Logical "OR" of Relaxed Ordering and IDO

Field: NS

Description:

This field represents the No Snoop bit value of the TLP.

Field: Address

Reg#/MsgRt/Cpl

HdrFC DataFC

Description:

This field has multiple representations. For a TLP transaction, the value(s) displayed depends on the TLP type as shown in the following table. For a DLLP transaction, the Flow Control header and data are displayed.

TLP Field	Description
< address >	**Memory request:** This field represents the memory address.
< address >	**IO request:** This field represents the IO address.
BDF: < > R: < > or BDF: < > O: < >	**Configuration request:** "BDF" represents the bus device function. "R" represents the register number. "O" represents the register byte offset. This offset is not displayed by default. To enable the display, you must set the dl_trace_options[1] attribute of the PCIe configuration class (svt_pcie_configuration). For example: <agent cfg>.pcie_cfg.dl_trace_options[1] = 1
< message >	**Message request:** This field represent the message routing as defined in Table 2-18 of the PCIe Specification.
ID: < > Stat: < >	**Completion request:** "ID" represents the Completion ID. "Stat" represents the Completion status as defined in Table 2-29 of the PCIe Specification. The status are SC, UR, CRS and CA.

DLLP Field	Description
< header > < data >	Flow Control header and data

For example:

TLP:

Reporter	Start Time (ns)	End Time (ns)	Dir	TLP Type DLLP Type	R_ID/Tag | ST	Seq Num	TC VC	T H	P H	I D O	R O	N S	Address Reg#/MsgRt/Cpl HdrFC DataFC
vip0	159140.00	159160.00	T	MRd32	0x0001/1f	139	0	0		0	0	0	**0x00245a28**

vip0	133500.00	133520.00	T	CfgWr0	0x0001/1c	2	0	0		0	0	0	**BDF:0x0000 R:0x004**
vip0	133304.00	133324.00	T	MsgD	0x0001/18	0	0	0		0	0	0	**Local Term Rcvr ID:0x0002 Stat:SC**
vip0	158868.00	158872.00	R	CplD	0x0001/1e	136	0	0		0	0	0	**BC:0004**

DLLP

Reporter	Start Time (ns)	End Time (ns)	Dir	TLP Type DLLP Type	R_ID/Tag | ST	Seq Num	TC VC	T H	P H	I D O	R O	N S	Address Reg#/MsgRt/Cpl HdrFC	DataFC
vip0	133596	133596	R	ACK		2							**116**	**230**
vip0	133124	133124	T	INITFC1_P_VC0									**102**	**1024**

Field: BE | ST

BC

MCode

Description:

This field has multiple representations of a TLP transaction.

TLP Field	Description
< byte enable >	**Memory request/IO request/Configuration request:** This field represents the byte enable.
< steering tag >	**Memory request:** When the TH field has a value of "1", this field represents the steering tag value.
BC: < >	**Completion request:** BC represents the byte count.
< message code >	**Message request:** This field represent the message code as defined in Table F-1 of the PCIe Specification.

For example:

Reporter	Start Time (ns)	End Time (ns)	Dir	TLP Type DLLP Type	R_ID/Tag | ST	Seq Num	TC VC	T H	P H	I D O	R O	N S	Address Reg#/MsgRt/Cpl HdrFC DataFC	BE | ST BC Mcode
vip0	133304.00	133324.00	T	MsgD	0x0001/18	0	0	0		0	0	0	Local Term Rcvr	**0x50**
vip0	158896.00	159124.00	T	MWr32	0x0001/09	138	0	0		0	0	0	0x00245a28	**1 c**
vip0	158868.00	158872.00	R	CplD	0x0001/1e	136	0	0		0	0	0	ID:0x0002 Stat:SC	**BC:0004**

Field: Len/Idx

DW

Description:

This field has 2 representations of a TLP transaction.

TLP Field	Description
< payload length >	For the TLP Header displayed on the first row of data, this field represents the length of the payload in double word (DW).
< data index >	For the TLP payload displayed from the second row of data and on, this field represents the accumulative count of DW data.

For example:

MWr32

Len/Idx DW	Prefix / Header / Data (All values in Hex)			
221	H400000dd	H0001091c	H00245a28	---
0	250e4795	e0b18822	9c1a2b30	a6824000
4	48105945	cafb12dd	8ed6f96b	df547dae
8	17505659	31998267	b37c242c	cc4f5f10
		< data continues >		
216	0a3336db	36bb1e9b	df73a4c9	43d8486b
220	00693366	---	---	---

In the above example, the "221" on the first row is the TLP payload length. The "0" through "220" on the subsequent rows are the data index.

CfgRd0

Len/Idx DW	Prefix / Header / Data (All values in Hex)			
1	H04000001	H00011e0f	H00020000	---

CplD

Len/Idx DW	Prefix / Header / Data (All values in Hex)			
1	H4a000001	H00010004	H00011500	---
0	06001000			

Field: Prefix / Header / Data

(All values in Hex)

Description:

This field represents the raw dword values of a TLP transaction.

- Values with an "(H)" prefix represent header DWORDs.
- Values with a "(P)" prefix represent the PCIe Prefix DWORDS.
- When the model is configured to show transaction data, values with a "(D)" prefix represent payload DWORDS.

By default, only the TLP header data is displayed along with the header fields. You can enable the display of payload data by using one of the following methods:

a. In the build phase of the simulation, set the static attribute "default_max_payload_print_dwords" of class "svt_pcie_dl_disp_pattern" to the default maximum number of payload DWORDs to be displayed.

b. Set the "SVT_PCIE_XACT_LOG_MAX_PAYLOAD_DWORDS_DEFAULT" macro to the default maximum number of payload DWORDs to be displayed.

Payload data for CfgWr and CfgRd (corresponding CplD) are displayed with a single DWORD. By default, this DWORD value is displayed in the Big Endian format. Alternatively, you can enable the DWORD to be displayed in the Little Endian format by setting the "dl_trace_options[0]" attribute of the PCIe configuration class (svt_pcie_configuration).

For example:

Default Big Endian payload data

0 06001000

Enable Little Endian format

<agent cfg>.pcie_cfg.dl_trace_options[0] = 1;

Little Endian payload data

0 00100006 LittleEndian

Field: EP

Description:

This field represents the poison bit as defined in the PCIe Specification.

Field: ECRC

Description:

This field represents the ECRC value of a TLP as defined in the PCIe Specification.

Field: LCRC

CRC

Description:

This field has 2 representations. For a TLP transaction, the LCRC value as defined in the PCIe Specification is displayed. For a DLLP transaction, the CRC value as defined in the PCIe Specification is displayed.

Field: TX/RX

Error

Description:

This field represents the type of error injection when error injection is enabled in a transmit (tx) transaction. For a receive (rx) transaction, this field represents the detected error.

Error Injection Type	Description
BadSeq	Illegal Sequence Number
CodeViol	TX Code Violation
CrcEr	LCRC LCRC Error
Disparty	Disparity Error
DupSeq	Duplicate Sequence Number
EIErr	Scenario injects error, then vip reported this.
HdrCRC	PCIE 8G Header CRC Error
HdrPAR	PCIE 8G Header Parity Error
LCRC	LCRC Error
NAK	NAK Received for TLP
NoACK	Missing ACK for Transaction
NoEND	Missing END
NoSTART	Missing START
NoSTP	NoSTART Missing START"
NullLCRC	Nullified TLP with corrupt CRC
NullTLP	Nullified TLP
ReplCnt	Replay count of 4 exceeded

4.2 The Symbol Logger

One log file is created per simulation. All agents share the log file. Each agent must be enabled independently as shown in Example 4-1. If more than one symbol_log_filename is set, then the last one set within the simulation serves as the filename. It is recommended that you have only one agent set the filename.

Example 4-1

```
class example extends uvm_test;
   . . .
```

```
   virtual function void task build_phase(uvm_phase phase);
      super.build_phase(phase);
      . . .
      root_cfg.pcie_cfg.enable_symbol_logging = 1;               // Enable for RC
      endpoint_cfg.prot_cfg.enable_symbol_logging = 1;           // Enable for EP
      root_cfg.pcie_cfg.symbol_log_filename = "symbol.log"; // Recommended to only set
the filename once
      . . .
   endfunction
endclass
```

The symbol log filename will be appended to the full hierarchical name of the port0 instance generating it.

4.2.1 Fields of the Symbol Log Header

The fields of the symbol log header are described in this section. These fields are listed from left to right as they appear on the header.

Field: TIME

Description:

This field represents the simulation time in ns. Symbol logging is performed at the PIPE interface. If the PIPE interface is configured as multiple bytes, all bytes transferred at a time step are logged at the same time step.

Field: INSTANCE

Description:

This field represents an instance of the VIP in the test environment. The symbol log information is reported for this VIP instance.

Field: < lane symbols >

Description:

This field represents symbols on the active lane(s). The format of the field header is:

R00 [R01] [R02] … [R<n>] | T00 [T01] [T02] … [T<n>]

Where, "R" represents the receive symbol on the lane. "T" represents the transmit symbol on the lane. <n> is the number of the highest configured lane.

The encoding of the lane symbols are listed in the following tables.

Table 4-1 Special Symbol Encodings for All Link Data Rates

Symbol	Description
z	Electrical idle
?	Invalid or unknown value

Table 4-1 Special Symbol Encodings for All Link Data Rates

Symbol	Description
.	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 time steps at either 2.5 GT/s 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 8GT/s, symbols after the sync headers are prepended with encodings of the sync headers listed in Table 4-2. Additional encodings for link operation at 8GT/s are listed in Table 4-3.

Table 4-2 Sync Header Encodings for Link Operation at 8GT/s

Symbol	Description
@	2'b'00 (Reserved)
*	2'b'01 (OS block)
=	2'b'10 (Data block)
$	2'b'11 (Reserved)

Table 4-3 Special Character Encodings for Link Operation at 8GT/s

Symbol	Description
::	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.

Field: LTSSM State

Description:

This field represents the state of the LTSSM state machine as defined in section 4.2.5 of the PCIe specification. For states such as L0 and L0s where the receive (rx) and transmit (tx) LTSSM states may diverge, the rx and tx states are displayed separately. Otherwise, only a single state is displayed for both rx and tx states.

For example:

```
TIME     INSTANCE               R00    |    T00    ->    LTSSM State
--------------------------------------------------------------------------------------------
  208:   root0                    z    |     z     ->    initializing
  208:   endpoint0                z    |     z     ->    initializing
  212:   root0                    z    |     z     ->    initializing
  212:   endpoint0                z    |     z     ->    initializing
root0                     --  Detected change in link width from 1 to 4.

TIME     INSTANCE               R00   R01   R02   R03   |   T00   T01   T02   T03   ->   LTSSM State
-----------------------------------------------------------------------------------------------------------------------
  216:   root0                    z     z     z     z   |     z     z     z     z   ->   Det.Quiet
endpoint0                 --  Detected change in link width from 1 to 4.

TIME     INSTANCE               R00   R01   R02   R03   |   T00   T01   T02   T03   ->   LTSSM State
-----------------------------------------------------------------------------------------------------------------------
  216:   endpoint0                z     z     z     z   |     z     z     z     z   ->   Det.Quiet
  220:   root0                    z     z     z     z   |     z     z     z     z   ->   Det.Quiet
  220:   endpoint0                z     z     z     z   |     z     z     z     z   ->   Det.Quiet
                          < symbol continues >
26840:   endpoint0               00    00    00    00   |    a1    75    47    aa   ->   tx = L0, rx = L0
26841:   root0                   a1    75    47    aa   |    00    00    00    00   ->   tx = L0, rx = L0
26841:   endpoint0               00    00    00    00   |    5b    2c    cd    17   ->   tx = L0, rx = L0
```

4.2.1.1 Configuration Messages in the Symbol Log

In addition to lane symbols, messages that indicate link changes are displayed in the symbol log. These messages are prepended by "--".

For example:

endpoint0 -- Detected change in link width from 1 to 4.

root0 -- Detected change in data rate to 8 Gt/s. See file header for special encodings.

4.2.1.2 Symbol Log with OS Information

The current symbol log format has been enhanced to include OS information. This enhanced format is enabled by setting the configuration attribute `svt_pcie_configuration::enable_enhanced_symbol_log_format` to 1. When the enhanced format is enabled, the symbol log will also contain information about when and what OS are terminated on individual lanes. The default value of `enable_enhanced_symbol_log_format` is 0.

For most OS's this would be on the last symbol of the OS, but for some OS's that have indeterminate length like SKP OS the completion information would be associated with the COM/sync header of the next OS/data block.

Example 4-2

```
12367: root0   COM  COM  COM  COM  |   4a   4a   4a   4a  -> Poll.Active
12371: root0   PAD  PAD  PAD  PAD  |   4a   4a   4a   4a  -> Poll.Active
12375: root0   PAD  PAD  PAD  PAD  |   4a   4a   4a   4a  -> Poll.Active
12379: root0   ff   ff   ff   ff   |   4a   4a   4a   4a  -> Poll.Active >TX:TS1_OS on L0
```

```
12383: root0   0e   0e   0e   0e   |  COM   4a   4a   4a  -> Poll.Active
12387: root0   00   00   00   00   |  PAD   4a   4a   4a  -> Poll.Active >TX:TS1_OS on L1,3
12391: root0   4a   4a   4a   4a   |  PAD  COM   4a  COM  -> Poll.Active
12395: root0   4a   4a   4a   4a   |   ff  PAD   4a  PAD  -> Poll.Active >TX:TS1_OS on L2
12399: root0   4a   4a   4a   4a   |   0e  PAD  COM  PAD  -> Poll.Active
12403: root0   4a   4a   4a   4a   |   00   ff  PAD   ff  -> Poll.Active
12407: root0   4a   4a   4a   4a   |   4a   0e  PAD   0e  -> Poll.Active
12411: root0   4a   4a   4a   4a   |   4a   00   ff   00  -> Poll.Active
12415: root0   4a   4a   4a   4a   |   4a   4a   0e   4a  -> Poll.Active
12419: root0   4a   4a   4a   4a   |   4a   4a   00   4a  -> Poll.Active
12423: root0   4a   4a   4a   4a   |   4a   4a   4a   4a  -> Poll.Active
12427: root0   4a   4a   4a   4a   |   4a   4a   4a   4a  -> Poll.Active >RX:TS1_OS on L0-3
12431: root0   COM  COM  COM  COM  |   4a   4a   4a   4a  -> Poll.Active
12435: root0   PAD  PAD  PAD  PAD  |   4a   4a   4a   4a  -> Poll.Active
12439: root0   PAD  PAD  PAD  PAD  |   4a   4a   4a   4a  -> Poll.Active
12443: root0   ff   ff   ff   ff   |   4a   4a   4a   4a  -> Poll.Active >TX:TS1_OS on L0
12447: root0   0e   0e   0e   0e   |  COM   4a   4a   4a  -> Poll.Active
12451: root0   00   00   00   00   |  PAD   4a   4a   4a  -> Poll.Active >TX:TS1_OS on L1,3
12455: root0   4a   4a   4a   4a   |  PAD  COM   4a  COM  -> Poll.Active
12459: root0   4a   4a   4a   4a   |   ff  PAD   4a  PAD  -> Poll.Active >TX:TS1_OS on L2
12463: root0   4a   4a   4a   4a   |   0e  PAD  COM  PAD  -> Poll.Active
12467: root0   4a   4a   4a   4a   |   00   ff  PAD   ff  -> Poll.Active
```

4.2.2 Synchronization of Simulation Time Between Transaction Log and Symbol Log

Transaction logging times represent times at the periphery of the VIP. Symbol logging times are captured at the PIPE interface, which may be internal or external to the VIP depending on the interface type.

For Serial and PMA interface, there is no correlation between the time displayed in the transaction log and the symbol log. Due to delay through the PHY layer, symbols are logged at a different time than the transaction log for the same packet.

For PIPE interface, the simulation time displayed in the transaction log and symbol log are synchronized for the same packet. The "Start Time" of the transaction log corresponds to the time of a transaction with the "STP" or "SDP" symbol in the symbol log. The "End Time" of the transaction log corresponds to the time of a transaction with the "END" symbol in the symbol log.

Transaction Log	Symbol Log
Start Time (ns)	TIME with "STP" or "SDP" symbol
End Time (ns)	TIME with "END" symbol

Example 1:

Transaction Log

vip0 **16332.00** **16336.00** T INITFC2_P_VC0 101 1025 0xb467

Symbol Log

16332: vip0 00 00 00 00 | **SDP** c0 19 84 -> tx = L0, rx = L0
16336: vip0 00 00 00 00 | 00 b4 67 **END** -> tx = L0, rx = L0

Example 2:

Transaction Log

vip0	**16344.00**	**16476.00**	T	MWr32	0x0001/10	0	0	0	0	0	0	0x797ffe94	7 f	

29	H4000001d	H0001107f	H797ffe94	---	0	0x88146f08
0	ee4f36e1	f7f2dc7e	2409961a	6a0e183a		
4	d39af0b3	7c9a4388	6143237c	ec6e36a2		
8	c56daf5b	05b6af19	49481dfc	036a8986		
12	70425548	b95aea17	91d4a8af	8d575fcc		
16	cbf0a790	ca6403af	196ebb7a	ff400faf		
20	67310374	6745f3f3	677e58c8	09bcfe06		
24	03223d90	ab52c830	df33cf23	700a0f1f		
28	b40b4d71	---	---	---		

Symbol Log

16344: vip0	00	00	00	00	|	**STP**	00	00	40	-> tx=L0, rx=L0
16348: vip0	00	00	00	00	|	00	00	1d	00	-> tx=L0, rx=L0
16352: vip0	00	00	00	00	|	01	10	7f	79	-> tx=L0, rx=L0
16356: vip0	00	00	00	00	|	7f	fe	94	ee	-> tx=L0, rx=L0
16360: vip0	00	00	00	00	|	4f	36	e1	f7	-> tx=L0, rx=L0
16364: vip0	00	00	00	00	|	f2	dc	7e	24	-> tx=L0, rx=L0
	< symbol continues >									
16456: vip0	00	00	00	00	|	22	3d	90	ab	-> tx=L0, rx=L0
16460: vip0	00	00	00	00	|	52	c8	30	df	-> tx=L0, rx=L0
16464: vip0	00	00	00	00	|	33	cf	23	70	-> tx=L0, rx=L0
16468: vip0	00	00	00	00	|	0a	0f	1f	b4	-> tx=L0, rx=L0
16472: vip0	00	00	00	00	|	0b	4d	71	88	-> tx=L0, rx=L0
16476: vip0	00	00	00	00	|	14	6f	08	**END**	-> tx=L0, rx=L0

For more details on Gen5 Symbol Logger, see "Gen5 Symbol Logging" on page 117.

4.3 The MBI Logger

When PIPE spec. version is set to 4.4 or above, all MBI transactions to or from the VIP are sent to the MBI logger. These transactions are distilled and written (one transaction per line) to the MBI log file.

By default, the MBI logger is disabled. To enable it and to start writing transactions to a file, use the `enable_mbi_logging` member of the `svt_pcie_configuration` class.

One log file is created per simulation. All agents share the log file. Each agent must be enabled independently as shown in Example 4-3. If more than one `mbi_log_filename` is set, then the last one set within the simulation serves as the filename. It is recommended that you have only one agent set for the filename.

Example 4-3

```
class example extends uvm_test;
. . .
virtual function void task build_phase(uvm_phase phase);
super.build_phase(phase);
. . .
root_cfg.pcie_cfg. enable_mbi_logging = 1; //Enable for RC
endpoint_cfg.pcie_cfg. enable_mbi_logging = 1; //Enable for EP
root_cfg.pcie_cfg. mbi_log_filename = "mbi.log"; //Recommended to only set
the filename once
```

```
. . .
endfunction
endclass
```

The `mbi` log filename will be appended to the full hierarchical name of the `port0` instance generating it.

4.3.1 Fields of the MBI Log Header

The fields of the MBI log header are described in this section. These fields are listed from left to right as they appear on the header.

Table 4-4 Fields - MBI Log

Field	Description
`Instance`	This field represents an instance of the VIP in the test environment. The mbi log information is reported for this VIP instance.
`Start Time (ns)`	This field represents the simulation time in ns when the MBI transaction starts.
`End Time (ns)`	This field represents the simulation time in ns when the MBI transaction ends.
`Dir`	This field represents the MBI command transmission on M2P message bus or P2M message bus direction.
`Lane`	This field represents lane having the MBI transaction.
`MBI Cmd`	This field represents current active MBI command.
`Address`	This field represents current active MBI command's address.
`Data`	This field represents data associated with current active MBI transaction.

MBI Log Illustration:

```
-------------------------------------------------------------------------------------------------
| Instance           | Start Time |  End Time  | Dir | Lane |        MBI Cmd        | Address | Data |
|--------------------|------------|------------|-----|------|-----------------------|---------|------|
| root_port          | 222.000000 | 230.000000 | M2P |  0   |  WRITE_UNCOMMITTED    |   402   |  1 |
| root_port          | 234.000000 | 242.000000 | P2M |  0   |  WRITE_UNCOMMITTED    |   403   |  30|
| root_port          | 234.000000 | 242.000000 | M2P |  0   |  WRITE_UNCOMMITTED    |   403   |  0 |
| root_port          | 246.000000 | 254.000000 | P2M |  0   |   WRITE_COMMITTED     |   404   |  18|
| root_port          | 246.000000 | 254.000000 | M2P |  0   |   WRITE_COMMITTED     |   404   |  0 |
| root_port          | 262.000000 | 262.000000 | M2P |  0   |      WRITE_ACK        |    0    |  0 |
```

4.4 The CTRL SKP Logger

When the VIP enables support for CTRL-SKP OS at 16G (Gen4 or above) or Rx Margining or Fault Isolation, the CTRL SKP OS transaction information is captured by the CTRL SKP Logger.

It indicates the logging of CTRL SKP command information handshake in L0 LTSSM state by the Downstream and Upstream port

- at 16GT/s and above for Active Component
- only at 16GT/s for Passive Component

By default, the CTRL SKP Logger is disabled. To enable it and to start writing transactions to a file, use the `enable_ctrl_skp_logging` member of the `svt_pcie_configuration` class. There are separate CTRL SKP log files for the different agents/components and every agent/component with CTRL SKP logging enabled will log CTRL SKP command handshake information into a separate file. Each agent must be enabled independently as shown in Example 5-4. If more than one `ctrl_skp_log_filename` is set, then the last one set within the simulation serves as the filename.

Example 4-4

```
class example extends uvm_test;
. . .
virtual function void task build_phase(uvm_phase phase);
super.build_phase(phase);
. . .
root_cfg.pcie_cfg.enable_ctrl_skp_logging = 1; // Enable for RC
endpoint_cfg.pcie_cfg.enable_ctrl_skp_logging = 1; // Enable for EP
root_cfg.pcie_cfg.ctrl_skp_log_filename = "ctrl_skp_log"; // Recommended to only set
the filename once
. . .
endfunction
endclass
```

The CTRL SKP log filename will be appended to the full hierarchical name of the port0 instance generating it.

4.4.1 Fields of the Ctrl SKP Log Header

The fields of the MBI log header are described in this section. These fields are listed from left to right as they appear on the header.

Table 4-5 Fields - CTRL SKP Log

Field	Description
`Start Time (ns)`	This field represents the simulation time in ns when the Ctrl SKP OS transaction starts.
`Reporter`	This field represents an instance of the VIP in the test environment. The Ctrl SKP log information is reported for this VIP instance.
`Dir`	This field represents the Ctrl SKP OS transaction transmission or reception direction for the Reporter. Where, "R" represents the receive Ctrl SKP OS transaction information on the lane. "T" represents the transmit Ctrl SKP OS transaction information symbol on the lane.
`Lane`	This field represents lane having the Ctrl SKP OS transaction.
`Receiver Number`	This field represents current active Ctrl SKP OS Receiver Number field value per spec 4.2.13.1.
`Margin Type`	This field represents current active Ctrl SKP OS Margin Type field value per spec 4.2.13.1.
`Margin Payload`	This field represents current active Ctrl SKP OS Margin Payload field value per spec 4.2.13.1.
`Command Type`	This field represents current active Ctrl SKP OS Margin Command per spec 4.2.13.1.

CTRL SKP Log Illustration:

```
|------------------------------------------------------------------------------------------------------------------------
| |                                       EP Active Component Report of CTRL SKPs in L0 LTSSM state                      |
|------------------------------------------------------------------------------------------------------------------------
| Start Time            Reporter   DIR    Lane     Receiver    Margin     Margin        Command Type                     |
|                                                  Number      Type       Payload                                        |
|----------------------------------------------------------------------------------------------------------------------
|130632.80170 ns        EP         T      Lane0    'h0         'h7        'h9c          NO_COMMAND
 130649.30170 ns        EP         R      Lane0    'h3         'h1        'h88          REPORT_MARGIN_CONTROL_CAPABLILITY
 136742.80170 ns        EP         T      Lane0    'h3         'h1        'h1f
 136759.30170 ns        EP         R      Lane0    'h0         'h7        'h9c          NO_COMMAND
 142852.80170 ns        EP         T      Lane0    'h0         'h7        'h9c          NO_COMMAND
 142869.30170 ns        EP         R      Lane0    'h3         'h5        'h7           VENDOR_DEFINED
```

4.5 Using Native Protocol Analyzer for Debugging

4.5.1 Introduction

This feature enables you to invoke Protocol Analyzer from Verdi GUI. You can synchronize the Verdi wave window, smart log and the source code with the Protocol Analyzer transaction view. Protocol Analyzer can be enabled in an interactive and post-processing mode. The features available in Native Protocol Analyzer includes layer based grouping of the transactions, Quick filter, Call stack, horizontal zoom and reverse debug with the interactive support.

The VIP supports the Synopsys Protocol Analyzer (PA) tool, which is an interactive graphical application which provides protocol-oriented analysis and debugging capabilitiesallows. It 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 transaction.

The Protocol Analyzer tool supports the following PCIe features:

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

4.5.2 Enabling the Protocol Analyzer

To enable protocol analyzer define the macro 'SVT_PCIE_INCLUDE_AC_PA' at compile time and set following configurations in svt_pcie_configuration class:

- enable_tl_xml_gen. Protocol file generation for the Transaction Layer
- enable_pl_xml_gen. Protocol file generation for the Physical Layer
- enable_dl_xml_gen. Protocol file generation for the Data Link Layer

The default value of each of these variable is 0, which means that protocol file generation is disabled by default.

To enable protocol file generation for a layer, set the value of the protocol generation enabling variable for that layer to 1 in the port configuration of each master or slave for which protocol file generation is desired.

The next time that the environment is simulated, protocol files will be generated according to the port configurations. The protocol files will be in XML format. Import these files into the Protocol Analyzer to view the protocol transactions.

4.5.3 Prerequisites

Protocol Analyzer uses transaction-level dump database. You can use the following settings to dump the transaction database:

4.5.4 Compile-Time Options

The following compile-time options must be enabled:

- -lca
- -kdb // dumps the work.lib++ data for source coding view
- +define+SVT_PCIE_INCLUDE_AC_PA
- +define+SVT_FSDB_ENABLE // enables FSDB dumping
- -debug_access

  For more information on how to set the FSDB dumping libraries, see "Appendix B" section in *Linking Novas Files with Simulators and Enabling FSDB Dumping* guide available at *$VERDI_HOME/doc/linking_dumping.pdf*.

4.5.5 Run Time Options

You can set the format type for FSDB dump either through simulator command-line option or via a configuration setting.

4.5.5.1 Configuration Setting

Set the svt_pcie_configuration::pa_format_type variable to FSDB.

```
< svt_pcie_configuration>.pa_format_type=svt_xml_writer::FSDB
```

Note
The XML type is in the process of being deprecated.

4.5.5.2 Command Line Option

The following svt_enable_pa runtime switch can be provided to your simulator:

```
+svt_enable_pa=FSDB
```

Enables FSDB output of transaction and memory information for display in Verdi.

4.5.6 Invoking Protocol Analyzer

Perform the following steps to invoke Protocol Analyzer in interactive or post-processing mode.

4.5.6.1 Post-Processing Mode

Load the transaction dump data and issue the following command to invoke the GUI:

verdi -ssf <dump.fsdb> -lib work.lib++

In Verdi, navigate to Tools ->Transaction Debug -> Transaction and Protocol Analyzer to invoke Protocol Analyzer.

4.5.6.2 Interactive Mode

Issue the following command to invoke Protocol Analyzer in an interactive mode:

<simv> -gui=verdi

You can invoke the Protocol Analyzer as shown above through Verdi. The Protocol Analyzer transaction view gets updated during the simulation.

4.5.7 Limitations

Interactive support is available only for VCS.

4.6 Verification Planner

The PCIe VIP provides verification plans which can be used for tracking verification progress of the PCIe protocol. A set of top-level plans and sub-plans are provided. The verification plans are available at:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans

For more information, refer to the README file, which is available at:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/README

4.7 Global Shadow Memory

The purpose of this global shadow memory is to provide a database 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. Returned data can be compared with the shadow copy to verify that the DUT did the original write and the read correctly.

The model captures the following data in Shadow Memory:

- Memory Writes
- Atomic Operations.
- Memory Reads The host memory must have the IN_ORDER attribute set. When the completion for that read comes back, this saved snapshot (and not the current state of the memory) will be used in the shadow comparison. Note however, the PCIe generally does not guarantee In Order write/read behavior.
- Configuration Writes. With this data you can determine various behaviors of the DUT.

The model does *not* capture the following data in Shadow Memory:

- Error Injections - we assume that an EI will cause failure of the transaction to do what it intended.
- Poisoned (EP bit) Atomic Ops - these should fail (other poisoned transactions may or may not fail - this is implementation dependent.)
- Any TLP determined to be badly formed.
- Type 1 Cfg requests (these are destined for switches, not endpoints)

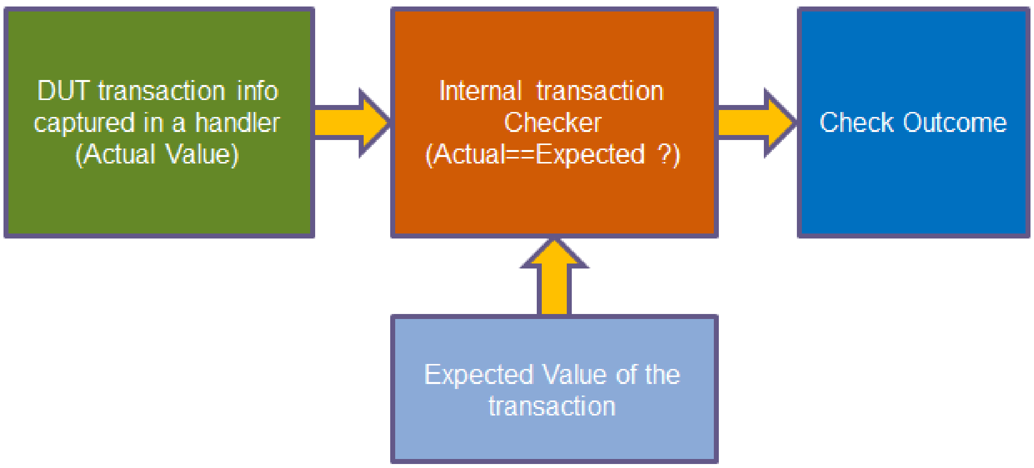
The VIP captures TLPs as they go on the wire and then are passed to the shadow's transaction handler which decodes all relevant transactions and updates the expected global shadow state. For example, outbound MemWrite TLPs are inspected and (if appropriate) written to the global shadow memory. This shadow 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. T

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).

The following illustration shows an usage example:



Global shadow needs to be explicitly declared and instantiated at the top when used:

```
`define EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH  test_top.global_shadow0
pciesvc_global_shadow #( .DISPLAY_NAME( "global_shadow0." ) ) global_shadow0();
```

4.7.1 Global Shadow Memory Classes

There are two classes for using Global Shadow Memory:

1. **svt_pcie_global_shadow** . This class is UVM Driver that implements a PCIE application namely Global Shadow. It provides a SIPP [Sequence Item Pull Port] to cater to services of type svt_pcie_global_shadow_service.
2. **svt_pcie_global_shadow_service.** This class represents service transactions for a PCIE Global Shadow Memory Application. The service_type attribute is the entry point to this object.

4.7.1.1 Class svt_pcie_global_shadow Members

The following table lists important members of the svt_pcie_global_shadow. It is derived as a uvm_component.

Table 4-6 Service Class Features for Global Shadow Memory

Member	Feature/Usage
ADD_MEM_RANGE(`SVT_PCIE_GLOBAL_SHADOW_SERVICE_ADD_MEM_RANGE)	Add supported memory range.

Table 4-6 Service Class Features for Global Shadow Memory (Continued)

Member	Feature/Usage
REMOVE_MEM_RANGE(`SVT_PCIE_GLOBAL_SHADOW_SERVICE_REMOVE_MEM_RANGE)	Remove supported memory range.
WRITE_MEM(`SVT_PCIE_GLOBAL_SHADOW_SERVICE_WRITE_MEM)	Memory write.
READ_MEM(`SVT_PCIE_GLOBAL_SHADOW_SERVICE_READ_MEM)	Memory read.
WRITE_CFG(`SVT_PCIE_GLOBAL_SHADOW_SERVICE_WRITE_CFG)	Configuration write.
READ_CFG(`SVT_PCIE_GLOBAL_SHADOW_SERVICE_READ_CFG)	Configuration read.
WRITE_CFG_CAP(`SVT_PCIE_GLOBAL_SHADOW_SERVICE_WRITE_CFG_CAP)	Cfg Cap write.
READ_CFG_CAP(`SVT_PCIE_GLOBAL_SHADOW_SERVICE_READ_CFG_CAP)	Cfg Cap read.
TRANSACTION_COMPLETE(`SVT_PCIE_GLOBAL_SHADOW_SERVICE_TRANSACTION_COMPLETE)	Transaction completed.
rand bit [63:0] address	Address to be read from or to be written to Global Shadow Memory.
rand bit [31:0] attributes	Attributes for ADD/REMOVE_MEM_RANGE service types.
rand bit [15:0] bdf	Bus-Device-Function for the device cfg reg in Global Shadow cfg.
rand bit [3:0] byte_enables	Byte Enables indicating enabled bytes during read/write of Global shadow memory.
svt_pcie_device_configuration cfg	Configuration pointer used to improve methods and constraints
rand bit [7:0] cfg_cap	The specific configuration-capability being read.
rand bit [31:0] cfg_reg	The specific configuration register being written in Global shadow cfg.
rand bit [31:0] data	Data to be written to Global Shadow Memory during WRITE service. When service_type is READ, it represents the data read.

Table 4-6 Service Class Features for Global Shadow Memory (Continued)

Member	Feature/Usage
rand bit [31:0] data_mask	Asserted bits indicate which bits of above data argument will be modified in the configuration-register (remaining bits will be unchanged.)
rand bit [31:0] dword_offset	Dword Offset indicating offset to the pcie_start_address. Valid offsets are 0 - data_length_in_dwords - 1.
bit error	Error indication if ADD/REMOVE_MEM_RANGE service types fail.
max_range	Upper address of the address range for ADD_MEM_RANGE and REMOVE_MEM_RANGE service types.
min_range	Lower address of the address range for ADD_MEM_RANGE and REMOVE_MEM_RANGE service types.
service_type	Memory target services
svt_sequence_item :: status_enum status	Status information about the current processing state
bit [31:0] task_status	Returns the status of READ/WRITE services.
rand bit [31:0] transaction_id	The combination of RequesterID and Tag for the transaction to cleanup.

4.7.2 Global Memory Examples

4.7.2.1 Random Memory Read using Global Shadow

User instantiates the requisite global shadow sequence in his parent sequence and reads data.

```
task user_parent_seq :: body()
…..
svt_pcie_global_shadow_service_random_rd_wr_sequence rand_rd_wr_seq;
…..

`uvm_config_db(mem_range_seq,p_sequencer.endpoint_virt_seqr.global_shadow,
{rand_rd_wr_seq == svt_pcie_global_shadow_service::    ;
 rand_rd_wr_seq. == 5; } );
```

4.7.2.2 Display Statistics of Global Shadow

Considering VIP as Endpoint. User instantiates the requisite global shadow sequence in his parent sequence and gets statistics.

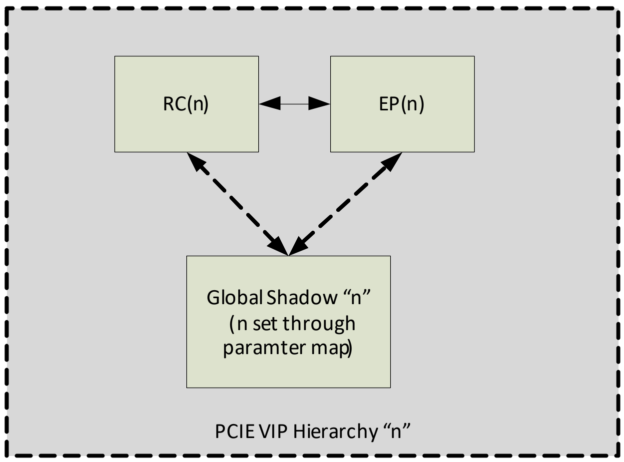
```
task user_parent_seq :: body()
          …..
svt_pcie_global_shadow_service_disp_stats_sequence     disp_stat_seq;
```

```
`uvm_config_db(mem_range_seq,p_sequencer.endpoint_virt_seqr.global_shadow,
{disp_stat_seq.service_type == svt_pcie_global_shadow_service::DISPLAY_STATS;} );
```

4.7.3 Multiple Global Shadows

Figure 4-1 shows a virtual hierarchy created by the physical connection of RC and EP and the virtual connection to the associated shadow. This feature is implemented in the model via the parameter HIERARCHY_NUMBER in the shadow and all of the instantiation models.

Figure 4-1 PCIe VIP Hierarchy Definition



4.7.3.1 Setting up Multiple Shadows

4.7.3.1.1 Module Level Construction

Instantiate a desired number of shadows (one for each PCIe VIP hierarchy described earlier). Relative location of the `pciesvc_global_shadow` instances to the device VIP instances in the module hierarchy is not important. The value of the associated `HIERARCHY_NUMBER` setting between the root, global shadow, and downstream endpoints is crucial.

Note
For setting up multiple shadows in a testbench, do not use the define `EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH`. Global shadow is a memory shared by one RC and all EP instances associated with it. The association is done using `HIERARCHY_NUMBER` parameters while creating the module instances.RC and EP VIP instances use the `SVT_PCIE_UI_HIERARCHY_NUMBER` parameter, whereas `pciesvc_global_shadow` uses the `HIERARCHY_NUMBER` parameter.

```
// Global Shadow Instances ( optional )
pciesvc_global_shadow#(.DISPLAY_NAME({DISPLAY_NAME, "global_shadow0." }),
.HIERARCHY_NUMBER(0)) global_shadow0(); 
pciesvc_global_shadow#(.DISPLAY_NAME({DISPLAY_NAME, "global_shadow1." }),
.HIERARCHY_NUMBER(1)) global_shadow1(); …

// PCIE VIP Hierarchy 0
<pcie RC module>#(.DISPLAY_NAME(…), .HIERARCHY_NUMBER(0),…) <inst name> …
<pcie EP module>#(.DISPLAY_NAME(…), .HIERARCHY_NUMBER(0),…) <inst name> …
 // PCIE VIP Hierarchy 1
<pcie RC module>#(.DISPLAY_NAME(…), .HIERARCHY_NUMBER(1),…) <inst name> …
<pcie EP module>#(.DISPLAY_NAME(…), .HIERARCHY_NUMBER(1),…) <inst name> …
```

4.7.3.1.2 SVT Global Shadow Component Instantiation

You must modify construction of the shadow agent within environments by replacing svt_pcie_global_shadow::type_id::create() with the specialized method svt_pcie_global_shadow::create_shadow() (see Table 4-7). This modification is required when you want to use multiple shadows. The hierarchy_number argument of create_shadow() must match the HIERARCHY_NUMBER parameter value of the corresponding pciesvc_global_shadow module instance.

Example instantiation of two svt_pcie_global_shadows linked to the corresponding global shadow modules 0 an 1 from the previous module instantiation:

```
global_shadow[0] = svt_pcie_global_shadow::create_shadow(0, "global_shadow[0]”, this);
global_shadow[1] = svt_pcie_global_shadow::create_shadow(1, "global_shadow[1]”, this);
```

Table 4-7 Multiple Shadow Support Functions

New Method	Arguments	Description
create_shadow	int hierarchy_number, // Global Shadow Hierarchy Number string name=”svt_pcie_global_shadow”, // Name of returned component uvm_component parent = null // Parent component	This is a function that will look up the specific shadow with the predetermined string format, then create and return the component. Similar to ::create() but will not concatenate parent and name as the lookup string. Accepts same arguments as create for “name” and “parent”.
get_hierarchy_number		Returns the hierarchy number of the global shadow agent

4.7.4 Disabling Global Shadows

Global shadow is a memory component used in PCIe SVT VIP to compare write data (MWr or IOWr payload) with read data (CplD payload in response of MRd or IORd).

Global shadow component is instantiated in the testbench as shown in the following code snippet:

```
pciesvc_global_shadow #( .DISPLAY_NAME( "global_shadow0." ) ) global_shadow0();
`define EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH  test_top.global_shadow0
```

If you want to disable global shadow at run time, you must invoke the following function:

```
<device_cfg>.enable_all_global_shadow_vars(0);
```

Alternatively, you can set following configuration attributes per component to enable/disable shadow memory checking:

```
<device_cfg>.driver_cfg[0].enable_tx_tlp_reporting = 0;
<device_cfg>.driver_cfg[0].enable_shadow_memory_checking = 0;
<device_cfg>.requester_cfg.enable_tx_tlp_reporting = 0;
<device_cfg>.requester_cfg.enable_shadow_memory_checking = 0;
<device_cfg>.pcie_cfg.dl_cfg.enable_tx_tlp_reporting = 0;
<device_cfg>.pcie_cfg.tl_cfg.enable_shadow_cfg_lookup = 0;
```

4.8 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.

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

This memory is also used as the storage mechanism for the Global Shadow Memory (see "Global Shadow Memory" on page 58 for details).

The service class *svt_pcie_mem_target_service* class represents service transactions for a PCIe Memory Target Application. The service_type attribute is the entry point to this object. The following table shows the various memory target service types.

Table 4-8 Memory Target Service Types

Service	Description
WRITE(`SVT_PCIE_MEM_TARGET_SERVICE_WRITE)	Memory write of single DWORD to Memory space of Target application. NOTE: This service call will be obsoleted in a future release.
READ(`SVT_PCIE_MEM_TARGET_SERVICE_READ)	Memory read of single DWORD to Memory space of Target application. NOTE: This service call will be obsoleted in a future release.
PRE_WRITE_HINT(`SVT_PCIE_MEM_TARGET_SERVICE_PRE_WRITE_HINT)	Add Pre-write hint to the memory. NOTE: This service call will be obsoleted in a future release.
ADD_MEM_RANGE(`SVT_PCIE_MEM_TARGET_SERVICE_ADD_MEM_RANGE)	Add supported memory range.
REMOVE_MEM_RANGE(`SVT_PCIE_MEM_TARGET_SERVICE_REMOVE_MEM_RANGE)	Remove supported memory range.
DISPLAY_STATS(`SVT_PCIE_MEM_TARGET_SERVICE_DISPLAY_STATS)	Display statistics variables.
CLEAR_STATS(`SVT_PCIE_MEM_TARGET_SERVICE_CLEAR_STATS)	Clear statistics variables.
RESET_APP(`SVT_PCIE_MEM_TARGET_SERVICE_RESET_APP)	Resets app back to its inital state. All will be lost.
WRITE_BUFFER(`SVT_PCIE_MEM_TARGET_SERVICE_WRITE_BUFFER)	Memory write of multiple DWORDs to Memory space of Target application.
READ_BUFFER(`SVT_PCIE_MEM_TARGET_SERVICE_READ_BUFFER)	Memory read of multiple DWORDs to Memory space of Target application.

A service call is used to mark a memory region as ignored. As such, the memory region will not have memory allocated for it. Read requests will be returned with random data. Write requests will have no affect.

```
/**< Add supported memory range.*/
ADD_MEM_RANGE = `SVT_PCIE_MEM_TARGET_SERVICE_ADD_MEM_RANGE,

/**< Remove supported memory range.*/
REMOVE_MEM_RANGE = `SVT_PCIE_MEM_TARGET_SERVICE_REMOVE_MEM_RANGE,

..........
//---------- Variables for ADD_MEM_RANGE and REMOVE_MEM_RANGE services ----------

  /**
   *  Lower address of the address range for ADD_MEM_RANGE and
   *  REMOVE_MEM_RANGE service types.
   */
  rand bit [63:0]      min_range = 'h0;

  /**
   *  Upper address of the address range for ADD_MEM_RANGE and
   *  REMOVE_MEM_RANGE service types.
   */
  rand bit [63:0]      max_range = 64'hFFFF_FFFF_FFFF_FFFC;

  /**
    * Attributes for ADD/REMOVE_MEM_RANGE service types.
    * - attributes[0]: Ignore. Assert this bit in the attributes to
    * disallow checking against this address range.
    * - attributes[1]: In Order. Assumes transaction will be handled in the DUT the
    * order that they were received, this provides the potential for checking some
    * alternating read/write transactions.
    */

  rand bit [31:0]      attributes = 32'h0;
```

4.8.1 Ignoring Memory Ranges

The service call `svt_pcie_mem_target_service::ADD_MEM_RANGE` (along with `REMOVE_MEM_RANGE`) are used to configure the memory range and ignored attribute. The sequence class `svt_pcie_mem_target_service_mem_range_sequence` provides access to this mechanism.

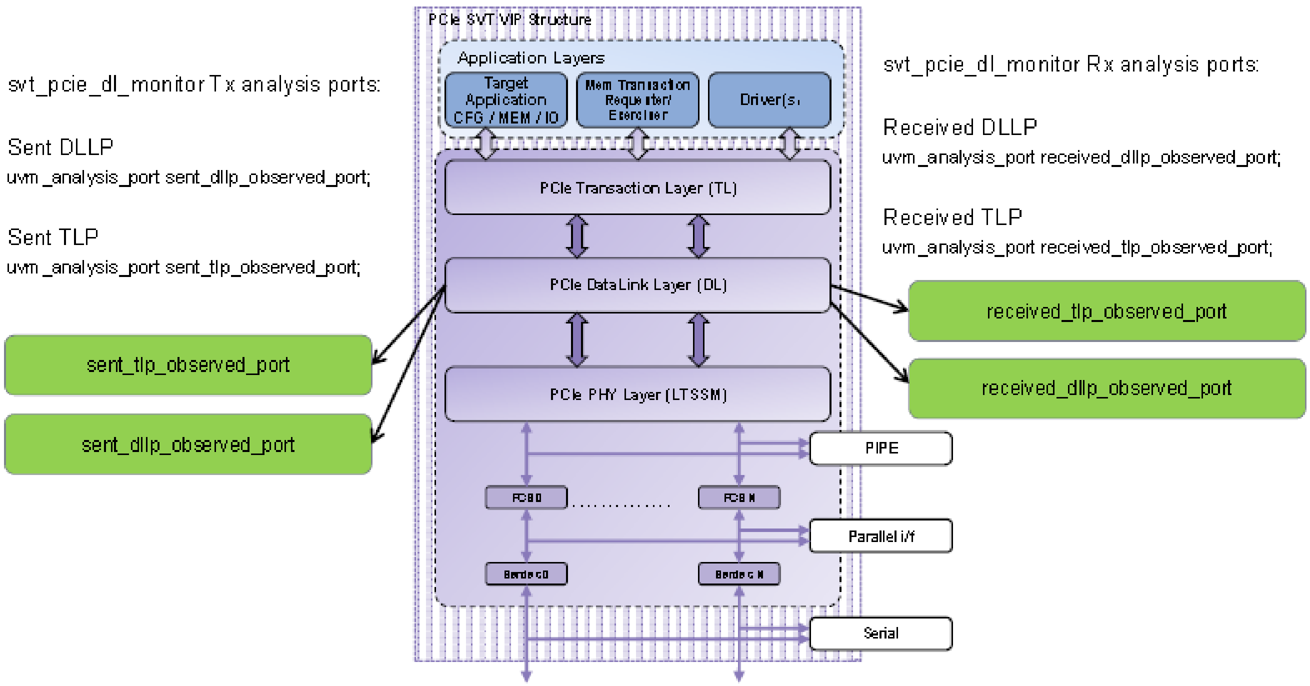
The configuration in the class `svt_pcie_target_app_configuration` for this is `uninit_mem_read_resp`, which can be set with various `UNINIT_MEM_READ_RESP_*` values.

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. In this way, a checker can determine if the transaction is expected to return a predicable result or not. Occasionally, there will be memory regions (for example, registers) that you do not want to check for correctness—write-only registers, status or statistics registers that may change sporadically, and so on. These regions in the shadow memory can be marked as `IGNORED` via the `AddMemRange()` task, which is identical as above, but is accessed via the `` `EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH `` define.

4.9 Data Link Monitor

The VIP has a Data Link Monitor which is used to indicate when TLPs and DLLP are sent and received. Figure 4-2 shows the various analysis ports for monitoring TLPs an DLLPs.

Figure 4-2 Data Link Monitor and Monitor Ports and Classes



The PCIe UVM VIP has the following TLM analysis ports in the DL to access sent/received TL packets.

- svt_pcie_dl::received_tlp_observed_port: Analysis port for to sample TLPs being received by the VIP. This port is generally used for scoreboarding.
- svt_pcie_dl::sent_tlp_observed_port: Analysis port for to sample TLPs being sent by the VIP. This port is generally used for scoreboarding.

The TLPs observed via these ports are controlled by a configuration variables in the DL namely:

- svt_pcie_dl_configuration::received_tlp_interface_mode
- svt_pcie_dl_configuration::sent_tlp_interface_mode.

For example:

```
endpoint_cfg.pcie_cfg.dl_cfg.received_tlp_interface_mode = 1;
endpoint_cfg.pcie_cfg.dl_cfg.sent_tlp_interface_mode = 1;
```

These configuration parameters are 2-bit variables. Bit 0 corresponds to the enabling of "good" packets and bit 1 corresponds to the enabling of "error" packets. Check the HTML class description for more details:

- $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_dl_configuration.html#item_received_tlp_interface_mode
- $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_dl_configuration.html#item_sent_tlp_interface_mode

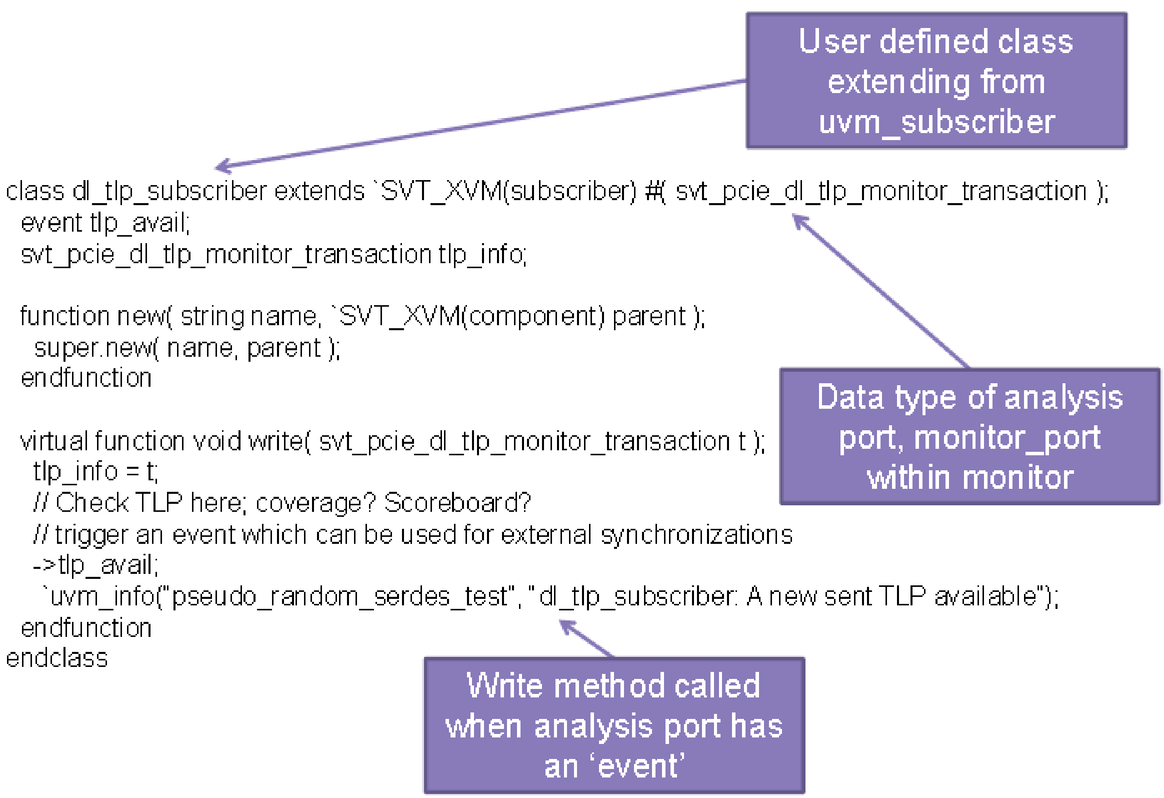
Once enabled these ports can be used to subscribe to transaction being sent/received by the VIP model with the use of UVM subscribers. The code example below illustrates the same.

Use the following flow to setup the DL Monitor.

1. Identify the analysis port on the DL monitor
2. The sent_tlp_observed_port will tell you when the TLP was sent, along with providing the TLP transaction class
3. Note the class name: svt_pcie_dl_monitor
4. Create a uvm_subscriber extension

   Goal: uvm_subscriber::write() will be called when the TLP is sent
5. Add a subscriber to your uvm_test
   a. Build phase : new your uvm_subscriber class
   b. Connect phase : connect to the uvm_analysis, monitor port in the class identified above
6. Done, when the TLP is sent, the write() method will be called.

The example which follows is annotated to explain the use of the DL Monitor following the previous steps.



```
class pseudo_random_serdes_test extends pcie_device_base_test ;

  dl_tlp_subscriber  sent_tlp_subscriber;

  …

  virtual function void build_phase(uvm_phase phase);
    `uvm_info("build_phase", "Entered...", UVM_LOW)
    super.build_phase(phase);
    sent_tlp_subscriber = new( “sent_tlp_subscriber", this );
    `uvm_info("build_phase", "Exiting...", UVM_LOW)
  endfunction: build_phase

….
```

Add subscriber to your uvm_test implementation

Within the build_phase, create an instance of your subscriber

```
 …

virtual function void connect_phase( uvm_phase phase );
  super.connect_phase( phase );
  env.endpoint.port.dl_monitor.sent_tlp_observed_port.connect( sent_tlp_subscriber.analysis_export );
endfunction
```

“connect” to the monitor within the connect_phase

Use analysis_export to make the connection

```
UVM_INFO ./ts.pseudo_random_test.sv(76) @ 81793300.10 ps:
uvm_test_top.tlp_subscriber dl_tlp_subscriber: A new sent TLP available
```

from log….

5 General VIP Protocol Features

This chapter describes the general VIP protocol features available with the Synopsys PCIe Verification IP. This chapter discusses the following topics:

5.1 PCIe Gen3 Support

To enable Gen3 features, the SVT_PCIE_ENABLE_GEN3 macro must be defined on the command line for VCS invocation and the svt_pcie_device_configuration::pcie_spec_ver must be set to svt_pcie_device_configuration::PCIE_SPEC_VER_3_0.

The following is an example of how to define a macro on a command line for VCS invocation:

```
vcs +define+SVT_PCIE_ENABLE_GEN3 other_switches
```

5.2 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.

5.3 Power Management

The Data Link Layer in the SVT PCIE VIP provides support for PM/ASPM functionality similar to the specification. As defined by the specification, the VIP can be directed into and out of particular power states. The VIP can also be configured to automatically enter states as specified by the protocol. Exit from low power states can be initiated by the VIP as well if it needs to transmit a TLP or DLLP.

NOTE: ASMP L1 entry must be enabled by the Data Link configuration, but PM L1 does not (due to specification controls).

The VIP has built in checkers to make sure the handshake process occurs per specification. Timeouts are used to make sure handshakes occur within a reasonable time frame.

5.3.1 ASPM

5.3.2 L0s Entry

For L0s entry, the VIP can be configured to automatically transition to Tx L0s when an idle period is reached. Or it can be directed to Tx L0s. The idle timer, when set to a non-zero value, enables the VIP's automatic entry to L0s. The DL can also be sent immediately to Tx L0s via a service request.

For exit from L0s to L0, the VIP can be directed or autonomously transition. User directed exit from Tx L0s is initiated via DL INITIATE_ASPM_EXIT service request. Autonomous exit occurs when any DLLP or TLP needs to be transmitted.

:

Table 5-1 DL L0s Configuration

Member	Description
l0s_idle_timer_limit_ns	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.

Table 5-2 DL Service Requests

Member	Description
INITIATE_ASPM_L0S_ENTRY	Initiates VIP to enter ASPM Tx L0s low power state.
NITIATE_ASPM_EXIT	Initiates VIP to transition back to L0 from ASPM low power state.

5.3.2.1 L1 Entry

ASPM L1 entry must be enabled by DL configuration variables listed below. The entry/exit to/from ASPM L1 is initiated by DL service requests INITIATE_ASPM_L1_ENTRY/ INITIATE_ASPM_EXIT.

For exit from L1 to L0, the VIP can be directed or autonomously transition. User directed exit from L1 is initiated via a DL INITIATE_ASPM_EXIT service request. Autonomous exit occurs when any DLLP or TLP needs to be transmitted.

:

Table 5-3 DL Service Requests

enable_aspm_l1_entry	Enable ASPM L1 entry
INITIATE_ASPM_L1_ENTRY	Initiates VIP to enter PM L1 low power state
INITIATE_ASPM_EXIT	Initiates VIP to transition back to L0 from ASPM low power state.

5.3.2.2 L1 Substate Entry

The VIP supports entry into L1 substates. Entry must be enabled via DL configuration variables listed below. These vars must be set in addition to the vars listed in the L1 section. If L1_1 and L1_2 are both enabled, the VIP will transition to the highest power savings state, which is L1_2. Entry/exit to/from L1 substates is initiated just like L1 entry/exit.

Table 5-4 DL Configuration Members for L1 Substrate Entry

Member	Description
enable_aspm_l1_2_entry	The variable enables ASPM L1.2 entry. Must be used in conjunction with enable_aspm_l1_entry and INITIATE_ASPM_L1_ENTRY service request
enable_aspm_l1_1_entry	The variable enables ASPM L1.1 entry. Must be used in conjunction with enable_aspm_l1_entry and INITIATE_ASPM_L1_ENTRY service request.

5.3.2.3 Active State NAK

Active state NAK TLP msgs must be initiated by the test case. Active State NAK TLPs received from the DUT can be forwarded to the test case via

5.3.3 PM

The VIP assumes the test case has performed the proper PM handshake in order for the VIP to transition to low power states. The VIP does not respond to PM TLP messages nor will it initiate any PM TLP messages. In order for the test case to complete the handshake with the VIP, the received PM TLP messages must be forwarded to the test case by the VIP. This is accomplished by routing PM TLPs to the testcase via the TLs mapping tables using ADD_RID_MSG_CODE_APPL_ID_MAP_ENTRY service request.

Once the test case has determined the functions in the DUT are ready for transition to low power states, the test case can initiate VIP transition per sections below. If the DUT is initiating the transition, the VIP will respond as if its functions are ready for PM low power transition.

5.3.3.1 L1

PM L1 does NOT have to be enabled by DL configuration. This behavior is enabled by default, similar to the specification. The entry/exit to/from PM L1 is initiated by DL service requests INITIATE_PM_L1_ENTRY/ INITIATE_PM_EXIT.

For exit from L1 to L0, the VIP can be directed or autonomously transition. User directed exit from L1 is initiated via the DL INITIATE_PM_EXIT service request. Autonomous exit occurs when any DLLP or TLP needs to be transmitted.

No configuration members exist for DL D1 configuration.

Table 5-5 L1 DL Service Requests

Member	Description
INITIATE_PM_L1_ENTRY	Initiates VIP to enter PM L1 low power state
INITIATE_PM_EXIT	Initiates VIP to transition back to L0 from PM low power state.

5.3.3.2 L1 Substate Entry

The VIP supports entry into L1 substates. Entry to L1 substates must be enabled via DL configuration variables listed below. If L1_1 and L1_2 are both enabled, then the VIP will transition to the highest power savings state, which is L1_2. Entry/exit to/from L1 substates is initiated just like L1 entry/exit

Table 5-6 L1 Substrate Entry Members

Member	Description
enable_pm_l1_2_entry	The variable enables PM L1.2 entry. Must be used in conjunction with enable_pm_l1_entry and INITIATE_PM_L1_ENTRY service request
enable_pm_l1_1_entry	The variable enables PM L1.1 entry. Must be used in conjunction with enable_pm_l1_entry and INITIATE_PM_L1_ENTRY service request.

5.3.3.3 L2/3 Entry

PM L2/3 entry does NOT need to be enabled by DL configuration. This behavior is enabled by default, similar to the specification The entry/exit to/from PM L2/3 is initiated by DL service requests INITIATE_PM_L23_ENTRY/INITIATE_PM_EXIT.

For exit from L2/3 to L0, the VIP can be directed or autonomously transition. User directed exit from L2/3 is initiated via DL INITIATE_PM_EXIT service request. Autonomous exit occurs when any DLLP or TLP needs to be transmitted

There is no configuration member for D1 L1.

Table 5-7 DL Service Requests

Member	Description
INITIATE_PM_L23_ENTRY	Initiates VIP to enter PM L2/L3 low power state.
INITIATE_PM_EXIT	Initiates VIP to transition back to L0 from PM low power state.

5.3.4 VIP PM/ASPM Checks

The VIP has automatic checking for PM/ASPM functionality. The checks can be demoted if the behavior is expected. The timeouts are configurable via DL configuration. By default, any PM request is expected to complete successfully. In the event that an ASPM active state NAK TLP is received, the VIP will flag this as a UVM_WARNING.

Table 5-8 Pm/ASPM Checks

Member	Description
MSGCODE_PCIESVC_DL_ASPM_L1_HANDSHAKE_TIMEOUT	• f DUT fails to respond to ASPM request handshake for this: • # symbols, NOTICE will be issued • and ASPM entry will be aborted. aspm_timeout_cnt_limit = `SVT_PCIE_ASPM_TIMEOUT_CNT_LIMIT_DEFAULT;
MSGCODE_PCIESVC_DL_ASPM_L1_1_HANDSHAKE_TIMEOUT	
MSGCODE_PCIESVC_DL_ASPM_L1_2_HANDSHAKE_TIMEOUT	
MSGCODE_PCIESVC_DL_PM_L1_HANDSHAKE_TIMEOUT	• If DUT fails to respond to PM request handshake for this: • # symbols, NOTICE will be issued and • * PM entry will be aborted. */ rand int unsigned pm_timeout_cnt_limit = `SVT_PCIE_PM_TIMEOUT_CNT_LIMIT_DEFAULT;
MSGCODE_PCIESVC_DL_PM_L1_1_HANDSHAKE_TIMEOUT	
MSGCODE_PCIESVC_DL_PM_L1_2_HANDSHAKE_TIMEOUT	
MSGCODE_PCIESVC_DL_PM_L23_HANDSHAKE_TIMEOUT	
MSGCODE_PCIESVC_DL_ASPM_L1_RX_ACTIVE_STATE_NAK	ASPM L1 Handshake terminated by receiving ACTIVE_STATE_NAK
MSGCODE_PCIESVC_DL_RECEIVED_UNEXPECTED_PM_ACK	Received PM_REQUEST_ACK DLLP when PM was not requested
MSGCODE_PCIESVC_DL_RECEIVED_TLP_ASPM_L1_STARTED	Received TLP when ASPM L1 entry is in progress. Dut should hold TLP until L1 hanshake is complete. Spec 5.4.1.2.1
MSGCODE_PCIESVC_DL_RECEIVED_TLP_PM_L1_STARTED	Received TLP when PM L1 entry is in progress. Dut should hold TLP until L1 hanshake is complete. Spec 5.3.2.1
MSGCODE_PCIESVC_DL_RECEIVED_TLP_PM_L23_STARTED	Received TLP when PM L2/L3 entry is in progress. Dut should hold TLP until L2/L3 hanshake is complete. Spec 5.3.2.3

5.4 Setting Coefficient and Preset for Gen3 Equalization

Transmitter equalization is adopted in Gen 3 to compensate for an increased signal distortion from operating at a higher data rate. On entry to the 8.0 GT/s data rate, the link partners exchange equalization presets and coefficients to determine the transmitter and receiver settings that yield an optimal signal-to-noise ratio on each lane. This trainable equalization process consists of 4 phases. The phase information is specified in the Equalization Control (EC) field in the TS1 Ordered Sets.

In phase 0, in Recovery.RcvrCfg before transitioning to 8.0 GT/s, the Downstream port transmits the recommended preset and coefficient values to the Upstream Port. The Upstream Port uses these recommended values in phase 0 and phase1.

In phase 1, the Downstream and Upstream Ports transmit with their respective coefficients. Both ports advertise the preset and post cursor values of their transmitters, the LF and FS values. Equalization is complete in phase 1 if a finer adjustment to the preset and coefficient values is not required. If the Downstream Port requests a finer adjustment to the presets and coefficients, then the ports proceed to phase 2.

In phase 2, the transmitter coefficients of the Downstream Port are optimized. The Upstream Port can request the Downstream Port to adjust its transmitter by setting the preset and coefficient fields in the transmitted TS OS. The Downstream Port either accepts or rejects these new values. Once an optimal preset and coefficient values are determined by the Upstream Port, the Upstream Port transitions to phase 3.

In phase 3, the transmitter coefficients of the Upstream Port are optimized. The Downstream Port can request the Upstream Port to adjust its transmitter by setting preset and coefficient fields in the transmitted TS OS. The Upstream Port either accepts or rejects these new values. Once an optimal preset and coefficient values are determined by the Downstream Port, the Downstream Port transitions to Recovery.RcvrLock.

5.4.1 Enabling Equalization

To configure the highest supported speed and link equalization behavior collectively in the VIP, use following functions:

- `svt_pcie_pl_configuration::set_link_speed_values(bit [31:0] supported_link_speeds_value, bit [31:0] target_link_speed_value = 32'h0, bit [31:0] expected_link_speed_value = 32'h0);`
- `svt_pcie_pl_configuration::set_link_eq_attribute_values(link_eq_mode_enum link_eq_mode_value = LINK_EQ_MODE_FULL_EQUALIZATION_REQUIRED, bit enable_direct_speed_up_from_2_5g_to_16g_value = 0, int unsigned highest_enabled_eq_phase_value = 3);`
  - Gen1 → Gen3 (With Equalization)

    ```
    cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_8_0G |
    `SVT_PCIE_SPEED_5_0G | `SVT_PCIE_SPEED_2_5G);
    cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values();
    ```
  - Gen1 → Gen3 (Without Equalization)

    ```
    cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_8_0G |
    `SVT_PCIE_SPEED_5_0G | `SVT_PCIE_SPEED_2_5G);
    cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(,,0
    /*highest_enabled_eq_phase*/);
    ```
  - Gen1 → Gen3 → Gen4 (With Equalization)

    ```
    cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_16_0G |
    `SVT_PCIE_SPEED_8_0G | `SVT_PCIE_SPEED_5_0G | `SVT_PCIE_SPEED_2_5G);
    cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values();
    ```
  - Gen1 → Gen3 → Gen4 (Without Equalization)

    ```
    cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_16_0G |
    `SVT_PCIE_SPEED_8_0G | `SVT_PCIE_SPEED_5_0G | `SVT_PCIE_SPEED_2_5G);
    cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(,,0
    /*highest_enabled_eq_phase*/)
    ```
  - Gen1 → Gen4 (With Equalization)

    ```
    cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_16_0G |
    `SVT_PCIE_SPEED_8_0G | `SVT_PCIE_SPEED_5_0G | `SVT_PCIE_SPEED_2_5G);
    ```

```
cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(,1
/*enable_direct_speed_up_from_2_5g_to_16g*/);
```

- Gen1 → Gen4 (Without Equalization)

```
cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_16_0G |
`SVT_PCIE_SPEED_8_0G | `SVT_PCIE_SPEED_5_0G | `SVT_PCIE_SPEED_2_5G);
cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(,1
/*enable_direct_speed_up_from_2_5g_to_16g*/, 0 /*highest_enabled_eq_phase*/)
```

- Gen1 → Gen3 → Gen4 → Gen5 (With Equalization)

```
cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_32_0G |
`SVT_PCIE_SPEED_16_0G | `SVT_PCIE_SPEED_8_0G | `SVT_PCIE_SPEED_5_0G |
`SVT_PCIE_SPEED_2_5G);
cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(LINK_EQ_MODE_FULL_EQUALIZ
ATION_REQUIRED);
```

- Gen1 → Gen5 (With Equalization)

```
cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_32_0G |
`SVT_PCIE_SPEED_16_0G | `SVT_PCIE_SPEED_8_0G | `SVT_PCIE_SPEED_5_0G |
`SVT_PCIE_SPEED_2_5G);

cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(LINK_EQ_MODE_EQ_BYPASS_TO
_HIGHEST_RATE);
```

- Gen1 → Gen5 (Without Equalization)

```
cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_32_0G |
`SVT_PCIE_SPEED_16_0G | `SVT_PCIE_SPEED_8_0G | `SVT_PCIE_SPEED_5_0G |
`SVT_PCIE_SPEED_2_5G);
cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(LINK_EQ_MODE_NO_EQUALIZAT
ION_NEEDED);
```

Equalization checking is disabled by default In the PCIe VIP. You can use the following attributes in the PHY layer configuration class (`svt_pcie_pl_configuration`) to enable equalization checking.

Table 5-9 Equalization Modes

Attribute	Description
enable_equalization_verification_mode	Enables equalization verification mode
enable_equalization_coefficients_checks	Enables equalization coefficient check
highest_enabled_equalization_phase	Specifies the highest equalization phase to be enabled. A value of 1 enables equalization phase 0 and phase 1. A value of 3 enables equalization phases 0, 1, 2, and 3.

For example:

```
root_cfg.pcie_cfg.pl_cfg.enable_equalization_verification_mode = 1;
root_cfg.pcie_cfg.pl_cfg.enable_equalization_coefficients_checks = 1;
root_cfg.pcie_cfg.pl_cfg.highest_enabled_equalization_phase = 3;

endpoint_cfg.pcie_cfg.pl_cfg.enable_equalization_verification_mode = 1;
endpoint_cfg.pcie_cfg.pl_cfg.enable_equalization_coefficients_checks = 1;
endpoint_cfg.pcie_cfg.pl_cfg.highest_enabled_equalization_phase = 3;
```

5.4.2 Specifying Coefficients, Presets, LF and FS Values

5.4.2.1 Initializing Presets with EQ TS OS

As a Downstream Port, the VIP transmits EQ TS OS with recommended preset values. The preset values are specified by the "upstream_preset_value" attribute in the PHY layer configuration class (svt_pcie_pl_configuration). If the recommended preset values are mapped to the valid coefficients in the DUT (Upstream Port), then the DUT transmits TS OS with the recommended preset values in equalization phases 0, 1, and 3. The TS OS coefficient fields of the DUT are specified with the corresponding coefficients from the preset mapping table of the DUT. The VIP compares the preset field of the received TS OS with the "upstream_preset_value" attribute. For more information on the mapping table, refer to the "Preset to Coefficient Mapping" section.

As an Upstream port, the VIP receives EQ TS OS with preset values recommended by the DUT. The VIP transmits TS1 OS with the recommended preset values in equalization phases 0, 1, and 3. The coefficient fields of the VIP are specified with the corresponding coefficients from the preset mapping table of the VIP in phases 0 and 3. The post cursor value is specified in phase 1.

You can use the following attributes in the PHY layer configuration class (svt_pcie_pl_configuration) to specify the coefficients, LF and FS values.

A

Table 5-10 Attributes of PHY Layer Configuration Class

Attribute	Description
upstream_preset_value	Specifies the Upstream Port preset value.
preset_to_coefficients_mapping_table	Maps received preset requests to coefficients for use in local transmitter settings. This table is indexed by a preset value. The coefficients are packed in the following format: { postcursor_coeff, cursor_coeff, precursor_coeff } The default value is { 6'h01, 6'h56, 6'h01} or 18'h01b81.
lf_value	Specifies the LF value advertised by the VIP in TS1s during equalization phase 1.
fs_value	Specifies the FS value advertised by the VIP in TS1s during equalization phase 1.

For example:

```
//Specify mapping table values to change default coefficients
root_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_table[0]='{16{18'h00543}};

root_cfg.pcie_cfg.pl_cfg.lf_value = '{32{'d9}};
root_cfg.pcie_cfg.pl_cfg.fs_value = '{32{'d24}};
```

5.4.2.2 Preset to Coefficient Mapping

The VIP includes two preset mapping tables in the PHY layer configuration class (svt_pcie_pl_configuration).

Table 5-11 Preset PHY Mapping

Attribute	Description
preset_to_coefficients_mapping_table	Maps received preset requests to coefficients for use in local transmitter settings. This table is indexed by a preset value. The coefficients are packed in the following format: { postcursor_coeff, cursor_coeff, precursor_coeff } The default value is { 6'h01, 6'h56, 6'h01} or 18'h01b81.
expected_preset_to_coefficients_mapping_table	Verifies the preset to coefficient mappings in the DUT. This table should be programmed with the same value as the preset table in the DUT. This table is indexed by a preset value. The coefficients are packed in the following format: { postcursor_coeff, cursor_coeff, precursor_coeff } The default value is { 6'h0c, 6'h24, 6'h00} or 18'h0c900.

As a Downstream Port, the VIP verifies the DUT preset mapping in Phase 0 and again in Phase 3. As an Upstream Port, the VIP verifies the DUT preset mapping in Phase 2. If the preset maps to an invalid entry, the VIP disables mapping check, transmits with the recommended coefficients specified by the DUT, and set the reject preset bit.

5.4.2.3 LF and FS Values

The LF and FS values are advertised in phase 1. The VIP uses the values advertised by the DUT to determine the DUT's acceptance or rejection of its recommended presets and coefficients. If the DUT does not accept or reject the presets and coefficients, the VIP issues a warning message.

For the LF and FS values advertised by the VIP, the VIP verifies that the presets and coefficients recommended by the DUT does not violate its LF and FS values.

5.4.2.4 Rejecting Presets or Coefficients

The VIP has built-in checks for preset or coefficient requests from the DUT as defined in the PCIe specification. In addition, the VIP includes a mode to manually force a rejection on a per lane basis regardless of the results of the built-in check.

The VIP issues a notice message when the DUT rejects a coefficient.

5.4.2.5 Automatic Rejection

For each preset or coefficient request, the VIP verifies that the mapped coefficients in the preset case or received coefficients does not violate the LF and FS rules as defined in section 4.2.3.1 of the PCIE Specification. If a violation occurs, the VIP asserts the "reject coefficient values" bit in the transmitted TS OS on that particular lane. This bit is also asserted for preset requests that do not map to valid coefficients. When a new set of presets and coefficients are received, the VIP performs a new check.

5.4.2.6 Manual Rejection

Manual rejection can be specified for any preset or coefficient request from the DUT on any lane. The VIP asserts the reject bit in the TS OS in the appropriate phase.

You can use the following attribute in the PHY layer configuration class (svt_pcie_pl_configuration) to reject preset or coefficient values requested by the DUT.

Attribute	Description
reject_preset_coefficient_request	Each bit maps to a corresponding lane. When a bit is set to 1'b1, the corresponding lane rejects the new preset and coefficient values. This bit is applicable only in equalization phase 2 for Downstream Ports and equalization phase 3 for Upstream Ports.

For example:

```
root_cfg.pcie_cfg.pl_cfg.reject_preset_coefficient_request = 32'h0
```

5.4.3 Preset and Coefficient Tuning Through Windowed Filtering

During `Recovery.Equalization` phases 2 and 3, the preset and coefficient of PHY transmitter are tuned for optimal signal integrity at receiver. PCIe specification describes the equalization process but does not specify the actual algorithm involving the preset and coefficient tuning. This section describes the supported optional algorithm to tune preset and coefficient through a library sequence.

Note
Currently, only the sequence operating on Endpoint VIP with master PIPE at Gen 3 is part of sequence library.

This section describes the preset and coefficient tuning algorithm from the perspective of VIP acting as upstream device with master PIPE interface, during `Recovery.Equalization` phase 2.

Figure of Merit (FOM) feedback mechanism is used for the process of fine tuning link partners PHY TX presets and Direction Change (DIR) feedback is used for fine tuning link partner's PHY TX coefficients. Both preset and coefficient tuning are options, MAC may choose to not to do any tuning or either one or both. If both preset and coefficient are to be tuned, preset tuning is done first followed by coefficient tuning.

Figure of Merit (FOM)

For preset selection through FOM feedback, MAC will send a set of user-programed presets to link partners PHY one at a time and will instruct its own PHY to evaluate incoming RX signal integrity. After PHY is done with evaluation of incoming signal, it will respond with a relative score for the preset sent, ranging from 0 to 255 through `LinkEvaluationFeedbackFigureMerit[7:0]` PIPE signal. Higher the FOM feedback value, better is the incoming RX signal integrity. Once MAC has evaluated all the programed presets, it will select the preset with the highest FOM feedback value.

Direction Change (DIR)

For coefficient tuning through DIR feedback, MAC sends coefficients (precursor, cursor and postcursor) for link partner PHY transmitter, then request its own PHY to evaluate the incoming RX signal integrity. After PHY is done with evaluating the incoming signal integrity, it will provide the feedback on `LinkEvaluationFeedbackDirectionChange[5:0]` PIPE signal and 2-bit feedback per coefficient. A value of 2'b00 indicates no further change is required for coefficient, a value of 2'b01 means increment the coefficient value by 1 and a value of 2'b10 means decrement value of coefficient by 1. Value of 2'b11 is reserved. Coefficient tuning is considered done if DIR feedback of zero (no change) is received on all lanes at the same time, this approach might take a long time, something may result in oscillatory behavior with settling down and DIR feedback does not converge on zero for all lanes at the same time. An alternate approach will be to use Windowed Filtering for DIR convergence.

DIR Convergence Criteria of Windowed Filtering

The simple convergence of DIR feedback is to get 2'b00 corresponding to precursor and postcursor coefficients on all relevant lanes, at times this can result in a scenario where the feedback oscillates between increment by 1 (2'b01) and decrement by 1 (2'b10) without settling down no change (2'b00) or DIR feedback of 2'b00 is not achieved for all lanes simultaneously. To overcome these issues, an alternate option of Windowed Filtering for DIR Convergence Criteria can be used.

Windowed Filtering for DIR Convergence Criteria involve requesting at least *D* number of coefficients, such that the maximum difference between the coefficients values in last *D* attempts are less than equal to a value *A*. The parameter *D* is Convergence Window Depth and *A* is Convergence Window Aperture.

5.4.3.1 Equalization Windowing Endpoint (Upstream Port) Sequence

The `svt_pcie_device_virtual_endpoint_mpipe_8g_equalization_seq` sequence will respond to equalization process initiated by downstream port. If the equalization process involves phase 2 and phase 3, in phase 2 it will provide means to fine tune downstream port PHY transmitter preset/coefficient and in Phase 3, it responds to downstream port's attempts to fine tune upstream port PHY's transmitter preset/coefficients.

This sequence performs equalization through Endpoint with MPIPE at Gen3 speed. The sequence takes control during equalization phase 2 when Endpoint (upstream port) is the equalization master. Sequence can be programed to tune preset, coefficients or both. Endpoint in MPIPE mode in equalization phase 2 sends presets and coefficients to link partner PHY and after getting acknowledgment from link partner, instructs Endpoint PHY to do evaluation of preset/coefficients (by asserting `rx_eq_eval`). Endpoint in MPIPE mode in equalization phase 3 receives the preset/coefficients from link partner and passes it on to Endpoint PHY through MPIPE signals (`TxDeemph`).

For preset tuning in phase 2, sequence can be programmed to send multiple presets, sequence keeps track of all the preset and FOM feedback they received. After sending all the presets, sequence picks the preset with highest FOM.

For coefficients tuning, sequence can be programmed through `dir_convergence_mode_8g`, either use Windowed Filtering method or feedback of zero on all lanes.

Coefficients tuning with feedback of zero on all lanes, sequence will calculate next set of coefficients based on DIR feedback, sends the new coefficients until DIR feedback precursor and postcursor coefficients are zero.

In Windowed Filtering convergence mode, sequence must be programed with `dir_convergence_window_depth_8g` and `dir_convergence_window_aperture_8g`.

- `dir_convergence_window_depth_8g` is the minimum number of coefficient sets to be tried.
- `dir_convergence_window_aperture_8g` is the acceptable difference between the minimum and maximum value of coefficient in last *D* tries, where *D* is `dir_convergence_window_depth_8g`.

Coefficient tuning during Windowed Filtering is considered done if the delta between the minimum and maximum coefficient is equal to or less than `dir_convergence_window_aperture_8g` for `dir_convergence_window_depth_8g`.

Note
This sequence will run only on upstream port (Endpoint) agent with MPIPE interface, otherwise results in fatal error.

Table 5-12 Controls

Control	Description
`rand bit tune_remote_tx_preset_8g`	If set, Endpoint will request preset in `preset_vector_8g` and find the best one through FOM feedback process.
`rand bit [15:0] preset_vector_8g`	List of presets to try. Each bit corresponds to a respective preset encoding, —that is, if `preset_vector_8g [0]` is set transmitter preset encoding "0000" will be requested. Same preset will be tried on all lanes as per the implemented algorithm.
`rand bit tune_remote_tx_coefficient_8g`	If set, sequence will do coefficient tuning in equalization phase 2.
`rand int unsigned max_iteration_for_coeff_convergence_8g`	Maximum number of iteration sequence will try for coefficient tuning process to converge. If coefficient tuning process does not converge even after reaching the maximum number of iterations, sequence will exit coefficient tuning with a warning.
`rand bit dir_convergence_mode_8g`	Coefficient tuning convergence mode • 0 - Feedback of zero on all lanes • 1 - Windowed filtering.
`rand int unsigned dir_convergence_window_depth_8g`	The minimum number of coefficient sets iteration to be tried before checking if convergence criteria is met. Only used if `dir_convergence_mode_8g` is windowed filtering.
`rand int unsigned dir_convergence_window_aperture_8g`	The acceptable difference between the minimum and maximum value of coefficient in last *N* iteration, when *N* is `dir_convergence_window_depth_8g`. Only used if `dir_convergence_mode_8g` is windowed filtering.

6 Gen4 Features

This chapter describes the Gen4 features available with the Synopsys PCIe Verification IP. This chapter discusses the following topics:

6.1 Gen4 Protocol Features

This section describes the Gen4 protocol features supported by PCIe SVT VIP and the usage notes to use in UVM environment. The VIP currently supports the 0.7 (November '16) version of the PCIe Gen4 specification. Implementation of Gen4 features is based on a draft specification and thus is subject to change as new versions of the specification are released. Gen4 feature can be enabled with either a library (VIP-LIBRARY-SVT + DesignWare-Regression) or suite (VIP-PCIE-G4-SVT) license. For more details, see Licensing Information.

6.1.1 Enabling Gen4

To enable Gen4 features, use the `SVT_PCIE_ENABLE_GEN4` define. The license manager checks for a Gen4 license when this define is set.

Note

To enable Gen4 protocol checks, set the specification version to 4.0.

`svt_pcie_device_configuration::pcie_spec_ver==svt_pcie_device_configuration::PCIE_SPEC_VER_4_0;`

Also, set the appropriate PIPE version with `svt_pcie_device_configuration::pipe_spec_ver.`

Currently, the following PIPE versions are supported:

- `PIPE_SPEC_VER_4`
- `PIPE_SPEC_VER_4_2`
- `PIPE_SPEC_VER_4_3`
- `PIPE_SPEC_VER_4_4`

6.1.2 Gen4 Feature Set

The following Gen4 features are supported:

6.1.2.1 Gen4 Speed Change

Gen4 speed is supported with the same command as other speeds. See `svt_pcie_pl_configuration::set_link_speed_values()`, for example the following will allow Gen1-Gen4 speeds.

```
cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_16_0G |

`SVT_PCIE_SPEED_8_0G |

`SVT_PCIE_SPEED_5_0G |

`SVT_PCIE_SPEED_2_5G);
```

6.1.2.2 Gen4 Equalization

The PCIe VIP supports the Gen4 changes to Equalization. Equalization is requested by using `svt_pcie_pl_service_request_equalization_sequence` PL service request.

Different FS/LF values are configured via `lf_value_16g[32]` and `fs_value_16g[32]` attributes declared in `svt_pcie_pl_configuration` class and these values advertised by VIP in TS1s during 16G Equalization phase 1.

VIP transmits a new preset/coefficient requests in Phase2 EP DUT/Phase3 RC DUT by using `svt_pcie_pl_service_queue_eq_tx_request_preset_coeff` PL service request.

6.1.2.2.1 Configuration Variables

There are a series of configuration variables in the `svt_pcie_pl_configuration` class and you can use these configuration variables as per your requirement to test a particular portion of the specification.

For more information on configuration variables, see “Class Listing” in the HTML class reference documentation.

The relevant controls are as follows:

- `enable_get_local_preset_coefficients`
- `enable_get_local_preset_coefficients_checking`

- `enable_local_lane_lf_checking`
- `enable_equalization_verification_mode`
- `enable_equalization_coefficients_checks`
- `highest_enabled_equalization_phase`
- `num_additional_ts_eq_before_trans_eq0`
- `num_additional_ts_eq_before_trans_eq1`
- `num_additional_ts_eq_before_trans_eq2`
- `num_additional_ts_eq_before_trans_eq3`
- `upstream_lanes_recovery_eq_phase0_timeout_ns`
- `upstream_lanes_recovery_eq_phase1_timeout_ns`
- `upstream_lanes_recovery_eq_phase2_timeout_ns`
- `upstream_lanes_recovery_eq_phase3_timeout_ns`
- `downstream_lanes_recovery_eq_phase1_timeout_ns`
- `downstream_lanes_recovery_eq_phase2_timeout_ns`
- `downstream_lanes_recovery_eq_phase3_timeout_ns`
- `reject_preset_coefficient_request`
- `tx_ts1_reset_eieos_interval_count_bit`
- `eq_rx_reset_eieos_interval`
- `min_spipe_preset_coefficients_delay`
- `max_spipe_preset_coefficients_delay`
- `enable_upconfigure_support = 1;`
- `attached_lf_16g[32]`
- `local_lf_16g[32]`
- `attached_fs_16g[32]`
- `local_fs_16g[32]`
- `lf_value_16g[32]`
- `fs_value_16g[32]`
- `min_eq_evaluation_delay[32]`
- `max_eq_evaluation_delay[32]`
- `max_eq_evaluation_delay[32]`
- `min_rx_eq_eval_delay[32]`
- `max_rx_eq_eval_delay[32]`
- `min_eq_preset_coeff_validation_delay[32]`
- `max_eq_preset_coeff_validation_delay[32]`
- `min_eq_eval_cycle_duration_in_ns`
- `max_eq_eval_cycle_duration_in_ns`
- `preset_to_coefficients_mapping_entry_valid_16g`
- `preset_to_coefficients_mapping_table_16g[16]`
- `expected_preset_to_coefficients_mapping_entry_enable_16g`
- `expected_preset_to_coefficients_mapping_table_16g[16]`

- upstream_receiver_preset_hint_16g[32]
- downstream_receiver_preset_hint_16g[32]
- upstream_preset_value_16g[32]
- downstream_preset_value_16g[32]
- quiesce_guarantee
- enable_rxeqeval_default_settings_vector

Example 6-1

```
cust_cfg.root_cfg.pcie_cfg.pl_cfg.highest_enabled_equalization_phase       = 3;
cust_cfg.endpoint_cfg.pcie_cfg.pl_cfg.highest_enabled_equalization_phase   = 3;
cust_cfg.root_cfg.pcie_cfg.pl_cfg.enable_equalization_verification_mode = 1;
cust_cfg.endpoint_cfg.pcie_cfg.pl_cfg.enable_equalization_verification_mode = 1;
cust_cfg.root_cfg.pcie_cfg.pl_cfg.enable_equalization_coefficients_checks = 1;
cust_cfg.endpoint_cfg.pcie_cfg.pl_cfg.enable_equalization_coefficients_checks = 1;
```

6.1.2.2.2 Status Variables

There are a series of status variables in `svt_pcie_pl_status` class to know the status of Equalization execution. You can use these status variables to know the status of the Equalization in different Phases, Phase0, Phase1, Phase2 and Phase3.

For more information on status variables, see “Class Listing” in the HTML class reference documentation.

The relevant controls are as follows:

- num_of_eq_phase1_successful
- num_of_eq_phase2_successful
- num_of_eq_phase3_successful
- num_of_tx_eq_preset_requests[32]
- num_of_tx_eq_coefficients_requests[32]
- num_of_rx_eq_preset_requests[32]
- num_of_rx_eq_coefficients_requests[32]
- num_of_rx_eq_preset_requests_rejected[32]
- num_of_rx_eq_coefficients_requests_rejected[32]
- perform_equalization
- request_equalization
- equalization_complete
- equalization_16g_complete
- equalization_16g_phase_1_successful
- equalization_16g_phase_2_successful
- equalization_16g_phase_3_successful
- eq_rx_requested_preset_coefficients_valid
- eq_rx_use_preset_bit[32]
- eq_rx_preset[32]
- eq_rx_precursor_coeff[32]
- eq_rx_cursor_coeff[32]
- eq_rx_postcursor_coeff[32]

- `eq_rx_reject_coefficients[32]`
- `eq_rx_reset_eieos_interval_count[32]`
- `eq_rx_request_equalization[32]`
- `eq_rx_quiesce_guarantee[32]`
- `rcvd_fs_lf_value_valid_16g`
- `rcvd_fs_value_16g[32]`
- `rcvd_lf_value_16g[32]`
- `rcvd_lf_value_16g[32]`
- `rcvd_upstream_receiver_preset_hint_16g[32]`
- `rcvd_downstream_receiver_preset_hint_16g[32]`
- `rcvd_upstream_preset_value_16g[32]`
- `rcvd_downstream_preset_value_16g[32]`
- `retimer_presence_detected`

Example 6-2

```
wait(endpoint_device.pcie_agent.pl.status.lane_status[1].eq_rx_precursor_coeff == 5);
wait(root_device.pcie_agent.pl.status.lane_status[1].eq_rx_precursor_coeff == 5);
```

6.1.2.2.3 Sequences

There are a series of service requests in the `svt_pcie_pl_service` class. Also, many of these are mapped to service sequences in `svt_pcie_pl_service_sequence_collection` class you can run directly.

For more information on sequences, see “Class Listing” in the HTML class reference documentation.

- `svt_pcie_pl_service_request_equalization_service`: Useful to request equalization process.
- `svt_pcie_pl_service_perform_equalization_service`: Useful in redo equalization case.
- `svt_pcie_pl_service_queue_eq_tx_request_preset_coeff`: Useful for queuing preset/coefficient requests in Phase2 EP DUT/Phase3 RC DUT.
- `svt_pcie_pl_service_queue_eq_direction_change_response`: Useful for queuing direction change response in PIPE mode.
- `svt_pcie_pl_service_queue_eq_figure_merit_response`: Useful for queuing figure of merit response in PIPE mode
- `svt_pcie_pl_service_initiate_retrain_link_sequence`: Useful for requesting the LTSSM to go to Recovery.

Example 6-3

```
svt_pcie_pl_service_request_equalization_sequence request_equalization;
request_equalization =
svt_pcie_pl_service_request_equalization_sequence::type_id::create("request_equalization
");
request_equalization.start(p_sequencer.root_virt_seqr.pcie_virt_seqr.pl_seqr );
```

Statistics

- `svt_pcie_eq_eval_cycle`
- `svt_pcie_eq_status`

Refer to the following tests at */pcie_svt/test/sverilog/tb_pcie_16g_device_system_uvm/tests:*

- *ts.eq_16g_ph0_ph1.sv*
- *ts.eq_16g_ph2_ph3.sv*
- *ts.equalization.sv*
- *ts.redo_eq_ep_requests_redo_eq_at_2_5g_rc_is_directed_link_reaches_8g_w_eq.sv*
- *ts.redo_eq_ep_requests_redo_eq_at_8g_rc_follows_w_eq.sv*
- *ts.redo_eq_rc_requests_redo_eq_at_2_5g_ep_follows_to_8g_w_eq.sv*

Also, see Gen4 TS tests at */pcie_test_suite_svt/test/sverilog/env/tests:*

- *ts.gen4_pl_recovery_equalization_phase0_to_phase1_case2_error.sv …*

6.1.3 EIEOS Format Change

The VIP supports the Electrical Idle Exit Ordered Set for Gen4. By default, it is on. It can be disabled with `svt_pcie_pl_configuration::enable_eieos_16g_0000_ffff`.

6.1.4 10-Bit Tag

The model supports the Gen4 10-bit tag feature. To enable 10-bit tag, set the `SVT_PCIE_ENABLE_10_BIT_TAGS` define. Also, the `svt_pcie_tl_configuration::remote_extend_tag_field_enabled` bit must be set to 1 for 10-bit and 8-bit tags. The PCIe specification version must be set to 4.0 or higher in the device configuration (SVT users) or in all of the various protocol layers (SVC users).

```
svt_pcie_device_configuration::pcie_spec_ver==svt_pcie_device_configuration::PCIE_SPEC_V
ER_4_0;
```

In order for the driver and requester application to generate 10-bit tags, the `max_num_tags` control must be set to 1024 to generate tags up to 3ff. A value of 256 means 8-bit tags. If this value is not set, then the model will consider it as 8-bit tag.

6.1.4.1 Setting Tag Properties for Requester ID

To allow 10-bit tag control on Requester ID (RID) basis, configure the driver before queuing any requests. A function named `set_req_id_tag_properties` needs to be called. There must be one call per Requester ID that the test intends to use. To preserve backwards compatibility, if a request is queued using an unconfigured Requester ID, then the driver will use the existing `max_num_tags` item in the configuration class. The `set_req_id_tag_properties` function will also define which portion of the tag pool is reserved for 8-bit tags and 10-bit tags.

When the driver is automatically managing the tag pool (that is, when user tags are disabled), it must know which kind of tag to issue when 8-bit tags and 10-bit tags are being used concurrently. To address this, the following function is added to the driver configuration:

```
set_req_id_tag_properties (bit [15:0] req_id, bit [31:0] max_num_tags, bit [31:0]
num_bits_for_concurrent_8bit_tags = 0)
```

where,

- `req_id` – The Requester ID that is being configured.
- `max_num_tags` – The maximum number of tags in the tag pool for a given Requester ID. If set to a value greater than 256, then 10-bit tags will be used. If set to a value less than or equal to 256, then 8-bit tags will be used. This variable must be set to 1024 to utilize the maximum 768 available tag from an RID. 10-bit tags will be utilized for NPR only.

- `num_bits_for_concurrent_8bit_tags` – This argument controls how many bits are available for 8-bit tag use if 8-bit and 10-bit tags are used concurrently. For example, if the number is set to 3, then the bottom 3 bits of the tag field will be reserved for 8-bit tags. Because 8-bit tags cannot be aliased for 10-bit tags, which implies that 8*3=24 tags will be removed from the 10-bit tag pool.

  For example, if `set_req_id_tag_properties` (1234, 1024, 1) is called, then the Requester ID 1234 will be enabled for 10-bit tags, with bit 0 reserved for 8 bit request. This means that a maximum of 2 requests can be outstanding to functions that support 8-bit tags ('b0000000000 and 'b0000000001). Additionally, 6 eligible tags have been removed from the 10-bit tag pool (tags 'b01000000000, 'b010000000001, 'b100000000000, 'b100000000001, 1100000000, and 110000000001). Set the `num_bits_for_concurrent_8bit_tags` argument to 0 if you do not want to use concurrent 8-bit and 10-bit tags.

  If a request is queued with an RID that has not been configured through `set_req_id_tag_properties()`, then the VIP will apply the default settings from `max_num_tags` in the configuration class and if 10-bit tags are enabled, no bits will be allocated for a 8-bit tag use. Concurrent tags are disabled by default unless explicitly enabled by the `set_req_id_properties` task. This maintains backwards compatibility for all the existing tests.

6.1.4.2 Limitation

- Tag properties should not be changed unless the driver is idle.
- Currently, VIP does not support the concurrent mode. Hence, the variable `num_bits_for_concurrent_8bit_tags` must be set to 0.

6.1.5 Retimer

The VIP will work in a system which supports one or two Retimers by adjusting the LTSSM behavior. The VIP does not provide a model of a Retimer, but instead model a system that contains one component (RC or EP) and one or two Retimers. When a Retimer is included, the interface changes to model the corresponding Pseudo port of the Retimer, therefore for an RC with a Retimer it is the Downstream Pseudo port of the Retimer that is the interface to the VIP, and for an EP with a Retimer it is the Upstream Pseudo port that is the interface.

6.1.5.1 Enabling Retimers

Retimers are enabled by configuring the VIP to include either one or two Retimers. This is done via the `set_retimer_present` configuration attribute.

Attribute	Description
set_retimer_present	Indicates the presence of Retimer. • When bit0 is set, the VIP will model a sub-system that includes at least one Retimer. This includes setting the Retimer present bit in the outgoing training sets. • When bit1 is set, the VIP will model a sub-system that includes two Retimers. This includes setting the two Retimers present bit in the outgoing training sets. • In addition to modifying the training sets, this attribute also impacts how the VIP will respond to RX Margin commands. This attribute will be used to decide when RX margin addresses the model will respond to with respect to requests for remote PHY Rx Margining. This feature is only enabled when `enable_ctrl_skp_support` is also enabled. • When bit0 is set, the VIP will respond to commands targeted to Retimer 1 including margin commands to receiver ports B (receiver_number 3'b010) and C (`receiver_number` 3'b011). • When bit1 is set, the VIP will respond to commands targeted to Retimer 2 including margin commands to receiver ports D (receiver_number 3'b100) and E (`receiver_number` 3'b101).

6.1.5.2 Retimer Presence Detect

In order to test a DUT's retimer presence detect capability, the VIP can set the retimer present bit in an outgoing TS at Gen1 with the `svt_pcie_pl_configuration::set_retimer_present` bit. Set to 1'b1 to enable.

In order for the VIP to detect a retimer, the bit `svt_pcie_pl_configuration:: retimer_presence_detect_supported` bit in the PHY layer configuration class must be set to 1. If a retimer is detected, then the LTSSM will behave according to the rules in the Rev 0.7 draft.

You can check the PL status class, `svt_pcie_pl_status:: retimer_presence_detected` to determine if the VIP detected a retimer during link training. Note that bit0 refers to the first retimer being present while bit1 refers to the second retimer being present.

6.1.6 Retimer Equalization

When the VIP is used as a component plus a Retimer, the equalization is modified to reflect that the interface is now one of the pseudo ports on the Retimer and so that this equalization now depends on internal events happening on the internal port in-between the Retimer and the component. There are seven new configuration attributes used to schedule these internal events. These attributes are only used if the svt_pcie_pl_configuration::set_retimer_present is set to a non-zero value indicating that the VIP includes at least one Retimer.

Table 6-1 Retimer Equalization

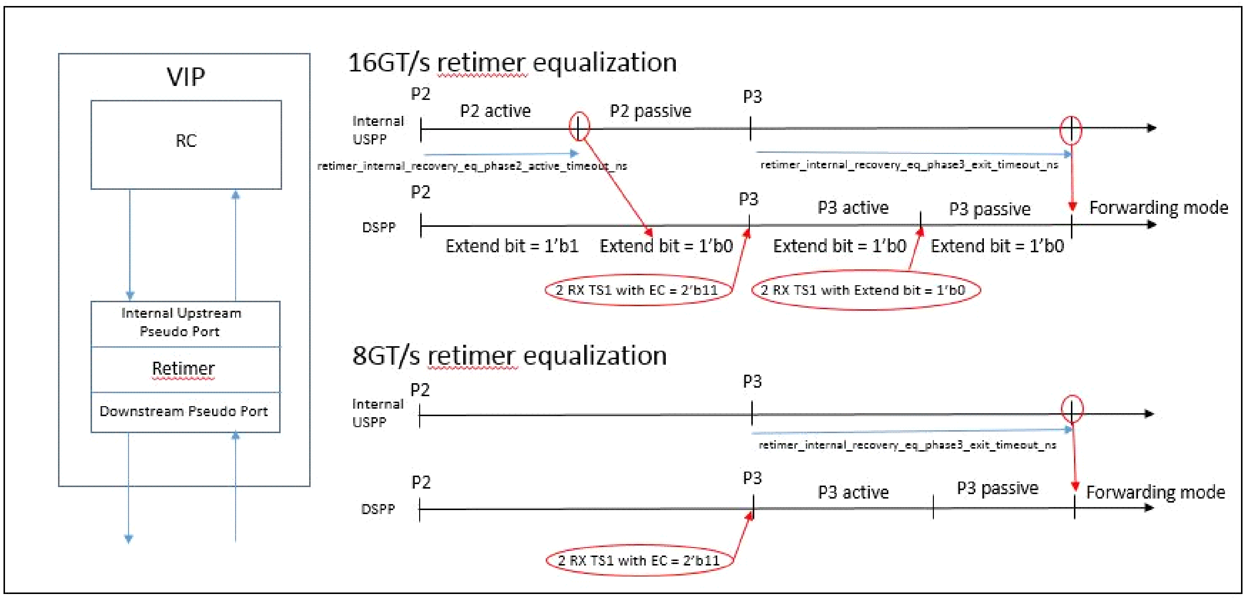
Attribute	Applicable	Default value	Description
retimer_internal_recovery_eq_phase2_active_timeout_ns	RC	10000	Specifies the timeout in NS for the internal upstream lanes from the Retimer to upstream device in Recovery.Equalization.Phase2. This value is used to control the Retimer Equalization Extend bit of the TS1 transmitted on the downstream lanes from the Retimer in Recovery.Equalization.Phase2. Only used in 16GT/s when VIP is modeling a system that includes a component (RC or EP) and at least one Retimer.
retimer_internal_recovery_eq_phase3_exit_timeout_ns	RC	12000	Specifies the timeout in NS for the internal upstream lanes from the Retimer to upstream device to exit Phase3 in Recovery.Equalization.Phase3. This value is used to control the exit of Phase3 in the downstream lanes from the Retimer in Recovery.Equalization.Phase3. Used in both 8GT/s and 16GT/s when VIP is modeling a system that includes a component (RC or EP) and at least one Retimer.
retimer_internal_recovery_eq_phase3_started_timeout_ns	EP	12000	Specifies the timeout in NS for the internal downstream lanes from the Retimer to downstream device from start of Recovery.Equalization.Phase2 until it enters Recovery.Equalization.Phase3. This value is used to control when the upstream lanes from the Retimer change to Recovery.Equalization.Phase3. Only used in 16GT/s when VIP is modeling a system that includes a component (RC or EP) and at least one Retimer.
retimer_internal_recovery_eq_phase3_active_timeout_ns	EP	10000	Specifies the timeout in NS for the internal downstream lanes from the Retimer to downstream device in Recovery.Equalization.Phase3. This value is used to control the Retimer Equalization Extend bit of the TS1 transmitted on the upstream lanes from the Retimer in Recovery.Equalization.Phase3. Only used in 16GT/s when VIP is modeling a system that includes a component (RC or EP) and at least one Retimer.
retimer_internal_recovery_eq_phase3_active_completed_timeout_ns	EP	12000	Specifies the timeout in NS for the internal downstream lanes from the Retimer to downstream device to exit Phase3 active in Recovery.Equalization.Phase3. The value is given from start of Recovery.Equalization.Phase2. This value is used to control the entry of Phase3 in the upstream lanes from the Retimer in Recovery.Equalization.Phase2. Only used in 8GT/s when VIP is modeling a system that includes a component (RC or EP) and at least one Retimer.

Table 6-1 Retimer Equalization

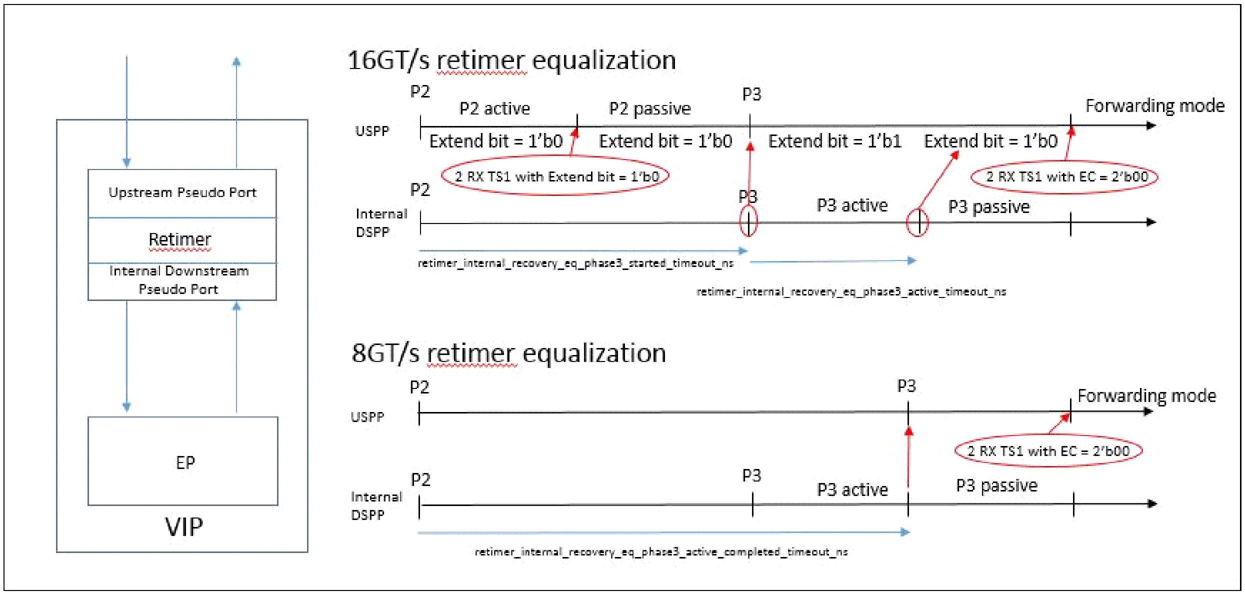
Attribute	Applicable	Default value	Description
retimer_internal_recovery_eq_force_timeout_eios_detected_timeout_ns	EP	1000	Specifies the timeout in NS for the Retimer internal ports to detect EIOS during `Force.Timeout`. This value is used to control as part of the exit condition from `Force.Timeout` to the entry of `Recovery.Speed`. Used in both 8GT/s and 16GT/s when VIP is modeling a system that includes a component (RC or EP) and at least one Retimer.
retimer_recovery_eq_force_timeout_timeout_ns	EP	48000	Specifies the timeout in NS for the Retimer `Force.Timeout` state timeout. Used in both 8GT/s and 16GT/s when VIP is modeling a system that includes a component (RC or EP) and at least one Retimer.

The following figures shows the relationship between events on the externally visible ports and the internally configuration driven events.

- VIP is RC and Retimer



- VIP is EP and Retimer



6.1.7 Retimer Latency

Table 6-2 Remiter Latency

Attribute	Description
rx_retimer_latency_2_5g_ns	Programmable latency used to mimic the presence of a retimer on the Rx data path. Setting this value in ns will result in the VIP adding an internal latency to the PIPE interface to the value specified rounded down to the nearest PIPE clock when running at 2.5G. This should be left at 0 unless a retimer is present and you want to intentionally add latency. Setting this to a large value may impact simulation speed.
rx_retimer_latency_5g_ns	Programmable latency used to mimic the presence of a retimer on the Rx data path. Setting this value in ns will result in the VIP adding an internal latency to the PIPE interface to the value specified rounded down to the nearest PIPE clock when running at 5G. This should be left at 0 unless a retimer is present and you want to intentionally add latency. Setting this to a large value may impact simulation speed.
rx_retimer_latency_8g_ns	Programmable latency used to mimic the presence of a retimer on the Rx data path. Setting this value in ns will result in the VIP adding an internal latency to the PIPE interface to the value specified rounded down to the nearest PIPE clock when running at 8G. This should be left at 0 unless a retimer is present and you want to intentionally add latency. Setting this to a large value may impact simulation speed.
rx_retimer_latency_16g_ns	Programmable latency used to mimic the presence of a retimer on the Rx data path. Setting this value in ns will result in the VIP adding an internal latency to the PIPE interface to the value specified rounded down to the nearest PIPE clock when running at 16G. This should be left at 0 unless a retimer is present and you want to intentionally add latency. Setting this to a large value may impact simulation speed.

6.1.8 Flow Control Credit Scaling

Flow Control Credit Scaling is enabled by the `svt_pcie_dl_configuration::enable_dl_feature_handshake` member. The scale values for each VC/type are configured via the `init_[p|np|cpl]_[data|hdr]_tx_credit_fc_scale[8]` attributes in TL configuration (`svt_pcie_tl_configuration`) to enable different scaling factors. For more details about controls, see the HTML class reference documentation.

The relevant controls are as follows:

- `svt_pcie_device_configuration::pcie_spec_ver`
- `svt_pcie_tl_configuration`
  - `init_[p|np|cpl]_[data|hdr]_tx_credit_fc_scale[8]`
- `svt_pcie_tl_status`
  - `local_fc_scale_mode[48]`
  - `remote_fc_scale_mode[48]`
- `svt_pcie_dl_configuration`
  - `enable_dl_feature_handshake`
  - `local_dl_feature_supported`
- `svt_pcie_dl_status`

- `local_dl_feature_supported`
- `remote_dl_feature_supported`
- `dl_feature_status`

Additional checks in Transaction layer and Data Link layer are as follows:

- Transaction layer
  - `MSGCODE_PCIESVC_TL_FC_HDR_SCALE_CHANGED`
  - `MSGCODE_PCIESVC_TL_FC_DATA_SCALE_CHANGED`
  - `MSGCODE_PCIESVC_TL_FC_DATA_HDR_SCALE_ZERO`
  - `MSGCODE_PCIESVC_TL_FC_DATA_HDR_SCALE_NON_ZERO`
- Data Link layer
  - `MSGCODE_PCIESVC_DL_FEATURE_ACK_DEASSERTED`
  - `MSGCODE_PCIESVC_DL_FEATURE_ACK_TIMEOUT`
  - `MSGCODE_PCIESVC_DL_FEATURE_RSVD_NON_ZERO`

Code snippet:

```
  cfg.rc_cfg.pcie_cfg.dl_cfg.enable_dl_feature_handshake=1;

 //Configure hdr and data Scale value.

cfg.rc_cfg.pcie_cfg.tl_cfg.init_p_hdr_tx_credit_fc_scale[0]=`SVT_PCIE_FC_SCALE_FACTOR_X1
;

cfg.rc_cfg.pcie_cfg.tl_cfg.init_p_data_tx_credit_fc_scale[0]=`SVT_PCIE_FC_SCALE_FACTOR_X
1;

cfg.rc_cfg.pcie_cfg.tl_cfg.init_np_data_tx_credit_fc_scale[0]=`SVT_PCIE_FC_SCALE_FACTOR_
X1;

cfg.rc_cfg.pcie_cfg.tl_cfg.init_np_hdr_tx_credit_fc_scale[0]=`SVT_PCIE_FC_SCALE_FACTOR_X
1;

cfg.rc_cfg.pcie_cfg.tl_cfg.init_cpl_hdr_tx_credit_fc_scale[0]=`SVT_PCIE_FC_SCALE_FACTOR_
X1;

cfg.rc_cfg.pcie_cfg.tl_cfg.init_cpl_data_tx_credit_fc_scale[0]=`SVT_PCIE_FC_SCALE_FACTOR
_X1;
```

The default value is zero, therefore you must select at least one of scaling factor as indicated in the above Scale factor (01).

```
01(`SVT_PCIE_FC_SCALE_FACTOR_X1),
10(`SVT_PCIE_FC_SCALE_FACTOR_X4),
11(`SVT_PCIE_FC_SCALE_FACTOR_X16)
```

The VIP will issue an error if the above TL configuration variables are not set after enabling `enable_dl_feature_handshake`.

6.1.9 Rx Margining

6.1.9.1 Initiating Remote PHY Rx Margining by Upstream Component (RC) Using CTRL-SKP OS

6.1.9.1.1 Control SKP OS

The Control SKP OS is used for the Upstream component to request parity and margining information from other components/ports. This OS is transmitted in `Configuration.Idle` and `Recovery.Idle` just prior to the SDS in 16GT/s and is used for parity initialization. It is also used in L0, where every other SKP transmitted is a CRTL-SKP OS.

6.1.9.1.2 Enabling CTRL-SKP OS and Rx Margining Features

CTRL-SKP OS and Rx Margining is enabled by the `svt_pcie_pl_configuration::enable_ctrl_skp_support` attribute in PHY layer configuration class. The default value of this attribute is '0' and when configured to '1', it will generate every other SKP OS during L0 as a CRTL-SKP instead of a regular SKP OS. This feature must be enabled to support Rx Margining because it uses the CTRL-SKP OS to transmit information.

In addition to enabling CTRL-SKP OS, you can also select the response mode when the VIP is acting as EP and Retimer. In this mode, the VIP will respond to CTRL-SKP OS's that are sending Rx Margin commands to a Retimer model by the VIP based on the configuration of the `svt_pcie_pl_configuration::set_retimer_present` attribute. The two response modes supported is manual mode and automatic mode configured using `svt_pcie_pl_configuration::rx_margin_automatic_response_mode`. In automatic mode, the VIP will generate response to the Rx Margin commands based on configured attributes and internal state of margin operations, while in manual mode the response is provided by the testbench via the `svt_pcie_pl_service::RX_MARGIN_RESPONSE`.

6.1.9.1.3 Using Rx Margin Command

Rx Margining commands can be transmitted through service channel request `svt_pcie_pl_service::CTRL_SKP_CMD`. This service request loads up from an enum of Rx Margining commands with lane number, indicating which lane the CTRL-SKP OS is going to be transmitted on, receiver number, receiver address, margin type and step direction (in case of Step Margin commands). This is only applicable when the model is at 16GT/s and in L0. The commands which can be transmitted through this service request are listed inTable 6-3. The CRTL-SKP service request is only applicable to the RC while the `INCREMENT_MARGIN_ERROR_COUNT` is applicable to both RC and EP.

Table 6-3 Rx Margining Commands

Command Name	Applicable	Command Attributes	Description
ACCESS_RETIMER_REGISTER (Optional)	RC	retimer[2:0],register_addr[7:0]	
REPORT_MARGIN_CONTROL_CAPABILITIES	RC	receiver_number[2:0]	Initiate a Margin command to return configuration attribute `MARGIN_CONTROL_CAPABILITIES` in Margin Payload response for associated receiver.

Table 6-3 Rx Margining Commands

Command Name	Applicable	Command Attributes	Description
REPORT_M_NUM_VOLAGE_STEPS	RC	receiver_number[2:0]	Initiate a Margin command to return configuration attribute `M_NUM_VOLAGE_STEPS` in Margin Payload response for associated receiver.
REPORT_M_NUM_TIMING_STEPS	RC	receiver_number[2:0]	Initiate a Margin command to return configuration attribute `M_NUM_TIMING_STEPS` in Margin Payload response for associated receiver.
REPORT_M_MAX_TIMING_OFFSET	RC	receiver_number[2:0]	Initiate a Margin command to return configuration attribute `M_MAX_TIMING_OFFSET` in Margin Payload response for associated receiver.
REPORT_M_MAX_VOLTAGE_OFFSET	RC	receiver_number[2:0]	Initiate a Margin command to return configuration attribute `M_MAX_VOLTAGE_OFFSET` in Margin Payload response for associated receiver.
REPORT_SAMPLING_RATE_VOLTAGE	RC	receiver_number[2:0]	Initiate a Margin command to return configuration attribute `SAMPLING_RATE_VOLTAGE` in Margin Payload response for associated receiver.
REPORT_SAMPLING_RATE_TIMING	RC	receiver_number[2:0]	Initiate a Margin command to return configuration attribute `SAMPLING_RATE_TIMING` in Margin Payload response for associated receiver.
REPORT_M_SAMPLE_COUNT	RC	receiver_number[2:0]	Initiate a Margin command to return configuration attribute M_SAMPLE_COUNT in Margin Payload response for associated receiver.
REPORT_M_MAX_LANES	RC	receiver_number[2:0]	Initiate a Margin command to return configuration attribute `M_MAX_LANES` in Margin Payload response for associated receiver.
SET_ERROR_COUNT_LIMIT	RC	receiver_number[2:0]; error_count_limit[5:0]	Initiate a Margin command to set the `error_count_limit` for the associated receiver.

Table 6-3 **Rx Margining Commands**

Command Name	Applicable	Command Attributes	Description
GO_TO_NORMAL_SETTINGS	RC	receiver_number[2:0]	Initiate a Margin command to return the associated receiver to its normal setting and terminate margin command.
CLEAR_ERROR_LOG	RC	receiver_number[2:0]	Initiate a Margin command to clear the `error_count` for the associated receiver.
STEP_MARGIN_TO_TIMING_OFFSET	RC	receiver_number[2:0]; step_direction; steps[5:0]	Initiate a Margin command to start a timing step margin operation for the associated receiver.
STEP_MARGIN_TO_VOLTAGE_OFFSET	RC	receiver_number[2:0]; step_direction; steps[5:0]	Initiate a Margin command to start a voltage step margin operation for the associated receiver.
LOAD_REGISTER	RC/EP	bit[7:0] value[3]	CTRL_SKP sub command to load up any value into the next CTRL-SKP OS using internal registers. This can also be used to send error CTRL-SKP OS.
CRTL_SKP_CMD	RC	sub command; lane	
INCREMENT_MARGIN_ERROR_COUNT	RC/EP	count (default 1); lane	Increment the current error count of the associated receiver lane.

6.1.9.1.4 Rx Margin Command Generation From RC

RC (Upstream component) lane margin of remote PHY in Retimer using CTRL-SKP (VIP acting as MAC).

- Wait for RC to reach 16G speeds and then enter L0 with data stream turned on.
- Issue the service request to send `REPORT_MARGIN_CONTROL_CAPABILITIES` command using `CTRL_SKP_CMD`.
- Wait for RC to receive a CTRL-SKP with `margin_type` set to 3'b001 and `receiver_number` matching requested address using attributes in the `svt_pcie_ctrl_skp_receiver_status` class.
- Then wait for the reception of a reflected `No Command`. There is no need to issue this command using the service request since the VIP will automatically send a `No Command` if no other command is issued.
- If required, issue additional `REPORT` commands one at a time and wait for their reply using the method above to request additional configuration values from port that is going to be margined.
- Issue the service request to send `SET_ERROR_COUNT_LIMIT` command using `CTRL_SKP_CMD`.
- Wait for RC to receive a CTRL-SKP with `margin_type` set to 3'b010 and `receiver_number` matching requested address and expected returned error count limit using attributes in the `svt_pcie_ctrl_skp_receiver_status` class.
- Wait for the reception of a reflected `No Command`. There is no need to issue this command using the service request since the VIP will automatically send a `No Command` if no other command is issued.

- Issue the service request to send `STEP_MARGIN_TO_TIMING_OFFSET` or `STEP_MARGIN_TO_VOLTAGE_OFFSET` command with appropriate Payload values using `CTRL_SKP_CMD`.
- Keep monitoring the status of the received CTRL-SKP OS's for margin status and error count and take appropriate action based on the received status. The `svt_pcie_ctrl_skp_receiver_status` gets updated every time the VIP receives a new CTRL-SKP OS and the testbench can monitor the reception by waiting for changes to the `num_ctrl_skp_received`.

6.1.9.1.5 Rx Margin Response Generation From EP (Automatic Response Mode)

EP (Downstream component) lane margin of remote PHY in Retimer using CTRL-SKP (VIP acting as Retimer PHY).

- Configure CTRL-SKP response attributes for each Retimer device, the VIP is modeling in addition to configuring the CTRL-SKP delay (`margin_response_time`).
- Wait for receiver CTRL-SKP to indicate start of margin operation.
- Issue the service request to `INCREMENT_RX_MARGIN_BIT_ERROR_COUNT` whenever the testbench wants to pretend it detected a bit error. This would make the VIP update the status returned through the CTRL-SKP.

6.1.9.1.6 Rx Margin Response Generation From EP (Manual Mode)

Response has to be manually loaded through the service channel request `svt_pcie_pl_service::RX_MARGIN_RESPONSE`. The response can be loaded with `ctrl_skp_lane_num` (which lane to transmit CTRL-SKP OS), `receiver_number`, `margin_type` and `margin_payload`.

Example 6-4 Usage Example

// Load REPORT_M_NUM_VOLAGE_STEPS using the service call for random number range.

```
for (int i=min_ctrl_skp_lane_no; i<=max_ctrl_skp_lane_no; i++) begin
    `svt_xvm_do_on_with(ctrl_skp_cmd_req,vip_seqr.pcie_virt_seqr.pl_seqr,{
          command == svt_pcie_pl_service::REPORT_M_NUM_VOLAGE_STEPS;
           ctrl_skp_lane_num == i;
            receiver_number   == 3'b010;
        });
 end
//preparing response for REPORT_M_NUM_VOLAGE_STEPS
@(dut_status.pcie_status.pl_status.ctrl_skp_receiver_status.num_ctrl_skp_received[max_ct
rl_skp_lane_no]);  // Wait for DUT to receive the rx margin command
for (int i=min_ctrl_skp_lane_no; i<=max_ctrl_skp_lane_no; i++) begin
`svt_xvm_do_on_with(ctrl_skp_response_req,dut_seqr.pcie_virt_seqr.pl_seqr,{
       ctrl_skp_lane_num == i;
        margin_type       == 3'b001;
        receiver_number   == 3'b010;
        rx_margin_response_persistent == 1'b1; //Enable continuous response
        margin_payload    == {{1'b0}, MNumVoltageSteps};
       });
  end
```

6.1.9.2 Initiating Local PHY Rx Margining by Either Upstream Component (RC) or Downstream

Component (EP) Using MBI

6.1.9.2.1 Local PHY Rx Margining VIP Acting as MAC (MPIPE Mode)

RC (Upstream component) or EP (Downstream component) lane margin of local PHY using MBI (VIP acting as MAC). There are two different methods for generating MBI requests when the VIP is modeling the MAC side of the MBI (Message Bus Interface).

The MBI_CMD method uses a fine-grained command specific to the MBI using a new service request type `svt_pcie_pl_service::MBI_CMD`. This service request can be used to send `NOP`, `WRITE_UNCOMMITED`, `WRITE_COMMITED` and `WRITE_ACK` commands.

The CTRL-SKP method uses the same service requests as the remote PHY CTRL-SKP service requests by mapping RC CTRL-SKP requests of type (`CLEAR_ERROR_LOG`, `STEP_MARGIN_TO_VOLTAGE_OFFSET`, `STEP_MARGIN_TO_TIMING_OFFSET` and `GO_TO_NORMAL_SETTINGS`) with the `receiver_number` set to 3'b010, or EP with `receiver_number` set to 3'b110 to corresponding MBI requests. The CTRL_SKP will not be updated and the device will continue to send CTRL-SKP with `No Command`.

- MBI_CMD
  - Wait for RC to reach 16G speeds and then enter L0 with data stream turned on.
  - Issue the service request to send `WRITE_UNCOMMITED` and/or `WRITE_COMMITED` commands with appropriate `MBI_REGISTER` and `MBI_DATA` values using `MBI_CMD`.
  - Keep monitoring the status of the internal `Rx_Margin_Status` registers to detect changes to execution status, sample count and error count and take appropriate action based on the received status changes. The internal registers get updated every time a `WRITE_COMMITED` to the status register is detected on the MBI from the PHY.
- CTRL-SKP
  - Wait for RC to reach 16G speeds and then enter L0 with data stream turned on.
  - Issue the service request to send `STEP_MARGIN_TO_TIMING_OFFSET` or `STEP_MARGIN_TO_VOLTAGE_OFFSET` command with appropriate Payload values using `CTRL_SKP_CMD`.
  - Keep monitoring the status of the internal `Rx_Margin_Status` registers to detect changes to execution status, sample count and error count and take appropriate action based on the

received status changes. The internal registers get updated every time a `WRITE_COMMITED` to the status register is detected on the MBI from the PHY.

Table 6-4 MBI Commands

Command Name	Applicable	Command Attributes	Description
MBI_CMD	RC/EP	mbi_cmd; mbi_lane_num; mbi_addr[11:0]; mbi_data[7:0]	Initiate a transmit of a MBI command on the MBI PIPE signals.
INCREMENT_MARGIN_ERROR_COUNT	RC/EP	count (default is 1)	Lane increment the current sample count of the associated receiver lane.
NOP	RC/EP	bit[3:0]	
WRITE_UNCOMMITTED	RC/EP	bit[3:0]	The `WRITE_UNCOMMITTED` command performs an uncommitted write.
WRITE_COMMITTED	RC/EP	bit[3:0]	The `WRITE_COMMITTED` command performs a committed write.
WRITE_ACK	RC/EP	bit[3:0]	The `WRITE_ACK` command transmits a write ACK.

6.1.9.2.2 Local PHY Rx Margining VIP Acting as PHY (SPIPE Mode)

RC (Upstream component) or EP (Downstream component) lane margin of local PHY using MBI (VIP acting as PHY).

- Configure MBI response delay (`write_ack_delay`).
- Wait for receiver MBI to indicate start of margin operation.
- Issue the service request to `INCREMENT_RX_MARGIN_SAMPLE_COUNT` and `INCREMENT_RX_MARGIN_BIT_ERROR_COUNT` whenever the testbench wants to pretend it detected a change in this internal counter. This would make the VIP generate MBI writes to status registers in MAC.

6.1.10 Limitations

The PCIe VIP does not support the following Gen4 features:

- Retimer specific compliance rules
- CompLoadBoard rules

6.2 Using SKP Ordered Sets

The SKP interval transmission and reception can be controlled in the PCIe VIP through the following attributes in the svt_pcie_pl_configuration class.

The VIP will transmit a SKP OS based on these settings:

- Gen1 and Gen2
  - min_tx_skp_interval_in_symbol_times
  - max_tx_skp_interval_in_symbol_times

- Gen3
  - min_tx_skp_interval_in_blocks
  - max_tx_skp_interval_in_blocks

The VIP will check the reception of the SKP OS from the DUT based on these settings:

- Gen1 and Gen2
  - min_rx_skp_interval_in_symbol_times
  - max_rx_skp_interval_in_symbol_times
- Gen3
  - min_rx_skp_interval_in_blocks
  - max_rx_skp_interval_in_blocks

This table shows the allowed ranges and the default setting for the SKP interval attributes.

Table 6-5 SKP Ordered Set Configuration Members

Type	Range	Default	Description
max_tx_skp_interval_in_symbol_times			
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			
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			
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			
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			
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			
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 6-5 SKP Ordered Set Configuration Members (Continued)

Type	Range	Default	Description
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			
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			
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			
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			
Integer	1-5	3	Maximum number of SKP symbols in an ordered set at 8GT/s. Acceptable values: 1 - 5.

The min/max_tx_skp_interval_in_<xxx >settings are randomized in the VIP to the min and max settings of the attributes. For example, the default setting for min_tx_skp_interval_in_symbol_times is 1180 symbol times. The default setting for max_tx_skp_interval_in_symbol_times is 1538 symbol times. The PCIe VIP will transmit the SKP OS based on these 2 settings. The SKP interval transmission will be randomized between the min and max values.

If the SKP OS interval needs to be set to a specific value, then set the min and max values to the same number. For example, to have the PCIe VIP transmit the SKP interval at 1275, the following setting would be done.

```
 //Configure min/max skip interval to a set value.
cfg.rc_cfg.pcie_cfg.pl_cfg.min_tx_skp_interval_in_symbol_times = 1275;
cfg.rc_cfg.pcie_cfg.pl_cfg.max_tx_skp_interval_in_symbol_times = 1275;
```

Similarly for the SKP interval in blocks for gen3, you would do the same thing.

```
//Configure min/max skip interval to a set block value.
cfg.rc_cfg.pcie_cfg.pl_cfg.min_tx_skp_interval_in_blocks = 372;
cfg.rc_cfg.pcie_cfg.pl_cfg.max_tx_skp_interval_in_blocks = 372;
```

For the min/max_rx_skp_interval_in_symbol_times and min/max_rx_skp_interval_in_blocks, the PCIe VIP will check for the reception of the SKP OS from the DUT. Again, the SKP interval will be checked based on the randomized min and max values. If the VIP is required to check the SKP interval reception for a particular value, then set the min and max values to be the value that is needed.

The PCIe VIP also allows for the number of SKP symbols to be included in the SKP OS. The default setting for both min and max is 3, so 3 SKP symbols will be sent in the SKP OS. The SKP OS can be adjusted to send a random number of SKP symbols and set to another value such as 5 by setting the min and max numbers to be different or the same number.

6.3 Address Translation Services

The PCIe VIP supports Address Translation Services, ATS, through the custom application interface. In particular, the model supports access to the AT field within the Memory Read and Memory Write TLPs. If the bit is set than the given request has been translated via the ATS protocol. It is up to the end user's application or test to utilize the AT bit field in the support of ATS.

For additional information, "Creating and Using Custom Applications" on page 180

6.4 PCIe VIP Bare COM Support

6.4.1 Background

In 2.5 GT/s or 5.0 GT/s transmission (8b10b encoding only), a normal transmitted SKP Ordered Set consists of a *COM* Symbol (K28.5) followed by three SKP Symbols (K28.0). The term *Bare COM* is used in reference to the PIPE interface during either 2.5 GT/s or 5.0 GT/s transmission where the datum being received across that interface is a *COM* symbol (K28.5 symbol) with the `RxStatus` equal to 2 indicating "1 SKP removed".

In a real PCIe system, the *Bare COM* PIPE scenario will be the result of a number of repeaters or re-timers each removing "1 SKP" Ordered Set. The last one in the string of three will then result in this *Bare COM* with the remaining three SKP symbols having been removed by previous repeaters leaving only the *COM* and the "1 SKP removed" `RxStatus` of 2 as indicated.

6.4.2 Enabling VIP Bare COM transmission (to mimic the system scenario)

When the PCIe VIP's `MIN_TX_SKP_SYMBOLS_IN_ORDERED_SET_VAR` variable is set to 0 in a test for the SPIPE model (`IS_PIPE_MASTER` is 0 and `PHY_INTERFACE_TYPE` is 0), the possibility of transmitting a *Bare COM* as a SKP Ordered Set is enabled. If that PCIe VIP SPIPE model's `MAX_TX_SKP_SYMBOLS_IN_ORDERED_SET_VAR` variable is also set to 0, every SKP Ordered Set is assured to be a *Bare COM*. This is the method that guarantees *Bare COM* transmission.

The above scenario results in a *COM* with `RxStatus` equal to 2 being transmitted on the PIPE interface (*Bare COM*).

6.4.2.1 Misconfiguration Warning Message

If the `MIN_TX_SKP_SYMBOLS_IN_ORDERED_SET_VAR` is set to 0 and either the VIP model's `IS_PIPE_MASTER` parameter is 1 (MPIPE) or the `PHY_INTERFACE_TYPE` is not 0 (SERDES or PMA models), then the VIP will issue the following warning (and, as indicated, will set the number of SKP symbols to the normal 3):

```
WARNING: endpoint0.port0.pl0.: 'Bare COM' (num_skp_symbols_to_send == 0) is illegal for
'IS_PIPE_MASTER' == 1 (s/b 0) OR 'PHY_INTERFACE_TYPE = 0'(s/b 0 - PIPE) to support 'Bare
COM' SKP OS) - set 'num_skp_symbols_to_send' to 3.
```

6.4.3 Enabling VIP Bare COM Reception

The reception of a *Bare COM* in any PCIe VIP PIPE model release that includes the *Bare COM* support in "Enabling VIP Bare COM transmission (to mimic the system scenario)" needs no special setup and is available by default for 2.5 GT/s and 5.0 GT/s data rates.

6.4.3.1 Ordered Set Checker Warnings

Bare COM reception in a PCIe MPIPE model will result in the following warning:

```
WARNING: endpoint0.port0.pl0.ordered_set_checker0.: Detected undersize SKP ordered set
with 1 COM and 1 SKP symbols. Expecting 3 SKP symbols.
```

The warning indicates "1 COM and 1 SKP" (rather than the expected "0 SKP") due to the `ordered_set_checker` interpreting the `RxStatus` equal to 2 being received with the *COM* as having removed a received SKP ("1 SKP removed" status) and therefore including that SKP Symbol as having been received (as would have been the case if a real PHY was attached to the PIPE).

To suppress the above warnings in any test that intentionally transmits *Bare COMs*, all the receiving model's `ordered_set_checker` modules requires message suppression using the `MSGCODE_PCIESVC_ORDERED_SET_CHECKER_UNDERSIZE_SKIP` message suppression code.

6.5 OBFF Feature Support

6.5.1 Basic Attributes

VIP support for OBFF feature is disabled by default. You can enable the OBFF feature—that is, to track CPU states either using `WAKE#` signal or OBFF MSG requests using `svt_pcie_tl_configuration::enable_obff` attribute.

Note
The `enable_obff` attribute must be set to the same value in VIP as the DUT's `enable_obff` value. As per base PCIe specifications, OBFF messaging and `WAKE#` signal based mechanism cannot be enabled simultaneously.

When the OBFF feature is enabled, VIP indicates the current CPU state using `svt_pcie_tl_status::cpu_state` attribute.

- When programmed as RC, VIP updates CPU state after it has transmitted OBFF message or after it has completely generated `WAKE#` signal pattern for CPU state transition.
- When programmed as EP, VIP updates CPU state after it has received OBFF message or after it has decoded complete `WAKE#` signal pattern for CPU state transition. The previous CPU state is made available via `svt_pcie_tl_status::prev_cpu_state` attribute.

6.5.2 Signal Connectivity

- In an environment with single EP and single RC/Bridge, `wake_n` of VIP must be connected to `wake_n` of DUT.
- In an environment where there are multiple EP VIP instances, VIP's respective `wake_n` signal must be tied together and connected to `wake_n` signal of RC/Bridge.
- In an environment with multiple PCIE hierarchies, all the `wake_n` signals within one hierarchy should be tied together. The `wake_n` signal from one PCIE hierarchy should not be tied to `wake_n` signal from another PCIE hierarchy.

Note
Users do not need to connect external pull up to this signal.

6.5.3 Generating CPU State Transition Using RCVIP

RC VIP can be directed to indicate CPU state transition to EP using `svt_pcie_tl_service::CHANGE_CPU_STATE` service request. Based on `svt_pcie_tl_configuration::enable_obff` programming, the `CHANGE_CPU_STATE` service request will either generate OBFF messages or `WAKE#` signal patterns.

RC VIP can be programmed to generate OBFF messages to indicate CPU state using one of the following approaches:

1. Using `svt_pcie_driver_app_transaction` objects' sequence where `tlp_type` is MSG and `message_type` is OBFF.
2. Using `svt_pcie_tlp` transaction objects' sequence where `tlp_type` is MSG and `message_type` is OBFF.
3. Using `svt_pcie_tl_service::CHANGE_CPU_STATE` service request with appropriate `obff_code` and programming `svt_pcie_tl_configuration::enable_obff = {1,2}`.

Note
When using this service request with OBFF message based mechanism, you must map `` `SVT_PCIE_DEFAULT_APPLICATION_NUMBER_TL `` application ID to a `Requester ID` as per the testbench requirements. If not programmed, default `Requester ID` of 0 will be mapped to application ID value of `` `SVT_PCIE_DEFAULT_APPLICATION_NUMBER_TL ``.

RC VIP can be programmed to generate `WAKE# signal` patterns to indicate CPU state using the following approach:

By programming `svt_pcie_tl_configuration::enable_obff = 3` and using `svt_pcie_tl_service::CHANGE_CPU_STATE` service request. When RC VIP receives `CHANGE_CPU_STATE` service request it starts generating `WAKE#` signal patterns after the delay of `svt_pcie_tl_configuration::wake_transition_delay_in_ns` NS.

RC VIP generated `WAKE#` signal pattern pulse widths are user-programmable using the following attributes:

1. `svt_pcie_tl_configuration::wake_high_pulse_width_in_ns`: This attribute indicates pulse width of `WAKE#` signal from a rising edge to a falling edge in terms of ns.
2. `svt_pcie_tl_configuration::wake_low_pulse_width_in_ns`: This attribute indicates pulse width of `WAKE#` signal from a falling edge to a rising edge in terms of ns.

Note
`svt_pcie_tl_configuration::wake_transition_delay_in_ns` is only applicable when VIP is acting as an Active device.

6.5.4 Decoding and Validating CPU State Transition Using EP VIP

EP VIP can be programmed to enable CPU state transitions' checking by setting `svt_pcie_tl_configuration::enable_cpu_state_transition_checking` to 1. When enabled, EP VIP can be programmed using `svt_pcie_tl_service::ADD_EXPECTED_CPU_STATES` service request ahead of time to verify CPU state transitions that a RC would carry out. An EP VIP will issue an error message on a mismatch between decoded CPU state transition as indicated by RC and programmed expected CPU state. If an EP VIP encounters any additional CPU state transition than programmed expected or any lesser CPU state transitions than programmed expected value; it issues an error message.

An EP VIP uses `svt_pcie_tl_configuration::wake_high_pulse_width_in_ns` and `svt_pcie_tl_configuration::wake_low_pulse_width_in_ns` attributes to decode `WAKE#` signal patterns to interpret CPU state transitions. These attribute values in VIP should be programmed to match DUT values. When `WAKE#` high and low pulse widths are generated by DUT ranges in a certain time zone, EP VIP is expected to be programmed with upper range of pulse width.

Note
`svt_pcie_tl_configuration::wake_high_pulse_width_in_ns` and `svt_pcie_tl_configuration::wake_low_pulse_width_in_ns` are used by VIP decode `Wake#` signaling pattern while acting as an Active device or Passive device.

6.5.5 Protocol Checks

- `cpu_idle_to_low_power_check`: Applicable when `enable_obff` is non-zero. The intention of this protocol check is to validate time required by software to take link to low power state (that is, not in L0) when RC CPU goes to IDLE state. If the time taken to transition link to low power state after CPU has transitioned to IDLE state is more than user programmable `svt_pcie_tl_configuration::cpu_idle_to_low_power_state_delay_in_ns` attribute value, then protocol check issues a violation.
- `non_idle_cpu_state_duration_check`: Applicable when `enable_obff` is non-zero. The intention of this protocol check is to validate that the amount of time CPU spends in either ACTIVE or OBFF or ACTIVE+OBFF states prior to transitioning to IDLE state is greater than user programmable `svt_pcie_tl_configuration::min_duration_for_non_idle_cpu_state_in_ns`. If the amount of time spent in ACTIVE or OBFF or ACTIVE+OBFF combined CPU state is lesser than this attribute, then the protocol check issues a violation.
- `cpu_obff_check`: Applicable when `enable_obff` is non-zero and VIP is programmed as RC. The protocol check is not applicable when CPU state is ACTIVE or IDLE. When CPU state is OBFF, protocol check issues a violation for any request generated by EP other than memory read or write. As per the base specifications, OBFF indication is simply a hint to EP, so EP is still permitted to generate other bus mastering requests when CPU state is OBFF however that is considered to be non-optimal from platform power consumption perspective. If EP is expected to generate requests other than memory reads or writes while in OBFF CPU state by design, this protocol check should be disabled.

Note
Protocol checks applicable in both active and passive modes of VIP.

6.5.6 Known Limitations

1. OBFF feature is supported by SVT VIP only.
2. OBFF feature is supported in UVM mode only.
3. OBFF feature is supported only when using unified interface.
4. VIP does not support `Variation A` mechanism using OBFF messages.

If user queues OBFF messages in low power states in `Variation A`, VIP will not discard them automatically.

6.6 Replay Timer

The replay timeout calculation as per the PCIe specification is as follows:

- For specification version prior to 4.0, the replay timeout value is based on a formula listed in the 3.1 specification.
- For specification version 4.0 and higher, the replay timeout value can be set to 1 of 2 values based on the extended sync bit value.

The specification provides an option to support the Gen3 replay timeout calculation when supporting Gen4 features. This option can only be used if speeds greater than 8G are not supported.

6.6.1 Configuration

The VIP will automatically select the replay timeout method based on the `pcie_spec_ver` variable and extended sync bit (when applicable).

Note
There is no change for the tests with `replay_timeout` settings in DL configuration.

Replay timeout settings for Gen3 and Gen4 are as follows:

- For Gen3, only `replay_timeout` variable is used.
- For Gen4, you can use the following variables:
  - `replay_timeout:` The `replay_timeout` variable sets timeout when the extended sync bit is 0. The default value of this variable is set to 31k.
  - `replay_extended_sync_timeout:` The `replay_extended_sync_timeout` variable provides control over the timeout value when the extended sync bit is 1. This variable is only applicable to specification version 4.0 or higher. The default value of this variable is set to 100k.

    Per the specification, the valid ranges are 24k–31k and 80k–100k (extended sync). Default is set to the largest suggested value such that no tuning is required.

The VIP provides checkers to verify the DUT's implementation of the replay timeout. Configuration of the checker variables is required only if the DUT's implementation deviates from the specification values. All configuration variables associated with the replay timeout checker have `attached_` prepended to the names. The checkers are configured as per the specification similar to the replay timeout. Based on the DUT's internal delays, you may configure the dependent variables.

Table 6-6 Replay Timer Variables

Variable	Description
`replay_timeout`	Specifies the timeout count value for the replay timer in symbols. • For `pcie_spec_ver < 4.0`, setting this variable to 0 enables automatic timeout updates if `#max_payload_size, link_width`, or speed change. • For `pcie_spec_ver >= 4.0`, this variable sets the simplified replay timeout of the VIP if the extended sync bit is not set. If randomized, the variable will resolve to a value within the range specified by the constraint `#valid_ranges`. Note: Utilized by both active and monitor component.
`replay_extended_sync_timeout`	Specifies the timeout count value for the replay timer in symbols when the extended sync bit is set. Only applicable for specification version 4.0 and using a simplified replay timer. If randomized, the variable will resolve to a value within the range specified by the constraint `#valid_ranges`. Note: Utilized by the active component only.

Table 6-6 Replay Timer Variables

Variable	Description
enable_gen3_replay_timeout_calc_in_gen4	Specifies how the replay timeout value in VIP is derived when supporting PCIe specification version 4.0 (Gen4). • If 16G speed is supported, the simplified retimer approach must be used. • If 16G speed is not supported at Gen4, the replay timeout value can optionally be derived per PCIe specification 3.1 (Gen3). - 0: Will enable a simplified replay timer. - 1: Will enable automatic update of timer per Gen3 specification. Note: Utilized by the active component only.
attached_replay_timeout	Specifies the length of the attached replay timeout in symbols. Used to check the DUT's replay timer. • For `pcie_spec_ver < 4.0`, setting this variable to 0 enables automatic updates if `#max_payload_size, link_width`, or speed change. • For `pcie_spec_ver >= 4.0`, this variable sets the simplified replay timeout of the DUT if extended sync bit is NOT set. * The value has to be within the range specified by the constraint `#valid_ranges`.
attached_replay_extended_sync_timeout	Specifies the length of the attached replay timer in symbols when extended sync bit is enabled. Used to check the DUT's replay timer. This value is used only for `pcie_spec_ver >= 4.0` and using a simplified replay timer. The value has to be within the range specified by the constraint `#valid_ranges`. Note: Utilized by the active component only.
enable_replay_timer_adjust_extended_sync_change	Specifies whether the replay timer is reset when extended sync bit is changed when replay timer is running. Per the specification, resetting the replay timer is optional. As this is enabled by default, replay timeouts will not occur. This variable is applicable to `pcie_spec_ver >= 4.0`. For `pcie_spec_ver < 4.0`, this variable is ignored. Note: Utilized by the active component only.

6.7 SRIS/SRNS

6.7.1 Introduction

The specification supports two kinds of clocking where the Tx and Rx Refclk rates differ. One clock allows at least 600 ppm difference with no SSC (Separate Reference Clocks With No SSC - SRNS) and the other clock allows 5600 ppm difference for separate Refclks utilizing independent SSC (Separate Reference Clocks with Independent SSC - SRIS) (SSC introduces a 5000 ppm difference and Tx/Rx crystal tolerance introduces another 600 ppm.

6.7.2 Basic Attributes

VIP support for SRIS/SRNS is disabled by default. You can enable this feature—that is to introduce ppm in serial bit clock on the transmit path using `svt_pcie_pl_configuration::ssc_mode`, `svt_pcie_pl_configuration::ssc_max_spread`, `svt_pcie_pl_configuration::ssc_modulation_rate` and `svt_pcie_pl_configuration::fixed_ppm_due_to_tx_rx_xo` attributes.

- `svt_pcie_pl_configuration::ssc_mode`: This attribute specifies the spread spectrum clocking (SSC) mode for serial bit clock on the transmit path. VIP allows two profiles to introduce spread spectrum—Down spread and Center spread.
- `svt_pcie_pl_configuration::ssc_max_spread`: This attribute specifies the SSC spread value in ppm (parts per million) for the rates that PCIe supports.
- `svt_pcie_pl_configuration::ssc_modulation_rate`: This attribute specifies the modulation rate for SSC clock. By default, it is set to 33 kHz.
- `svt_pcie_pl_configuration::fixed_ppm_due_to_tx_rx_xo`: This attribute specifies the fixed ppm value in tx serial bit clk which comes due to different tx/rx crystal oscillator.

  All these attributes are only applicable for VIP running with serial interface. For more details about the usage, see "svt_pcie_pl_configuration" in the HTML class reference documentation.

Note
At higher speeds, you need to be aware of the inherent maximum of 1fs precision in SystemVerilog. For example, at 16G, trying to run with a 600ppm difference would result in the default bit period 0.0675ns being adjusted by 0.0000375ns or 37.5fs. To generate a clock with this period requires an attosecond resolution, which SystemVerilog does not support. It is therefore recommend to round the ppm either up or down to the nearest 1fs so that there will be no rounding errors in the clock period and adjust SKP intervals accordingly.

6.7.3 Enabling the SRNS Mode

Set the following configurations:

- `svt_pcie_pl_configuration::ssc_mode` for all rates should be configured with `svt_pcie_pl_configuration::SSC_MODE_DISABLED` (default).
- `svt_pcie_pl_configuration::fixed_ppm_due_to_tx_rx_xo` has to be configured with a ppm value within the range of -300 to +300ppm.

Example code where +300ppm constant ppm value in introduced in transmit serial bit clock:

```
cfg.rc_cfg.pcie_cfg.pl_cfg.ssc_mode[0] = svt_pcie_pl_configuration::SSC_MODE_DISABLED;
cfg.rc_cfg.pcie_cfg.pl_cfg.ssc_mode[1] = svt_pcie_pl_configuration::SSC_MODE_DISABLED;
cfg.rc_cfg.pcie_cfg.pl_cfg.ssc_mode[2] = svt_pcie_pl_configuration::SSC_MODE_DISABLED;
cfg.rc_cfg.pcie_cfg.pl_cfg.ssc_mode[3] = svt_pcie_pl_configuration::SSC_MODE_DISABLED;
cfg.rc_cfg.pcie_cfg.pl_cfg.fixed_ppm_due_to_tx_rx_xo = 300;
```

For SRNS, there is no change in SKP OS scheduling interval. The SKP Ordered Set must be scheduled for transmission at an interval between 1180 and 1538 symbol times if the device is running in Gen1 or Gen2 rate. For higher rates, SKP Ordered Set must be scheduled for transmission at an interval between 370 to 375 blocks.

6.7.4 Enabling the SRIS Mode

Set the following configurations:

- VIP provides mechanism to introduce SSC in a particular line rate.
- `svt_pcie_pl_configuration::ssc_mode` should be configured with `svt_pcie_pl_configuration::SSC_MODE_DOWN_SPREAD` or `svt_pcie_pl_configuration::SSC_MODE_CENTER_SPREAD` for the desired rate.

- `svt_pcie_pl_configuration::ssc_max_spread` should be configured with the SSC spread value in terms of ppm for the desired rate. Specification allows a maximum of 5000 ppm difference.
- `svt_pcie_pl_configuration::ssc_modulation_rate` can be used to specify the modulation rate of SSC clock. Allowed range is 30 kHz to 33 kHz.
- `svt_pcie_pl_configuration::fixed_ppm_due_to_tx_rx_xo` has to be configured with a ppm value within the range of -300 to +300 ppm.

  When SRIS mode is enabled by the device, specification constraints the SKP OS scheduling interval with respect to rate. The SKP Ordered Set must be scheduled for transmission at an interval of less than 154 Symbol Times for Gen1 and Gen2 rates. For higher rates, a SKP Ordered Set must be scheduled for transmission at an interval less than 38 Blocks, when the LTSSM is not in the Loopback state or is a Loopback Slave that has not started looping back the incoming bit stream.

Above configurations can be done by using the following `svt_pcie_pl_configuration` attributes:

- `min_tx_skp_interval_in_symbol_times`
- `max_tx_skp_interval_in_symbol_times`
- `min_tx_skp_interval_in_blocks`
- `max_tx_skp_interval_in_blocks`

In L0 LTSSM state, specification also allows SKP OS generation at the rate used in SRNS mode even though port is running in SRIS.

This is applicable when the following conditions are true.

- Bit is set for the appropriate data rate in the Lower SKP OS `Reception Supported Speeds Vector` field of the Link Capabilities 2 register of the receiver.
- Bit is set for the appropriate data rate in the Lower SKP OS `Generation Supported Speeds Vector` field of the Link Capabilities 2 register of the transmitter.

If both conditions are true, then the above mentioned SKP interval configurations can be reconfigured during L0 LTSSM state as per SKP generation at the rate used in SRNS.

Following is the example code where VIP is configured to introduce down spread SSC in transmit bit clock for all supported rates.

```
cfg.rc_cfg.pcie_cfg.pl_cfg.ssc_mode[0] =
svt_pcie_pl_configuration::SSC_MODE_DOWN_SPREAD;
cfg.rc_cfg.pcie_cfg.pl_cfg.ssc_mode[1] =
svt_pcie_pl_configuration::SSC_MODE_DOWN_SPREAD;
cfg.rc_cfg.pcie_cfg.pl_cfg.ssc_mode[2] =
svt_pcie_pl_configuration::SSC_MODE_DOWN_SPREAD;
cfg.rc_cfg.pcie_cfg.pl_cfg.ssc_mode[3] =
svt_pcie_pl_configuration::SSC_MODE_DOWN_SPREAD;
cfg.rc_cfg.pcie_cfg.pl_cfg.fixed_ppm_due_to_tx_rx_xo = 300;

cfg.rc_cfg.pcie_cfg.pl_cfg.min_tx_skp_interval_in_symbol_times = 153;
cfg.rc_cfg.pcie_cfg.pl_cfg.max_tx_skp_interval_in_symbol_times = 153;
cfg.rc_cfg.pcie_cfg.pl_cfg.min_tx_skp_interval_in_blocks = 37;
cfg.rc_cfg.pcie_cfg.pl_cfg.max_tx_skp_interval_in_blocks = 37;
```

If lower SKP OS generation is supported and allowed during SRIS mode, above SKP interval configurations must be reconfigured with the following values in L0 LTSSM state.

```
cfg.rc_cfg.pcie_cfg.pl_cfg.min_tx_skp_interval_in_symbol_times = 1180;
cfg.rc_cfg.pcie_cfg.pl_cfg.max_tx_skp_interval_in_symbol_times = 1538;
cfg.rc_cfg.pcie_cfg.pl_cfg.min_tx_skp_interval_in_blocks = 370;
```

```
cfg.rc_cfg.pcie_cfg.pl_cfg.max_tx_skp_interval_in_blocks = 375;
```

6.7.5 Configuring the VIP Receiver for SRIS/SRNS Mode

The VIP receiver supports both SRIS and SRNS mode. When VIP receiver is running with SRIS mode and expected to receive SKP OS in the interval specified as per SRIS specification, then the following `svt_pcie_pl_configuration` attributes must be configured with proper values as shown below. Otherwise, VIP will trigger `phy_too_many_skp_sets` check.

```
cfg.rc_cfg.pcie_cfg.pl_cfg.min_rx_skp_interval_in_symbol_times = 1;
cfg.rc_cfg.pcie_cfg.pl_cfg.max_rx_skp_interval_in_symbol_times = 153;
cfg.rc_cfg.pcie_cfg.pl_cfg.min_rx_skp_interval_in_blocks = 1;
cfg.rc_cfg.pcie_cfg.pl_cfg.max_rx_skp_interval_in_blocks = 37;
```

7 Gen5 Features

This chapter describes the Gen5 features available with the Synopsys PCIe Verification IP.

This chapter discusses the following topics:

7.1 Version Support

The Synopsys PCIe SVT VIP supports the following specification versions:

Protocol Specification	Version
PCI Express Base Spec	PCIe Gen 5.0 Revision 1.0
PIPE Interface Spec	• PIPE 5.1.1 • PIPE 4.4 / 4.4.1

7.2 Supported Interfaces

Currently, the Gen5 support is available only for serial and PIPE interface using the PCIe SVT Unified model.

Note

- Gen5 support for legacy instantiation models will not be supported.
- For PIPE Interface, this feature is currently available only when PCIe SVT VIP is used in active mode only.
  - PIPE 5.1.1 LPC, PIPE5 Features.
  - For PIPE 5.1.1, SerDes architecture is available as EA. For more details, contact Synopsys Support.

7.3 VIP License Requirements

To enable Gen5 features, use the Gen5 license key `VIP-PCIE-G5-SVT` or `VIP-LIBRARY2019-SVT`. The Gen5 license is standalone and inclusive of all other PCIe SVT VIP features. For more details, see Licensing Information.

7.4 Supported Features

The following features are implemented for Gen5:

- To advertise support and run at 32 GT/s
- Enhanced Link Control Behavior.
  - Equalization controls at 32GT/s.
  - Modified TS1/TS2 OS
- Precoding
- Equalization via Loopback
- Multi lane BERT Testing

7.5 Enabling Gen5 Support

Setting up the VIP to run at Gen5 is similar to the steps followed with other versions. Following are the additional steps required:

1. Add the `` `define SVT_PCIE_ENABLE_GEN5 `` define to your environment to compile Gen5 features.
2. Configure the VIP to support the PCIe 5 specification in the device configuration.

   ```
   vip_cfg.pcie_spec_ver = svt_pcie_device_configuration::PCIE_SPEC_VER_5_0;
   ```
3. In case of PIPE interface configuration, the VIP with PIPE specification setting in the device configuration is as follows:
   - PIPE Spec version 5.1.1:

     ```
     vip_cfg.pipe_spec_ver = svt_pcie_device_configuration::PIPE_SPEC_VER_5_1;
     ```
   - PIPE Spec version 4.4

     ```
     vip_cfg.pipe_spec_ver = svt_pcie_device_configuration::PIPE_SPEC_VER_4_4;
     ```
4. Set the supported speeds to include 32GT/s in the PL configuration.

   ```
   vip_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_32_0G |
   `SVT_PCIE_SPEED_16_0G | `SVT_PCIE_SPEED_8_0G | `SVT_PCIE_SPEED_5_0G | `SVT_
   PCIE_SPEED_2_5G, `SVT_PCIE_SPEED_32_0G, `SVT_PCIE_SPEED_32_0G)
   ```

7.6 Gen5 Configuration Settings

7.6.1 32G Equalization

The new 32G Equalization items mirror the existing ones that are used at 8G and 16G. For more details, see HTML class reference documentation:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_pl_configuration.html#group_gen5_params

The VIP attributes specific for Gen5 are post-fixed with `_32g`.

In the HTML class reference documentation, you can search for `_32g` in the "search" tab to get the list of attributes.

Table 7-1 32G Equalization Configuration Settings

Parameter	Description
`fs_value_32g`	The FS value that the VIP advertises during phase 1 EQ at 32G.
`fs_value_32g`	The FS value that the VIP advertises during phase 1 EQ at 32G.
`expected_lf_value_32g`	The LF value that the VIP expects the DUT to advertise at 32G during phase 1 EQ.
`expected_fs_value_32g`	The LF value that the VIP expects the DUT to advertise at 32G during phase 1 EQ.
`preset_to_coefficients_mapping_table_32g`	This sets up what coefficients the VIP will use when asked for a specific preset during phase2/3 equalization.
`preset_to_coefficients_mapping_entry_valid_32g`	Indicates that a given preset is considered valid by the VIP during equalization at 32G. If a preset entry is invalid, the VIP will automatically reject any received request for that preset during phase2/phase3 EQ.
`expected_preset_to_coefficients_mapping_table_32g`	The table which sets the DUT's preset to coefficient mapping at 32G.
`expected_preset_to_coefficients_mapping_entry_enable_32g`	Sets which presets are valid and supported by the DUT at 32G.
`upstream_receiver_preset_hint_32g`	Depending on whether the VIP is configured at a RC or EP, this sets the preset hint to either be advertised or expected in received EQTS2 that are received before Phase 0 EQ at 32G.
`downstream_receiver_preset_hint_32g`	Downstream hint transmitted in EQTS1s during phase 1 EQE at 32G.
`downstream_receiver_preset`	Sets the default downstream preset when entering EQ at 32G.

Table 7-1 32G Equalization Configuration Settings

Parameter	Description
upstream_receiver_preset	Sets the preset the upstream port should be using when entering EQ at 32G.
upstream_advertise_32g_support_before_16g_eq	Controls whether or not the upstream port advertised 32G before 16G equalization has completed. Downstream must follow the rules outlined in the specification.
enable_upstream_tx_16g_eqts2_with_preset_in_rcvr_cfg	Optionally allow the upstream port to communicate presets to the downstream port before EQ begins.

7.6.2 Configuring Coefficient and Preset Requests

Coefficient requests and preset requests are handled at 32G the same way they are handled in 8G and 16G—by using the `QUEUE_EQ_TX_REQUEST_PRESET_COEFF` service call.

Service Call Sequence

```
task svt_pcie_pl_service_queue_eq_tx_request_preset_coeff::body();
  begin
    svt_pcie_pl_service queue_eq_tx_request_preset_coeff_req;
    `svt_verbose("body", "Physical Layer, Queue Equalization Preset/Coefficient Request started");
    `svt_xvm_do_with(queue_eq_tx_request_preset_coeff_req,{service_type == svt_pcie_pl_service::QUEUE_EQ_TX_REQUEST_PRESET_COEFF;
                                              eq_lane_num == local::eq_lane_num;
                                              preset_valid == local::preset_valid;
                                              preset_value == local::preset_value;
                                              precursor_coeff == local::precursor_coeff;
                                              cursor_coeff == local::cursor_coeff;
                                              postcursor_coeff == local::postcursor_coeff;
                                              expect_reject == local::expect_reject;})
    `svt_verbose("body", "Physical Layer, Queue Equalization Preset/Coefficient Request ended");
  end
endtask : body
```

Example 7-1 Sequence Usage

```
//Preset requests
for (int lane = 0; lane < vip_cfg.pcie_cfg.pl_cfg.get_expected_link_width_value(); lane++)
begin
  `svt_xvm_do_on_with(preset_coeff_exception_sequence, dut_seqr.pcie_virt_seqr.pl_seqr, {
  eq_lane_num == lane;
  preset_valid == 1'b1;
  preset_value == 4'h2;
  expect_reject == 0;
  })
end
```

7.6.3 Enhanced Link Behavior Control - TS1/2 Symbol 5 (Optional)

Control	Description

Full Equalization required (00b)	`cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(LINK_EQ_MODE_FULL_EQUALIZATION_REQUIRED);`
	Note: `set_link_eq_attribute_values` (`LINK_EQ_MODE_FULL_EQUALIZATION_REQUIRED`) Internally sets Pl configuration attributes:
	`equalization_bypass_to_highest_rate_support_disable = 1;`
	`no_equalization_needed_support_disable = 1;`

Equalization bypass to highest rate support(01b)	`cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(LINK_EQ_MODE_EQ_BYPASS_TO_HIGHEST_RATE);`
	Note: `set_link_eq_attribute_values` (`LINK_EQ_MODE_EQ_BYPASS_TO_HIGHEST_RATE`) Internally sets Pl configuration attributes:
	`equalization_bypass_to_highest_rate_support_disable = 0;`
	`no_equalization_needed_support_disable = 1;`

No Equalization needed(10b)	`cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_eq_attribute_values(LINK_EQ_MODE_NO_EQUALIZATION_NEEDED);`
	Note: `set_link_eq_attribute_values` (`LINK_EQ_MODE_NO_EQUALIZATION_NEEDED`) Internally sets Pl configuration attributes:
	`equalization_bypass_to_highest_rate_support_disable = 0;`
	`no_equalization_needed_support_disable = 0;`

Modified TS1/TS2 Ordered Sets supported(11b)	`cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_modified_ts_mode_values ( 1 /*advertise_support_for_modified_ts */);`

7.6.4 Skew Support for 32G

Parameter	Description
`min_tx_lane_skew_32g`	When the lane skew mode is set to random, specifies the minimum lane skew between transmit lanes in symbol times at 32Gt/s.
`max_tx_lane_skew_32g`	When the lane skew mode is set to random, specifies the maximum lane skew between transmit lanes in symbol times at 32Gt/s.

7.6.5 nFTS at 32G

Parameter	Description
`num_rx_fts_required_32g`	Specifies the minimum number of FTS ordered sets required for a receiver to infer lock when exiting from electrical idle at 32GT/s.

7.6.6 Retimer Latency on RX Data Path

Parameter	Description
`rx_retimer_latency_32g_ns`	Programmable latency used to mimic the presence of a Retimer on the rx data path

7.6.7 Precoding

Parameter	Description
`transmitter_precode_request_32g`	When set to a 'b1, the VIP will request that the far-end transmitter use precoding

7.6.8 32G Loopback Controls

Service Call: Loopback is initiated using a service call `svt_pcie_pl_service_initiate_loopback`.

To control Gen5-specific features, use the following fields:

Parameter	Description
`enable_loopback_w_equalization`	Enables VIP to perform equalization to the lane under test before entering loopback.
`loopback_lane_under_test`	Defines the lane under test when loopback with equalization is being performed.
`set_ts_transmit_modified_compliance_in_loopback`	When set, the loopback master will set the `transmit_modified_compliance_in_loopback` bit in outgoing TS when requesting loopback.

7.7 EIEOSQ Controls

This feature is used for block alignment state machine testing.

Parameter	Description
`num_tx_eieos_in_eieosq_32g`	Number of EIEOS ordered sets in an EIEOSQ a transmitter should send.
`num_rx_eieos_in_eieosq_32g`	Number of EIEOS ordered sets a receiver should expect in an EIEOSQ.

7.8 Gen5 Support for PIPE Interface

The PCIe SVT VIP supports Gen5 speed over PIPE when it is compatible with *PCIe Base Specification Revision 5.0* and *PIPE Specifications Revision 4.4/4.4.1*. This feature is currently available only when PCIe SVT VIP is used in active mode and only when using the Unified instantiation module.

7.8.1 Enabling Gen5 Support

To enable Gen5 support, set the following configurations:

- Set the Gen5 license key. For more details, see Licensing Information.

- Define the `SVT_PCIE_ENABLE_GEN5` macro in the makefile.
- Set the PCIe specification version `PCIE_SPEC_VER_5_0`.
- Set the PIPE specification version `PIPE_SPEC_VER_4_4`.
- Set the `svt_pcie_pl_configuration::enable_gen5_using_pipe_4_4` attribute to *1*.

Note
No changes in callbacks and analysis ports.

7.8.2 Protocol Checks and Exceptions

All 128/130b checks at 8G and 16G will be applicable for 32G. All 8G and 16G equalization checks will be applicable for 32G.

For the detailed list of checks, see HTML class reference documentation.

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/protocolChecks.html

Example 7-2 Searching "PCIE v5.0" in the Browser

SELECT ALL	SELECT ALL			
Group	**Sub Group**	**Protocol Check Instance name**	**Reference**	**Description**
ACTIVE_PL	TS_OS	phy_received_modified_ts_when_not_negotiated	PCIE v5.0: 4.2.6.3.2 Configuration.Linkwidth.Accept	The variable use_modified_TS1_TS2_Ordered_Set must be set to 1b if all of the following conditions are true:LinkUp = 0b, The component had transmitted Modified TS1/TS2 Ordered Sets supported value 11b in the Enhanced Link Behavior Control field in Symbol 5 of TS1 and TS2 Ordered Sets in Polling and Configuration states since entering the Polling State. The received eight consecutive TS2 Ordered Sets on all Lanes of the currently configured Link that caused the transition from Polling.Configuration to Configuration state had the Modified TS1/TS2 Ordered Sets supported value 11b in the Enhanced Link Behavior Control field in Symbol 5 and 32.0 GT/s data rate is supported bit is set to 1b in the receieved eight consecutive TS2 Ordered Sets.
ACTIVE_PL	CONFIGURATION_LTSSM	phy_supported_linkwidth_exceeds_configured_width	PCIE v5.0: 4.2.6.3 Configuration	Supported linkwidth exceeds configured linkwidth
ACTIVE_PL	FRAMING	phy_stp_not_aligned_to_lane0_after_idl_on_rx	PCIE v5.0: 4.2.1.2 Framing and Application of Symbols to Lanes	Physical Layer received STP token not aligned to Lane0 after receiving IDL on Rx path
ACTIVE_PL	FRAMING	phy_received_multiple_stp_at_same_symbol_time	PCIE v5.0: 4.2.1.2 Framing and Application of Symbols to Lanes	Physical Layer detected multiple STP tokens at the same symbol time on Rx path
ACTIVE_PL	VIP_INTERNAL	phy_vip_internal_tx_fifo_overflow	PCIE v5.0: 4 Physical Layer Logical Block: VIP internal message.	VIP specific message

7.8.3 Gen5 Example

For details on Gen5 example, see Testbench Structure and for usage details, contact Synopsys Support

7.9 Gen5 Symbol Logging

When the VIP is enabled to run at Gen5 and symbol logging is enabled, by default the Symbol Logger will display the following set of Gen5-specific data points for Gen5 debug analysis.

7.9.1 Precoding Display

- A marker when speed changes to 32.0 GT/s indicating the transmitter Precoding status of DUT(RX) and VIP(TX).

Example

The following example illustrates the scenario where VIP's link partner (on LHS) requested VIP to turn ON precoding but VIP (on RHS) did not make the same request.

Transmitter Precode Request, Symbol 7, bit 0 of 128b/130b EQ_TS2 OS for operating at 32.0 GT/s or higher Data Rate.

Following message displays Precoding ON/OFF marker.

```
---------------------------------------------------------------|--------------------------------------
   33803: spd_1                           *2d  *2d  *2d  *2d  |   00   00   00   00  -> Recov.Rcvrcfg   >RX:EQ_TS2_OS on Lane0
   33803: spd_1                            00   00   00   00  |   00   00   00   00  -> Recov.Rcvrcfg
   33804: spd_1                            00   01   02   03  |   98   98   98   98  -> Recov.Rcvrcfg
   33804: spd_1                            ff   ff   ff   ff  |   45   45   45   45  -> Recov.Rcvrcfg
   33805: spd_1                            be   be   be   be  |   45   45   45   45  -> Recov.Rcvrcfg
   33805: spd_1                            00   00   00   00  |   45   45   45   45  -> Recov.Rcvrcfg
   33806: spd_1                            00   00   00   00  |   45   45   45   45  -> Recov.Rcvrcfg
   33806: spd_1                            81   81   81   81  |   45   45   45   45  -> Recov.Rcvrcfg
   33807: spd_1                            45   45   45   45  |   45   45   45   45  -> Recov.Rcvrcfg
   33807: spd_1                            45   45   45   45  |   20   45   45   45  -> Recov.Rcvrcfg
   33808: spd_1                            45   45   45   45  |   08   45   45   f7  -> Recov.Rcvrcfg   >TX:EQ_TS2_OS on Lane0-3
   33808: spd_1                            45   45   45   45  |  *2d  *2d  *2d  *2d  -> Recov.Rcvrcfg
   33809: spd_1                            45   45   45   45  |   00   00   00   00  -> Recov.Rcvrcfg
   33809: spd_1                            45   45   45   45  |   00   01   02   03  -> Recov.Rcvrcfg
   33810: spd_1                            45   45   45   45  |   ff   ff   ff   ff  -> Recov.Rcvrcfg
   33810: spd_1                            08   45   45   f7  |   fe   fe   fe   fe  -> Recov.Rcvrcfg   >RX:EQ_TS2_OS on Lane1-3
.
.
.
spd_1                         --  Detected change in data rate to 32 GT/s. TX Precoding: On, RX Precoding: Off. See file header for special encodings.
```

7.9.2 ELBC Display

- Mutually supported ELBC when entering `Config.Idle`.

Example

The following example illustrates the scenario where VIP's link partner (on LHS) advertised "Full Equalization required" and VIP (on RHS) advertised "No Equalization needed" and the mutually supported EQ option is resolved to Full EQ Required.

Full Equalization in Symbol 5, bit [7:6].

No Equalization Symbol 5, bit [7:6].

```
21504: spd_1                    COM  COM  COM  COM  |   45   45   45   45  -> Cfg.Complete
21508: spd_1                     00   00   00   00  |   45   45   45   45  -> Cfg.Complete
21512: spd_1                     00   01   02   03  |   45   45   45   45  -> Cfg.Complete
21516: spd_1                     ff   ff   ff   ff  |   45   45   45   45  -> Cfg.Complete
21520: spd_1                     7e   7e   7e   7e  |   45   45   45   45  -> Cfg.Complete
21524: spd_1                     00   00   00   00  |   45   45   45   45  -> Cfg.Complete   >TX:TS2_OS on Lane0-3
21528: spd_1                     45   45   45   45  |  COM  COM  COM  COM  -> Cfg.Complete
21532: spd_1                     45   45   45   45  |   00   00   00   00  -> Cfg.Complete
21536: spd_1                     45   45   45   45  |   00   01   02   03  -> Cfg.Complete
21540: spd_1                     45   45   45   45  |   ff   ff   ff   ff  -> Cfg.Complete
21544: spd_1                     45   45   45   45  |   7e   7e   7e   7e  -> Cfg.Complete
21548: spd_1                     45   45   45   45  |   80   80   80   80  -> Cfg.Complete
21552: spd_1                     45   45   45   45  |   45   45   45   45  -> Cfg.Complete
21556: spd_1                     45   45   45   45  |   45   45   45   45  -> Cfg.Complete
21560: spd_1                     45   45   45   45  |   45   45   45   45  -->>  Cfg.Idle: Mutually Supported EQ Option: Full EQ Required,
Transition Reason: LTSSM_TRANSITION_STANDARD
```

7.9.3 EQ Via Loopback Display

Example

The following example illustrates the scenario where VIP is Loopback master and has chosen lane 3 to be 'Lane Under Test' and slave is not required to send Modified Compliance on lanes that are not under test.

```
19511: spd_0                       4a   4a   4a   4a  |  4a   4a   4a   4a  -->> Loopback.Entry: lane_under_test = 3,
transmit_modified_compliance_pattern_in_loopback = 0,  Transition Reason: LTSSM_TRANSITION_DIRECTED
19515: spd_0                       4a   4a   4a   4a  |  4a   4a   4a   4a  -> Loopback.Entry
19519: spd_0                      COM  COM  COM  COM  |  4a   4a   4a   4a  -> Loopback.Entry
19523: spd_0                      PAD  PAD  PAD  PAD  |  4a   4a   4a   4a  -> Loopback.Entry
19527: spd_0                      PAD  PAD  PAD  PAD  |  4a   4a   4a   4a  -> Loopback.Entry
19531: spd_0                       ff   ff   ff   ff  |  4a   4a   4a   4a  -> Loopback.Entry
19535: spd_0                       3e   3e   3e   3e  |  4a   4a   4a   4a  -> Loopback.Entry
19539: spd_0                       80   80   80   80  |  4a   4a   4a   4a  -> Loopback.Entry
19543: spd_0                       4a   4a   4a   4a  | COM  COM  COM  COM  -> Loopback.Entry
19547: spd_0                       4a   4a   4a   4a  | PAD  PAD  PAD  PAD  -> Loopback.Entry
19551: spd_0                       4a   4a   4a   4a  | PAD  PAD  PAD  PAD  -> Loopback.Entry
19555: spd_0                       4a   4a   4a   4a  |  ff   ff   ff   ff  -> Loopback.Entry
19559: spd_0                       4a   4a   4a   4a  |  3e   3e   3e   3e  -> Loopback.Entry
19563: spd_0                       4a   4a   4a   4a  |  00   00   00   44  -> Loopback.Entry
19567: spd_0                       4a   4a   4a   4a  |  4a   4a   4a   4a  -> Loopback.Entry
19571: spd_0                       4a   4a   4a   4a  |  4a   4a   4a   4a  -> Loopback.Entry
19575: spd_0                       4a   4a   4a   4a  |  4a   4a   4a   4a  -> Loopback.Entry
19579: spd_0                       4a   4a   4a   4a  |  4a   4a   4a   4a  -> Loopback.Entry
19583: spd_0                      COM  COM  COM  COM  |  4a   4a   4a   4a  -> Loopback.Entry
19587: spd_0                      PAD  PAD  PAD  PAD  |  4a   4a   4a   4a  -> Loopback.Entry
19591: spd_0                      PAD  PAD  PAD  PAD  |  4a   4a   4a   4a  -> Loopback.Entry
19595: spd_0                       ff   ff   ff   ff  |  4a   4a   4a   4a  -> Loopback.Entry
19599: spd_0                       3e   3e   3e   3e  |  4a   4a   4a   4a  -> Loopback.Entry
19603: spd_0                       80   80   80   80  |  4a   4a   4a   4a  -> Loopback.Entry
19607: spd_0                       4a   4a   4a   4a  | COM  COM  COM  COM  -> Loopback.Entry
19611: spd_0                       4a   4a   4a   4a  | PAD  PAD  PAD  PAD  -> Loopback.Entry
19615: spd_0                       4a   4a   4a   4a  | PAD  PAD  PAD  PAD  -> Loopback.Entry
19619: spd_0                       4a   4a   4a   4a  |  ff   ff   ff   ff  -> Loopback.Entry
19623: spd_0                       4a   4a   4a   4a  |  3e   3e   3e   3e  -> Loopback.Entry
19627: spd_0                       4a   4a   4a   4a  |  00   00   00   44  -> Loopback.Entry
19631: spd_0                       4a   4a   4a   4a  |  4a   4a   4a   4a  -> Loopback.Entry
```

Lane#3 is under test, bit 5 is 0

7.9.4 use_modified_TS1_TS2_Ordered_Set Variable Display

- Point at which `use_modified_TS1_TS2_Ordered_Set` variable gets set. Modified TS will be sent and received beyond this point.

Example

The following example illustrates the scenario where both devices support Modified TS.

Bit[7:6] is 2'b11, both support modified TS. Following message displays the marker where use_modified_TS1_TS2 _Ordered_Set variable gets set.

```
17918: spd_1      4a   4a   4a   4a  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17922: spd_1     COM  COM  COM  COM  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17926: spd_1      00   00   00   00  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17930: spd_1     PAD  PAD  PAD  PAD  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17934: spd_1      ff   ff   ff   ff  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17938: spd_1      3e   3e   3e   3e  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17942: spd_1      c0   c0   c0   c0  | COM  COM  COM  COM  -> Cfg.Linkwidth.Accept
17946: spd_1      4a   4a   4a   4a  |  00   00   00   00  -> Cfg.Linkwidth.Accept
17950: spd_1      4a   4a   4a   4a  | PAD  PAD  PAD  PAD  -> Cfg.Linkwidth.Accept
17954: spd_1      4a   4a   4a   4a  |  ff   ff   ff   ff  -> Cfg.Linkwidth.Accept
17958: spd_1      4a   4a   4a   4a  |  3e   3e   3e   3e  -> Cfg.Linkwidth.Accept
17962: spd_1      4a   4a   4a   4a  |  c0   c0   c0   c0  -> Cfg.Linkwidth.Accept
17966: spd_1      4a   4a   4a   4a  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17970: spd_1      4a   4a   4a   4a  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17974: spd_1      4a   4a   4a   4a  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17978: spd_1      4a   4a   4a   4a  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17982: spd_1      4a   4a   4a   4a  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
17986: spd_1     COM  COM  COM  COM  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
  .
  .
  .
18162: spd_1      00   00   00   00  |  4a   4a   4a   4a  -> Cfg.Linkwidth.Accept
18166: spd_1      00   00   00   00  |  4a   4a   4a   4a  -->>  Cfg.Lanenum.Wait: use_modified_ts1_ts2_ordered_set = 1,
Transition Reason: LTSSM_TRANSITION_STANDARD
18170: spd_1      00   00   00   00  |  4a   4a   4a   4a  -> Cfg.Lanenum.Wait
```

8 PIPE Features

This chapter describes the PIPE feature support available with Synopsys PCIe Verification IP.

This chapter discusses the following topics:

- "PIPE Support" on page 121
- "RxStandby/RxStandbyStatus Handshake Support" on page 125
- "Nominal Empty Mode for EFIFO" on page 126
- "PIPE Signals Controllability in VIP" on page 127

8.1 PIPE Support

Configure the attribute `svt_pcie_device_configuration::pipe_spec_ver` to enable the required PIPE specification features as follows:

- `PIPE_SPEC_VER_2` – Enables PIPE version 2.0
- `PIPE_SPEC_VER_4` – Enables PIPE version 4.0
- `PIPE_SPEC_VER_4.2` – Enables PIPE version 4.2
- `PIPE_SPEC_VER_4.3` – Enables PIPE version 4.3
- `PIPE_SPEC_VER_4.4` – Enables PIPE version 4.4
- `PIPE_SPEC_VER_5_1` – Enables PIPE version 5.1

8.1.1 PIPE Version 2.0

Table 8-1 lists the PIPE specification version 2.0 supported features and signals.

Table 8-1 PIPE Version 2.0

Feature Name	PIPE Signal (S) Used	VIP Support
PIPE Reset	Reset#	Yes
Data transmission and reception (8b/10b at 2.5 GT/s and 5 GT/s data rates)	TxData, TxDataK, RxData, RxDataK	Yes

Table 8-1 PIPE Version 2.0

Feature Name	PIPE Signal (S) Used	VIP Support
Power Management (P0, P0s, P1 and P2)	PowerDown, PhyStatus	Yes
PCLK as PHY output setup only	PCLK	Yes
Changing Signaling Rate (2.5 GT/s and 5 GT/s)	Rate, PhyStatus	Yes
Fixed data path implementations (8-bit and 16-bit PIPE width)	NA	Yes
Transmitter Margining	TxMargin	Yes
Transmitter Voltage Swing Level	TxSwing	Yes
Selectable De-emphasis	TxDeemph	Yes
Receiver Detection	TxDetectRx/Loopback, PhyStatus	Yes
Transmitting/Detecting a beacon	PowerDown, TxElecIdle, RxElecIdle	Yes
Clock tolerance compensation	RxStatus	Yes
Error detection	RxStatus	Yes
Loopback	TxDetectRx/Loopback	Yes
Polarity inversion	RxPolarity	Yes
Setting negative disparity	TxCompliance	Yes
Electrical Idle sequence	TxElecIdle, RxElecIdle, RxValid	Yes
Lane turn off signaling	TxElecIdle, TxCompliance	Yes

8.1.2 PIPE Version 4.0

Table 8-2 lists the PIPE specification version 4.0 additional supported features and signals.

Table 8-2 PIPE Version 4.0

Feature Name	PIPE Signal (S) Used	VIP Support
PCLK as PHY output setup only	PCLK, Max PCLK	Yes
Operating Mode	PHY mode	No
8b/10b Encoding/Decoding	EncodeDecodeBypass	No
Data transmission and reception (128/130b at 8 GT/s data rates)	TxData, RxData, BlockAlignControl, TxStartBlock, RxStartBlock, TxDataValid, RxDataValid, TxSyncHeader, RxSyncHeader	Yes
Power Management (L1 substates PHY specific power states P4-P7)	PowerDown, PhyStatus	Yes

Table 8-2 PIPE Version 4.0

Feature Name	PIPE Signal (S) Used	VIP Support
Changing Signaling Rate (2.5 GT/s, 5 GT/s and 8 GT/s)	Rate, PhyStatus, PCLK Rate	Yes
Fixed/Variable data path implementations (8-bit, 16-bit and 32-bit PIPE width)	Width, DataBusWidth, PhyStatus	Yes
Link Equalization Evaluation	FS, LF, TxDeemph, RxPresetHint, RxEqEval, LinkEvaluationFeedbackFigureMerit, LinkEvaluationFeedbackDirectionChange, InvalidRequest	Yes
Receiver Standby	RxStandby, RxStandbyStatus	Yes
Data Throttling	TxDataValid, RxDataValid	Yes
Dynamic Preset Coefficient Updates (Dynamic Equalization)	LocalFS, LocalLF, GetLocalPresetCoeffcients, LocalPresetIndex, LocalTxCoefficientsValid, LocalTxPresetCoefficients	Yes

8.1.3 PIPE Version 4.2

Table 8-3 lists the PIPE specification version 4.2 additional supported features and signals.

Table 8-3 PIPE Version 4.2

Feature Name	PIPE Signal (S) Used	VIP Support
PCLK as PHY output/input setup	PCLK, Max PCLK	Yes
Data transmission and reception (128/130b at 8 GT/s and 16 GT/s data rates)	TxData, RxData, BlockAlignControl, TxStartBlock, RxStartBlock, TxDataValid, RxDataValid, TxSyncHeader, RxSyncHeader	Yes
Changing Signaling Rate (2.5 GT/s, 5 GT/s, 8 GT/s and 16 GT/s)	Rate, PhyStatus, PCLK Rate, PclkChangeOk, PclkChangeAck	Yes
Link Equalization Evaluation	FS, LF, TxDeemph, RxPresetHint, RxEqEval, LinkEvaluationFeedbackFigureMerit, LinkEvaluationFeedbackDirectionChange, InvalidRequest, RxEqInProgress	Yes

8.1.4 PIPE Version 4.3

Table 8-4 lists the PIPE specification version 4.3 additional supported features and signals.

Table 8-4 PIPE Version 4.3

Feature Name	PIPE Signal (S) Used	VIP Support
Power Management (L1 substates PHY specific power states P4-P15)	PowerDown, PhyStatus, AsyncPowerChangeAck	Yes
Elastic Buffer information	ElasticBufferLocation	No

8.1.5 PIPE Version 4.4

Table 8-5 lists the PIPE specification version 4.4 additional supported features and signals.

Table 8-5 PIPE Version 4.4

Feature Name	PIPE Signal (S) Used	VIP Support
SRIS (Separate Refclk Independent SSC) support	SRISEnable	Yes
Nominal Empty Elastic Buffer Mode	Elasticity Buffer Mode	Yes
L1 substates sideband signals handshake	RxEIDetectDisable, TxCommonModeDisable	Yes
Dynamic Preset Coefficient Updates (Dynamic Equalization)	LocalFS, LocalLF, GetLocalPresetCoeffcients, LocalPresetIndex, LocalTxCoefficientsValid, LocalTxPresetCoefficients	Yes
Electrical Idle sequence	TxElecIdle, RxElecIdle, RxValid	Yes
Setting negative disparity	TxCompliance	Yes
Lane turn off signaling	TxElecIdle, TxCompliance	Yes
PCIe RX margining and elastic buffer depth control (PCLK as PHY input mode)	MBI (Message Bus Interface) - M2P_MessageBus and P2M_MessageBus	Yes

8.1.6 PIPE Version 4.4.1

Table 8-6 lists the PIPE version 4.4.1 additional supported features and signals.

Table 8-6 PIPE Version 4.4.1

Feature Name	PIPE Signal (S) Used	VIP Support
PCIe RX margining and elastic buffer depth control (PCLK as PHY input/output mode)	MBI (Message Bus Interface) - M2P_MessageBus and P2M_MessageBus	Yes

VIP supports all the sub-features of PIPE 4.4 specifications when programmed to be `PIPE_SPEC_VER_4_4`. The PIPE specification version 4.4.1 has made only a single clarification above PIPE specification version 4.4. This clarification is with respect to MBI support when PCLK is PHY output. For all practical purposes both PIPE 4.4 and PIPE 4.4.1 are same. Thus, VIP does not have a separate programming mode for

PIPE_SPEC_VER_4_4_1. VIP supports all the sub-features of PIPE specification version 4.4.1 when programmed to be PIPE_SPEC_VER_4_4.

8.1.7 PIPE Version 5.1.1

The Synopsys PCIe SVT VIP supports the *PIPE Specification Version 5.1.1.* For more information, see *"PIPE5 Features"* on page 141.

8.2 RxStandby/RxStandbyStatus Handshake Support

8.2.1 Basic Attributes

VIP support for RxStandby/RxStandbyStatus handshake is enabled by default. VIP can be programmed to support RxStandby/RxStandbyStatus handshake using svt_pcie_pl_configuration::rx_standby_supported.

Note
Value of rx_standby_supported must be set to the same value in VIP as per the DUT's support for RxStandby/RxStandbyStatus handshake.

Significance of PL configuration attribute rx_standby_supported:

- When rx_standby_supported is enabled and VIP is programmed as MPIPE/MAC, VIP asserts RxStandby.
- When rx_standby_supported is enabled and VIP is programmed as SPIPE/PHY, VIP responds RxStandby with RxStandbyStatus assertion.
- When rx_standby_supported is disabled and VIP is programmed as MPIPE/MAC, VIP does not assert RxStandby.
- When rx_standby_supported is disabled and VIP is programmed as SPIPE/PHY, VIP does not need RxStandby assertion and will not respond with RxStandbyStatus.

8.2.2 Additional Attributes

- svt_pcie_pl_configuration::rx_standby_controls: Specifies the controls for RxStandby assertion by MPIPE VIP in various scenarios only when configuration attribute rx_standby_supported is set to 1.
  - Bit 0: Enables RxStandby assertion during rate change. RxStandbyStatus assertion is expected in this case.
  - Bit 1: Enables RxStandby assertion in Rx.L0s.Idle LTSSM state. RxStandbyStatus assertion is expected in this case.
  - Bit 2: Enables RxStandby assertion in P1 or lower power states (for PowerDown change in L1 and L2 LTSSM states). RxStandbyStatus assertion is not expected in this case.
  - Bit 3: Enables RxStandby assertion for PowerDown change in Detect.Quiet LTSSM state. RxStandbyStatus assertion is not expected in this case.
  - Bit 4: Enables RxStandby assertion for PowerDown change in Disabled LTSSM state. RxStandbyStatus assertion is not expected in this case.
  - Bit 5: Enables RxStandby assertion on unused/inactive lanes or turned off lanes. RxStandbyStatus assertion is not expected in this case.

- Bit 6: Enables RxStandby assertion when LTSSM moves from L0 state to Recovery state by Inferring Electrical Idle.
- Bit 7: Enables RxStandby assertion in Recovery.Speed LTSSM state even when there is no rate change.

The default value of this attribute is set to 8'b00111111.

- svt_pcie_pl_configuration::rxstandbystatus_timeout_ns: Specifies the time in ns, MPIPE VIP after asserting RxStandby will wait for RxStandbyStatus to go high only for those scenarios listed above where RxStandByStatus assertion is expected. The attribute controls the timeout value for phy_rxstandbystatus_timeout protocol check. This attribute needs to be programmed to match DUT PHY behavior so that MPIPE VIP can check DUT's timing for RxStandbyStatus signal assertion if checking actual timing is desired. The default value of this attribute is set to 1000.

8.2.3 Protocol Checks

phy_rxstandbystatus_timeout: Applicable when svt_pcie_pl_configuration::rx_standby_supported is enabled along with individual control as described by svt_pcie_pl_configuration::rx_standby_controls and PowerDown is either P0 or P0s. The intention of this protocol check is to validate that PHY responds to RxStandby with RxStandbyStatus assertion within the stipulated time period given by PL configuration attribute rxstandbystatus_timeout_ns.

8.3 Nominal Empty Mode for EFIFO

This feature is supported when VIP is compliant with PIPE specification version 4.4.

8.3.1 Interface

VIP supports this mode only in Unified interface mode. Unified interface now has a new signal called svt_pcie_pipe_if.elasticity_buffer_mode. The signal is implemented as a shared signal between all lanes.

8.3.2 Status

VIP indicates Nominal Empty mode for EFIFO by capturing the signal value and providing it as part of PIPE status class using the svt_pcie_pipe_status::elasticity_buffer_mode attribute.

8.3.3 Configuration

To enable Nominal Empty mode for EFIFO, set the svt_pcie_pl_configuration::elasticity_buffer_mode attribute.

VIP provides control over whether to assert RxDataValid in an optimized manner after it has been de-asserted to indicate FIFO empty condition. This can be programmed using svt_pcie_pl_configuration::enable_optimized_rxdatavalid_assertion attribute.

In order to verify MAC DUT behavior in Nominal Empty mode to handle empty condition of FIFO, SPIPE VIP provides the following attributes to control the number of SKP/AA symbols in SKIP OS and control over number of FIFO empty conditions within a SKIP interval and delay between subsequent FIFO empty conditions.

- `svt_pcie_pl_configuration::remove_all_skp_aa_symbols`
- `svt_pcie_pl_configuration::num_of_empty_symbols_in_skp_interval_8b10b`
- `svt_pcie_pl_configuration::num_of_empty_symbols_in_skp_interval_128b_130b`
- `svt_pcie_pl_configuration::empty_symbol_interval_8b10b[6]`
- `svt_pcie_pl_configuration::empty_symbol_interval_128b130b[24]`

For detailed information about all these attributes, see HTML class reference documentation.

8.3.4 Protocol Checks

VIP supports the following protocol checks in active and passive mode.

- `pipe_elasticity_buffer_mode_check`
- `pipe_elasticity_buffer_empty_condition_check`
- `pipe_elasticity_buffer_skip_removal_check`
- `pipe_elasticity_buffer_no_underflow_check`
- `pipe_elasticity_buffer_rxdatavalid_assertion_check`

For detailed information about these protocol checks, see HTML class reference documentation.

8.4 PIPE Signals Controllability in VIP

8.4.1 Using MPIPE VIP

Table 8-7 MPIPE Signals

Signal Name	User Controllability Availability	User APIs

TxData	Yes	Using PIPE callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCE_DATA`
		`forced_data = <value>`

TxDataK	Yes	Using PIPE callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCE_DATA_K`
		`forced_data_k = <value>`

Table 8-7 MPIPE Signals

Signal Name	User Controllability Availability	User APIs
TxDataValid	Yes	Using PIPE callback as below: • Callback Class name: svt_pcie_pl_callback • Function name: pre_pipe_data_out_put • Attach exception using svt_pcie_pipe_data_exception with below attributes: error_kind = svt_pcie_pipe_data_exception::FORCE_DATA_VALID forced_data_valid = <value>
TxStartBlock	Yes	Using PIPE callback as below: • Callback class name: svt_pcie_pl_callback • Function name: pre_pipe_data_out_put • Attach exception using svt_pcie_pipe_data_exception with below attributes: error_kind = svt_pcie_pipe_data_exception::FORCE_START_BLOCK forced_start_block = <value>
PHY Mode	Not applicable	
SRISEnable	Yes	Using PL configuration class attribute: svt_pcie_pl_configuration::sris_mode_enabled
Elasticity Buffer Mode	Yes	Using PL configuration class attribute: svt_pcie_pl_configuration::elasticity_buffer_mode
TxDetectRx/Loopback	Yes	Using PL service request as below: Service request class name: svt_pcie_pl_service Attributes used: service_type = svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL signal_name = svt_pcie_types::PIPE_TXDETECTRX event_delay = <value> event_width = <value> value = <value>
TxElecIdle	Yes	Using PIPE callback as below: • Callback class name: svt_pcie_pl_callback • Function name: pre_pipe_data_out_put • Attach exception using svt_pcie_pipe_data_exception with below attributes: error_kind = svt_pcie_pipe_data_exception::FORCE_ELEC_IDLE forced_data = <value>

Table 8-7 MPIPE Signals

Signal Name	User Controllability Availability	User APIs

TxCompliance	Yes	Using PIPE callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind =`
		`svt_pcie_pipe_data_exception::FORCE_TX_COMPLIANCE`
		`forced_tx_compliance = <value>`

RxPolarity	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type =`
		`svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL`
		`signal_name = svt_pcie_types::PIPE_RXPOLARITY`
		`event_delay = <value>`
		`event_width = <value>`
		`value = <value>`
		Using PL configuration class attribute:
		`svt_pcie_pl_configuration::invert_tx_polarity`

Reset#	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type =  svt_pcie_pl_service::PIPE_INJECT_RESET`
		`event_delay = <value>`
		`event_width = <value>`

Table 8-7 MPIPE Signals

Signal Name	User Controllability Availability	User APIs

PowerDown	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type = svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL`
		`signal_name = svt_pcie_types::PIPE_POWERDOWN`
		`event_delay = <value>`
		`event_width = <value>`
		`value = <value>`
		Using PL configuration class attributes:
		• `svt_pcie_pl_configuration::pipe_powerdown_state_for_l1_0`
		• `svt_pcie_pl_configuration::pipe_powerdown_state_for_l1_1`
		• `svt_pcie_pl_configuration::pipe_powerdown_state_for_l1_2_entry`
		• `svt_pcie_pl_configuration::pipe_powerdown_state_for_l1_2_idle`
		• `svt_pcie_pl_configuration::pipe_powerdown_state_for_l1_2_exit`

RxEIDetectDisable	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type = svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL`
		`signal_name = svt_pcie_types::PIPE_RXELECIDLE_DISABLE`
		`event_delay = <value>`
		`event_width = <value>`
		`value = <value>`
		Using PL configuration class attribute:
		• `svt_pcie_pl_configuration::l1ss_rxelecidle_disable_assertion_delay_ns`
		• `svt_pcie_pl_configuration::l1ss_rxelecidle_disable_deassertion_delay_ns`

Table 8-7 MPIPE Signals

Signal Name	User Controllability Availability	User APIs

TxCommonModeDisable	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type = svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL`
		`signal_name = svt_pcie_types::PIPE_TXCOMMONMODE_DISABLE`
		`event_delay = <value>`
		`event_width = <value>`
		`value = <value>`
		Using PL configuration class attribute:
		• `svt_pcie_pl_configuration::l1ss_txcommonmode_disable_assertion_delay_ns`
		• `svt_pcie_pl_configuration::l1ss_txcommonmode_disable_deassertion_delay_ns`

Rate	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type = svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL`
		`signal_name = svt_pcie_types::PIPE_RATE`
		`event_delay = <value>`
		`event_width = <value>`
		`value = <value>`
		Using PL configuration class function:
		`svt_pcie_pl_configuration::set_link_speed_values`

Width	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type = svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL`
		`signal_name = svt_pcie_types::PIPE_WIDTH`
		`event_delay = <value>`
		`event_width = <value>`
		`value = <value>`
		Using PL configuration class attribute:
		`svt_pcie_pl_configuration::pipe_width`

Table 8-7 MPIPE Signals

Signal Name	User Controllability Availability	User APIs

PCLK Rate	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type =`
		`svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL`
		`signal_name = svt_pcie_types::PIPE_PCLK_RATE`
		`event_delay = <value>`
		`event_width = <value>`
		`value = <value>`
		Using PL configuration class attribute:
		`svt_pcie_pl_configuration::pclk_rate`

TxDeemph	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type =  svt_pcie_pl_service::SET_LANE_TX_DEEMPH`
		`tx_deemph = <value>`
		Using PL configuration class attribute:
		• `svt_pcie_pl_configuration::tx_select_deemphasis`
		• `svt_pcie_pl_configuration::preset_to_coefficients_mapping_table*`

RxPresetHint	Yes	Using PL configuration class function: • `svt_pcie_pl_configuration::upstream_receiver_preset_hint*` • `svt_pcie_pl_configuration::downstream_receiver_preset_hint*`

LocalPresetIndex	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type =`
		`svt_pcie_pl_service::PIPE_GET_LOCAL_PRESET_COEFFICIENTS`
		`event_delay = <value>`
		`preset_value = <value>`
		Using PL configuration class attribute:
		`svt_pcie_pl_configuration::enable_get_local_preset_coefficients`

Table 8-7 MPIPE Signals

Signal Name	User Controllability Availability	User APIs

GetLocalPreset Coeffcients	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type = svt_pcie_pl_service::PIPE_GET_LOCAL_PRESET_COEFFICIENTS`
		`event_delay = <value>`
		`preset_value = <value>`
		Using PL configuration class attribute:
		`svt_pcie_pl_configuration::enable_get_local_preset_coefficients`

FS	Yes	Using PL configuration class function: • `svt_pcie_pl_configuration::use_dynamic_fs_lf_values` • `svt_pcie_pl_configuration::fs_value*` • `svt_pcie_pl_configuration::attached_fs*`
LF	Yes	Using PL configuration class function: • `svt_pcie_pl_configuration::use_dynamic_fs_lf_values` • `svt_pcie_pl_configuration::fs_value*` • `svt_pcie_pl_configuration::attached_fs*`

RxEqEval	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type = svt_pcie_pl_service::QUEUE_EQ_TX_REQUEST_PRESET_COEFF`
		`preset_valid = <value>`
		`preset_value = <value>`
		`cursor_coeff = <value>`
		`precursor_coeff=<value>`
		`postcursor_coeff=<value>`
		`expect_reject=<value>`
		Using PL configuration class attribute:
		• `svt_pcie_pl_configuration::pipe_rxeqeval_assertion_delay`
		• `svt_pcie_pl_configuration::pipe_rxeqeval_deassertion_delay`

RxEqInProgress	Yes	Using PL configuration class function: • `svt_pcie_pl_configuration::simultaneous_deassertion_of_rxeqinprogress` • `svt_pcie_pl_configuration::pipe_rxeqinprogress_assertion_delay` • `svt_pcie_pl_configuration::pipe_rxeqinprogress_deassertion_delay`

Table 8-7 MPIPE Signals

Signal Name	User Controllability Availability	User APIs
InvalidRequest	Yes	Using PL configuration class function: • `svt_pcie_pl_configuration::min_num_pclk_cycles_to_assert_invalid_request` • `svt_pcie_pl_configuration::max_num_pclk_cycles_to_assert_invalid_request`
TxMargin	Yes	Using PL configuration class function: `svt_pcie_pl_configuration::tx_margin`
TxSwing	Yes	Using PL configuration class function: `svt_pcie_pl_configuration::tx_swing`

TxSyncHeader	Yes	Using pipe callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCE_SYNC_HDR`
		`- forced_sync_hdr = <value>`

BlockAlignControl	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type =`
		`svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL`
		`signal_name = svt_pcie_types::PIPE_BLOCK_ALIGN_CONTROL`
		`event_delay = <value>`
		`event_width = <value>`
		`value = <value>`

RxStandby	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type =`
		`svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL`
		`signal_name = svt_pcie_types::PIPE_RXSTANDBY`
		`event_delay = <value>`
		`event_width = <value>`
		`value = <value>`
		Using PL configuration class function:
		• `svt_pcie_pl_configuration::rx_standby_supported`
		• `svt_pcie_pl_configuration::rx_standby_controls`

Table 8-7 MPIPE Signals

Signal Name	User Controllability Availability	User APIs
EncodeDecode Bypass	Not Available	
PclkChangeAck	Yes	Using PL configuration class function: `svt_pcie_pl_configuration::pclk_change_ack_delay`
AsyncPowerChangeAck	Yes	Using PL configuration class function: `svt_pcie_pl_configuration::async_power_change_ack_delay`

M2P_MessageBus	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type =  svt_pcie_pl_service::MBI_CMD`
		`mbi_lane_num= <value>`
		`mbi_cmd=<value>`
		`mbi_addr=<value>`
		`mbi_data=<value>`

8.4.2 Using SPIPE VIP

Table 8-8 SPIPE Signals

Signal Name	User Controllability Availability	User APIs

RxData	Yes	Using pipe callback as below:
		• Callback Class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCE_DATA`
		`- forced_data = <value>`

RxDataK	Yes	Using pipe callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCE_DATA_K`
		`- forced_data_k = <value>`

Table 8-8 SPIPE Signals

Signal Name	User Controllability Availability	User APIs

RxDataValid	Yes	Using pipe callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCE_DATA_VALID`
		`- forced_data_valid = <value>`

RxStartBlock	Yes	Using pipe callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCE_START_BLOCK`
		`- forced_start_block = <value>`

RxElecIdle	Yes	Using pipe callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCE_ELEC_IDLE`
		`- forced_data = <value>`

LocalFS	Yes	Using PL configuration class function: `svt_pcie_pl_configuration::local_fs*`
LocalLF	Yes	Using PL configuration class function: `svt_pcie_pl_configuration::local_lf*`

RxSyncHeader	Yes	Using pipe callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCE_SYNC_HDR`
		`- forced_sync_hdr = <value>`

Table 8-8 SPIPE Signals

Signal Name	User Controllability Availability	User APIs
RxStandbyStatus	Yes	Using PL service request as below: Service request class name: svt_pcie_pl_service Attributes used: service_type = svt_pcie_pl_service::PIPE_DRIVE_SIDEBAND_SIGNAL signal_name = svt_pcie_types::PIPE_RXSTANDBY_STATUS event_delay = <value> event_width = <value> value = <value> Using PL configuration class function: svt_pcie_pl_configuration::rx_standby_supported
PclkChangeOk	Yes	Using PL configuration class function: svt_pcie_pl_configuration::pclk_change_ok_delay
P2M_Message Bus	No	
LocalTxPresetCoefficients	Yes	Using PL configuration class function: • svt_pcie_pl_configuration::enable_get_local_preset_coefficients • svt_pcie_pl_configuration::preset_to_coefficients_mapping_table* • svt_pcie_pl_configuration::min_spipe_preset_coefficients_delay • svt_pcie_pl_configuration::max_spipe_preset_coefficients_delay
LocalTxCoefficientValid	Yes	Using PL configuration class function: • svt_pcie_pl_configuration::enable_get_local_preset_coefficients • svt_pcie_pl_configuration::preset_to_coefficients_mapping_table* • svt_pcie_pl_configuration::min_spipe_preset_coefficients_delay • svt_pcie_pl_configuration::max_spipe_preset_coefficients_delay

Table 8-8 SPIPE Signals

Signal Name	User Controllability Availability	User APIs

LinkEvaluationFeedbackFigureMerit	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type = svt_pcie_pl_service::QUEUE_EQ_FIGURE_MERIT_RESPONSE`
		`eq_lane_num = <value>`
		`figure_of_merit = <value>`
		Using PL configuration class attribute:
		• `svt_pcie_pl_configuration::min_rx_eq_eval_delay`
		• `svt_pcie_pl_configuration::max_rx_eq_eval_delay`

LinkEvaluationFeedbackDirectionChange	Yes	Using PL service request as below:
		Service request class name: svt_pcie_pl_service
		Attributes used:
		`service_type = svt_pcie_pl_service::QUEUE_EQ_DIRECTION_CHANGE_RESPONSE`
		`eq_lane_num  = <value>`
		`direction_change_response = <value>`
		Using PL configuration class attribute:
		• `svt_pcie_pl_configuration::min_rx_eq_eval_delay`
		• `svt_pcie_pl_configuration::max_rx_eq_eval_delay`

RxValid	No	
PhyStatus	Yes	Using PL configuration class function: • `svt_pcie_pl_configuration::min_spipe_phystatus_delay` • `svt_pcie_pl_configuration::max_spipe_phystatus_delay` • `svt_pcie_pl_configuration::enable_per_lane_spipe_phystatus_rand_delay` Using `svt_pcie_pipe_phystatus_response_configuration` handle present inside `svt_pcie_pl_configuration`. `svt_pcie_pipe_phystatus_response_configuration phystatus_response_cfg[$].`

RxStatus	Yes	Using pipe callback as below:
		• Callback class name: svt_pcie_pl_callback
		• Function name: pre_pipe_data_out_put
		• Attach exception using svt_pcie_pipe_data_exception with below attributes:
		`error_kind = svt_pcie_pipe_data_exception::FORCED_RX_STATUS`
		`- forced_rx_status = <value>`

Table 8-8 SPIPE Signals

Signal Name	User Controllability Availability	User APIs
ElasticBufferLocation	Not available	
DataBusWidth	No	

9 PIPE5 Features

This chapter describes the PIPE5 feature support available with Synopsys PCIe Verification IP.

This chapter discusses the following topics:

9.1 Version Support

The Synopsys PCIe SVT VIP supports the *PIPE Specification Version 5.1.1, July 2018*. Functionality and controls will be updated as the specification changes.

9.2 Supported Interfaces

Support to PIPE5 feature set is available when using the `svt_pcie_pipe5_if` interface with PCIe SVT Unified model. For more details, PCIe PIPE Interface.

Note
PIPE5 support for legacy instantiation models will not be supported.

9.3 Supported Features

9.3.1 PIPE 5.1.1 Features

Following are the features introduced in *PIPE Specification Version 5.1.1*:

- Mapping of legacy signals with Message Bus Registers
- Low pin count interface

- `m2p_message_bus`
- `p2m_message_bus`
- SERDES architecture
  - Including 64-bit wide data path

The *PIPE Specification Version 5.1.1* provides PIPE interface for devices compatible with *PCI Express Base Specification Version 5.0* or earlier. However, PCIe VIP provides support to *PIPE Specification Version 5.1.1* only when programmed as a device compatible with *PCI Express Base Specification Revision 5.0 Version 0.7*. With this approach, VIP provides a new low pin count interface named `svt_pcie_pipe5_if` where legacy signals are not available. The `svt_pcie_pipe5_if` is added as a subinterface of `svt_pcie_if` interface.

Note
To enable support to Gen5 speed when using traditional architecture and legacy signals using *PIPE Specification Version 4.4* or *4.4.1*, you must use `svt_pcie_pipe_if` and enable Gen5 speed support as described in Gen5 Support for PIPE Interface.

9.3.2 PIPE 5.1.1 Features Supported in Active VIP

Following are the PIPE 5.1.1 features supported in this release:

- Initial link negotiation
  - x1 to x32
- Speed negotiation for speeds 2.5G, 5G, 8G, 16G, and 32G
- PIPE `Width`/`Rate`/`PCLKRate` configurations as per Appendix PCIe PIPE Interface.
- Per lane sideband signals in `svt_pcie_pipe5_if`
- Low power states, L1SS with sideband signals
- `PclkChangeOk`/`PclkchangeAck` handshake support for rate and width changes
- Receiver detection in P2 power states
- MAC and PHY register space associated with MBI
  - All MAC and PHY registers compatible with PIPE 5.1.1
  - API to generate MBI commands from the testbench.

    The `svt_pcie_pl_service_request_mbi_cmd_sequence` service sequence is used to initiate MBI commands.
  - Auto-generation of `WriteAck` and `ReadCompletion` in response to received write and read.
  - Tracking of MAC and PHY registers of MBI in MPIPE and SPIPE mode.
- Mapping of legacy handshake over MBI

    - Polarity Inversion
    - Block Align Control
    - Receiver Equalization
    - Dynamic Preset Coefficient updates
    - `FS`, `LF`, `LocalFS`, `LocalLF`, `LocalG4FS`, `LocalG4LF`, `LocalG5FS`, `LocalG5LF`, and `TxDeemph`
    - Rx Margining
    - Lane Margining
- MBI Handshakes for lane reversal configurations
- `RefClkRequired#` signal
- Functional coverage for Original PIPE Architecture and Low Pin Count Interface
- Mapping of legacy signals into Message Bus registers
- Low pin count interface
    - `m2p_message_bus`
    - `p2m_message_bus`
- SerDes architecture
- Including 64-bit wide data path
- Custom support for `PCLK` as PHY output. For more details, see VIP Requirements.

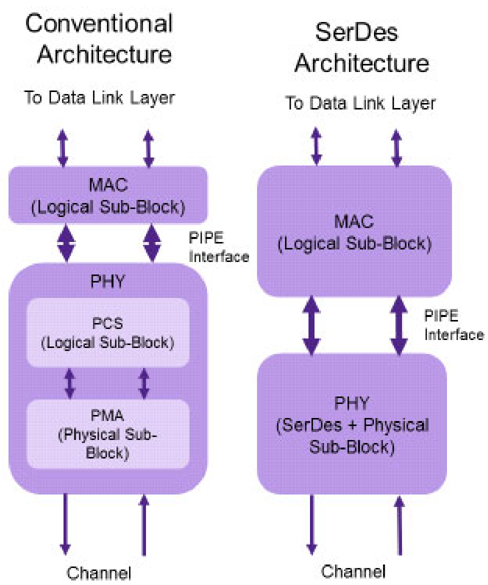
Note
The existing VIP APIs for the above features are applicable for PIPE 5.1.1 and earlier versions. Additionally, the handshake mechanism is the only feature updated for PIPE 5.1.1.

Note
The PIPE 5.1.1 specification mandates `PCLK` as PHY input.

9.3.2.1 SerDes Architecture

Support for SerDes architecture is available from P-2019.06-3 release. Figure 9-1 shows the comparison between SerDes architecture and traditional architecture.

Figure 9-1 Traditional Architecture Vs SerDes Architecture



Note
Verification of SerDes architecture is ongoing. This feature is in EA stage.

9.3.2.1.1 Enabling SerDes Mode

Serdes architecture mode is enabled using a configuration attribute `svt_pcie_pl_configuration::enable_serdes_arch`.

Additionally, you must define the `SVT_PCIE_ENABLE_SERDES_ARCH` macro.

9.3.2.1.2 Controlling RX Data Path PIPE Configuration

In SerDes architecture mode, the TX data path PIPE configuration is still controlled using the existing `pipe_width` and `pclk_rate` attributes from `svt_pcie_pl_configuration` class. The RX data path configurations can now be controlled using `svt_pcie_pl_configuration::pipe_rx_width` attribute. For detailed information of this attribute, see HTML class reference guide.

9.3.2.1.3 Controlling SSC, PPM, and rxclk Skew Settings

In SerDes architecture mode, when the VIP is operating as SPIPE mode, VIP provides controls over SSC, PPM and skew settings of `rxclk`. Refer to the existing attributes `ssc_max_spread`, `ssc_modulation_rate`, and `fixed_ppm_due_to_tx_rx_xo` for SSC and PPM controls of `rxclk`. For detailed information about these attributes, see `svt_pcie_pl_configuration::min_rx_clk_skew` and `svt_pcie_pl_configuration::max_rx_clk_skew` attributes attributes.

9.4 VIP Requirements

To enable PIPE 5.1.1 feature, set the following configurations:

- Set the following Gen4 and Gen5 licenses:
  - `VIP-PCIE-G4-SVT`
  - `VIP-PCIE-G5-SVT`

  For more details, see Licensing Information.
- Define the `SVT_PCIE_ENABLE_GEN5` macro.

- Define the `SVT_PCIE_ENABLE_PIPE5` macro.
- Set the PCIe specification version.

  `svt_pcie_device_configuration::PCIE_SPEC_VER_5_0`
- Set the PIPE specification version.

  `svt_pcie_device_configuration::PIPE_SPEC_VER_5_1`
- Set the interface `svt_pcie_pipe5_if`.

  For more details, PIPE Interface Usage Model.
- To enable PCLK as PHY output feature, define the `SVT_PCIE_ENABLE_PIPE5_PCLK_AS_PHY_OUTPUT_MODE` mode (this mode is outside the PIPE 5.1 specification which states that PCLK as PHY output mode is not supported for PCIe 5.0 and above).
- To enable SerDes architecture mode, you must define `SVT_PCIE_ENABLE_SERDES_ARCH`.

9.5 Limitations

VIP supports PIPE 5.1.1 only when used as an Active agent. Currently, PIPE 5.1.1 is not supported for VIP as a Passive agent. The features listed below will be available in the upcoming releases.

9.5.1 Known Limitations in Active VIP

Following PIPE 5.1.1 features are not supported in this release:

- Support to CCIX ESM speeds when using `svt_pcie_pipe5_if`.
- 64/80 bit PIPE width is not fully supported.

9.6 PIPE Interface Usage Model

The PCIe SVT VIP implements `svt_pcie_if` interface for DUT connectivity. For PIPE interfaces, `svt_pcie_if` offers two subinterfaces namely `svt_pcie_pipe_if` and `svt_pcie_pipe5_if`. You must use only one PIPE interface—that is, either `svt_pcie_pipe_if` or `svt_pcie_pipe5_if`. To choose between `svt_pcie_pipe_if` and `svt_pcie_pipe5_if`, refer to the descriptions in Applicable Scenarios for Using svt_pcie_pipe_if and Applicable Scenarios for Using svt_pcie_pipe5_if respectively.

9.6.1 Applicable Scenarios for Using svt_pcie_pipe_if

- Supports *PIPE Specification Version 4.4.1* or earlier.
- To be used only when:
  - Design/DUT is compatible with *PCIe Specification Version 4.0* or earlier.
  - Using a custom mode where 64-bit data width is required to achieve Gen3/Gen4/Gen5 speeds (for *PIPE Specification Version 4.3* and later).
  - Using a custom enhancement which supports PIE8 interface.
  - Using a custom enhancement which supports Gen5 speed and *PIPE Specification Version 4.4.1*.

9.6.2 Applicable Scenarios for Using svt_pcie_pipe5_if

- Supports *PIPE Specification Version 5.1.1* and later.
- To be used only when design/DUT is compatible with *PCIe Specification Version 5.0* and later and *PIPE Specification Version 5.1.1* and later.

- Using a custom mode where 64-bit data width is required to achieve Gen1/Gen2/Gen3/Gen4/Gen5 speeds (for *PIPE Specification Version 5.1.1* and later).

9.6.3 Features Supported in svt_pcie_pipe_if and svt_pcie_pipe5_if

Table 9-1 lists the supported features in `svt_pcie_pipe_if` and `svt_pcie_pipe5_if`.

Table 9-1 Feature Comparison

Features	svt_pcie_pipe_if	svt_pcie_pipe5_if
Legacy pin interface	Supported	Not supported
Low pin count interface	Supported ONLY for RX margining	Supported
Original PIPE architecture	Supported	Supported
SERDES architecture	Not supported	Supported
Link speeds up to 16G	Supported	Supported
32G link speed	Supported with custom enhancement	Supported
Compatible with PCIe specification version	4.0	5.0

10 PCIe Verification Topologies

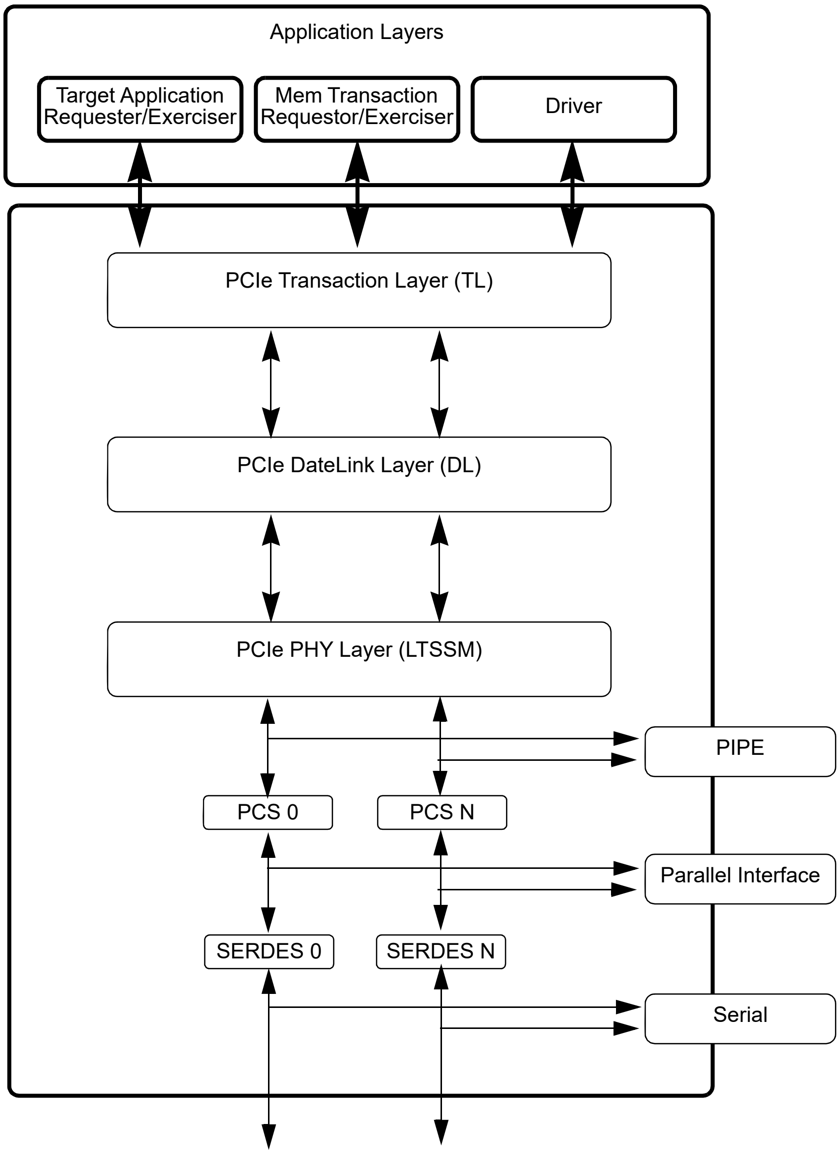
This chapter shows the supported interfaces of the PCIe VIP. It contains the following sections:

10.1 Introduction

The PCI Express Transceiver VIP is a bus functional model that can generate and respond to PCI Express transactions. It can be used to verify PCI Express endpoint, switch, or root complex devices.

Figure 10-1 shows all the supported layers in the PCIe VIP. The layers below the PHY layer will be supported based on the interface selected for the VIP. If the SERDES interface is selected, then the PCS and SERDES layers are included in the VIP along with the other three layers. If the Parallel interface is selected, than the PCS layer gets included along with the other three layers. For the PIPE interface, the VIP will contain the Transaction Layer, the Data Link Layer, and the PHY Layer.

Figure 10-1 PCIe VIP Structure



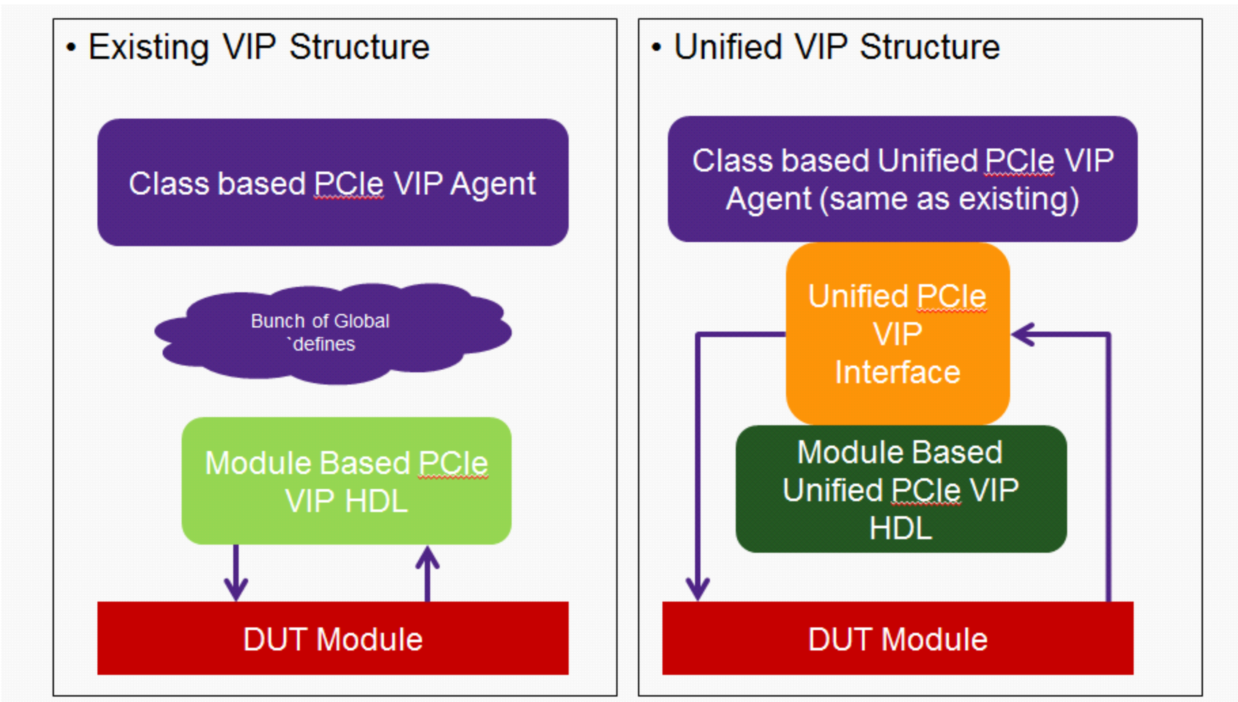
10.2 Unified PCIe VIP Component

10.2.1 Introduction

This section provides key features and the use model of SVT PCIe Unified (or PCIe Single Port Device) VIP component for use in the UVM testbench. The Unified PCIe VIP component is a combination of Unified PCIe VIP Agent (`UVM_COMPONENT`), Unified PCIe VIP Interface (`svt_pcie_if`) and Unified PCIe VIP HDL (`svt_pcie_single_port_device_agent_hdl`) module.

Note
Unified model is the interconnect moving forward as the legacy instantiation models will be deprecated in the future releases.

Figure 10-2 Existing VIP and Unified VIP Structure



10.2.2 Existing VIP Structure

The existing VIP consists of SystemVerilog HDL module and global defines. At the core of the model is a SystemVerilog HDL module. It handles all basic tasks in managing the PCIe stack. It communicates with an UVM SystemVerilog Agent which is the interface to the user's testbench. You configure the model using both UVM configuration classes and global defines.

Connecting the model to the user DUT is done in the following ways:

- Signal connectivity is done through the Verilog module.
- There are separate HDL instantiation models defining all width, rate, and PHY connectivity variations (PIPE/SERDES/PMA). There are currently 40+ instantiation models that define all permissible variations of PIPE, SERDES, and PMA interfaces. For more information on instantiation models, see Instantiation Models.
- The UVM based interface model (dynamic) and the Verilog module interfaces (static) rely on a combination of Verilog defines and parameters to control the compile type connection topology of VIP. The macro based settings being global in nature restrict the testbench to use of a single setting across multiple VIP instances.
- As a result, these factors limit VIP usage when trying to verify highly configurable, multi-link PCIe designs.
- Signal connections to passive monitor component is done as a separate step.

10.2.3 Unified PCIe VIP Component

The PCIe VIP is enhanced with a Unified interface VIP component to use the model seamlessly for configurable single link/multi-link designs. It consists of the following components:

- Uses Unified PCIe Verilog HDL VIP as single SystemVerilog module configurable through parameters (does not rely on defines).
- Uses a Unified PCIe VIP Interface as single interface object. This interface is non-parameterized and non-macro based. Each of the sub-interfaces supports maximum link width of 32 lanes.
  - The signal names used in the sub-interfaces are all in lower case. If the specification refers these using mixed case, then the rule followed is insert an `_` preceding each capital letter and convert the string to lower case. The trailing numerical portion in signal names is used to indicate the specific lane that the signal is associated with. An example showing TxData for lane 0 is represented by tx_data_0[31:0] and TxData for lane 1 is represented by tx_data_1[31:0].
  - Since the model supports multiple versions of PIPE specification, it implements some signals on shared basis (per link), and some on per lane basis. For example, PIPE Spec version 4.2 and above support supports the option of the PCLK signal being driven by Controller instead of the PHY (that is, parameter `SVT_PCIE_UI_PIPE_CLK_FROM_MAC` is set to 1).

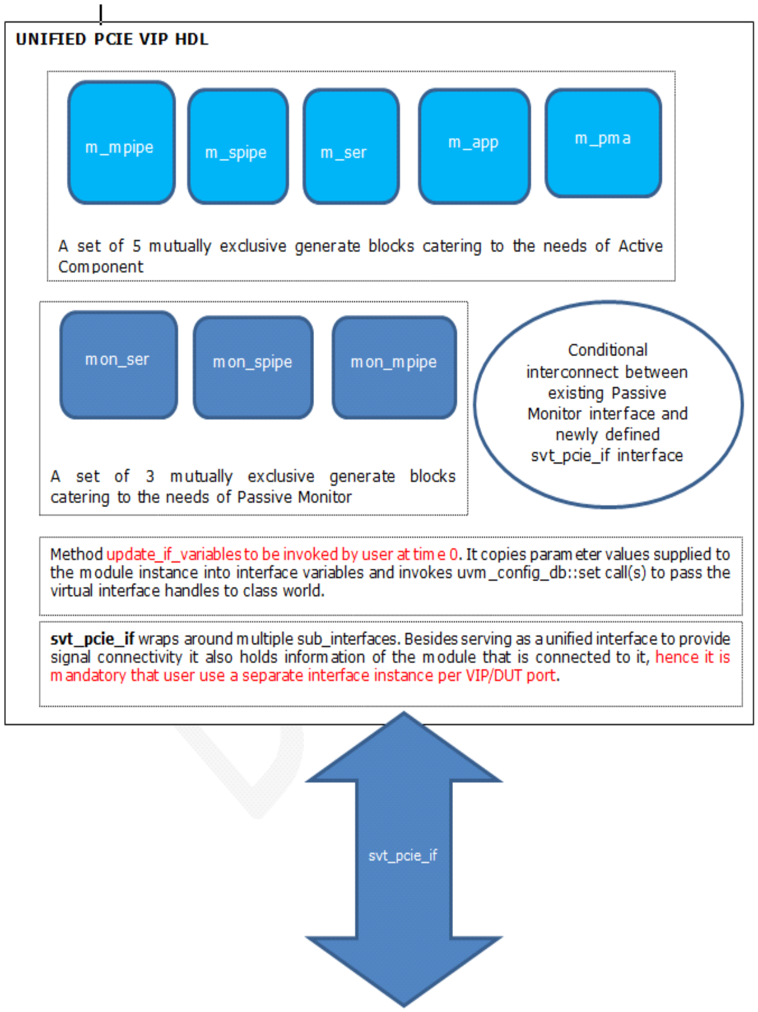
    This functionality is modeled using pclk_* signals (on per lane basis), but for PIPE Spec Version prior to 4.2, the PCLK signal is driven from the PHY to controller and this is modeled using pclk (on shared basis).

    For more details on individual signals, see the in-line comments in *svt_pcie_pipe_if.svi*, *svt_pcie_pma_if.svi*, and *svt_pcie_serdes_if.svi*. files available at *<design dir>/include/sverilog/*.

    See also, `svt_pcie_if (`Interfaces tab) in the in PCIe VIP UVM class reference.
- Shared signal hookup between Active and Passive Monitor VIP components. Scalable and simplified integration use model.

Figure 10-3 provides an overview of the Unified PCIe Verilog HDL VIP Framework.

Figure 10-3 Unified PCIe Verilog HDL VIP



Note

For debugging the active VIP component, you can use the waveform viewer to access the internal nodes by accessing one of the following paths. The strings in bold represent the name of the mutually exclusive generate blocks shown above.

```
Instance_of_svt_pcie_single_port_device_agent_hdl.m_ser.port0...   OR
Instance_of_svt_pcie_single_port_device_agent_hdl.m_pma.port0...   OR
Instance_of_svt_pcie_single_port_device_agent_hdl.m_spipe.port0... OR
Instance_of_svt_pcie_single_port_device_agent_hdl.m_mpipe.port0... OR
Instance_of_svt_pcie_single_port_device_agent_hdl.m_app.port0...
```

10.2.3.1 Configurable Parameters for the Unified PCIe Verilog HDL VIP

The Unified VIP instance supports a number of configuration settings using some parameters. Some of these parameters should not be changed post compilation. These are called static parameters. Other parameters can be changed during the simulation through various configuration classes, such as `svt_pcie_device_configuration`. These are referred to as dynamic parameters.

Table 10-1 shows the list of available parameters.

Table 10-1 Configurable Parameters for the Unified PCIe Verilog HDL VIP

Parameter Name	Default	Static Vs Dynamic	Remarks
SVT_PCIE_UI_PCIE_SPEC_VER	SVT_PCIE_UI_PCIE_SPEC_VER_3_0	Dynamic	Represents PCIe spec version that the VIP instance implements.
SVT_PCIE_UI_PIPE_SPEC_VER	SVT_PCIE_UI_PIPE_SPEC_VER_4	Static	Represents PIPE spec version that the VIP instance implements.
SVT_PCIE_UI_SERIAL_CLK_TOLERANCE	0.000100	Static	Controls the behavior of VIP receive path during the clock recovery process.
SVT_PCIE_UI_NUM_PHYSICAL_LANES	32	Dynamic as long as the new value is less than the value set at the compile time.	Represents the maximum number of lanes to be supported by the VIP instance. The legal values are 1, 2, 4, 8, 16, and 32.
SVT_PCIE_UI_NUM_PMA_INTERFACE_BITS	10	Static	Represents the maximum number of bits (per lane) that need to be supported by PMA interface of the VIP. The legal values are 10,16,20,32,40,64,130.
SVT_PCIE_UI_NUM_PIPE_INTERFACE_BITS	32	Dynamic	Represents the maximum number of bits (per lane) that need to be supported by the PIPE interface of the VIP. The legal values are 8,16, and 32.
SVT_PCIE_UI_DISPLAY_NAME	single_port_device	Static	Instance name of the HDL module along with the trailing ".". It is used to identify instance in symbol log file.
SVT_PCIE_UI_HIERARCHY_NUMBER	0	Static	Used to distinguish PCIe tree originating at different roots in a multi root system.

Table 10-1 Configurable Parameters for the Unified PCIe Verilog HDL VIP

Parameter Name	Default	Static Vs Dynamic	Remarks
SVT_PCIE_UI_ENABLE_SHADOW_MEMORY_CHECKING	1	Dynamic	The application layer of VIP has a logic to check payload from end-to-end transfer point of view. Setting this parameter to 1 allows testbench to enable this checker.
SVT_PCIE_UI_ENABLE_CFG_BLOCK	1	Dynamic	When set to 1, the Active VIP Driver and Requester application layer components implement a structure similar to PCIe configuration space.
SVT_PCIE_UI_MPIPE	1	Static	Applicable only if `SVT_PCIE_UI_PHY_INTERFACE_TYPE = ` `` `SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE `` Or `` SVT_PCIE_UI_MON_PHY_INTERFACE_TYPE=`SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE `` If 1, indicates instance represents a PIPE compliant “MAC”. If 0, indicates instance represents a PIPE compliant “PHY”.
SVT_PCIE_UI_PHY_INTERFACE_TYPE	`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES	Static	The legal Values are: • `SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDE` • `SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE` • `SVT_PCIE_UI_PHY_INTERFACE_TYPE_PMA` • `SVT_PCIE_UI_PHY_INTERFACE_TYPE_APP`
SVT_PCIE_UI_DEVICE_IS_ROOT	1	Dynamic	If 1, then indicates, instance is configured as a Root Complex. Any other value is an Endpoint. If 0, all layers of active component are turned off.
SVT_PCIE_UI_CONNECT_ACTIVE_VIP	1	Static	If 0, all layers of active component are turned off.

Table 10-1 Configurable Parameters for the Unified PCIe Verilog HDL VIP

Parameter Name	Default	Static Vs Dynamic	Remarks
SVT_PCIE_UI_CONNECT_DOWNSTREAM_PORT_MONITOR	0	Static	If 0, then monitor interface within this instance is optimized out. If 1, a monitor interface is instantiated and connected such that traffic transmitted by Root Complex is checked against spec requirements. The handle to this interface is then passed to a `uvm_component` instance.
SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR	0	Static	If 0, then monitor interface within this instance is optimized out. If 1, a monitor interface is instantiated and connected such that traffic transmitted by Endpoint is checked against spec requirements. The handle to this interface is then passed to a `uvm_component` instance.
SVT_PCIE_UI_DUT_IN_V2V_CTL_TB	1	Static	Applicable only to VIP-to-VIP testbenches. No impact on standalone VIP functionality. **Note**: Do not care from DUT connections point of view.
SVT_PCIE_UI_PIPE_CLK_FROM_MAC	0	Static	Applicable only if `SVT_PCIE_UI_PHY_INTERFACE_TYPE = ``SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE` PIPE spec 4.2 and above support `pclk` to be sourced by MAC instead of PIPE. When set to 1, VIP instance (with SVT_PCIE_UI_MPIPE = 1) will source the PCLK.
SVT_PCIE_UI_ENABLE_IO_SKEW	0	Static	Can be set to 1 if SVT_PCIE_UI_MPIPE = 1. This parameter enables clocking block for PIPE signals.
SVT_PCIE_UI_SETUP_PS	10ps	Static	Applicable if `SVT_PCIE_UI_MPIPE = 1 & SVT_PCIE_UI_ENABLE_IO_SKEW = 1` This real-time parameter specifies setup time of clocking block. Smallest non-zero value can be 1fs.

Table 10-1 Configurable Parameters for the Unified PCIe Verilog HDL VIP

Parameter Name	Default	Static Vs Dynamic	Remarks
SVT_PCIE_UI_HOLD_PS	5ps	Static	Applicable if `SVT_PCIE_UI_MPIPE = 1 & SVT_PCIE_UI_ENABLE_IO_SKEW = 1` This real-time parameter specifies hold time of clocking block. Smallest non zero value can be 1fs.

SVT_PCIE_UI_MON_PHY_INTERFACE_TYPE	`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES	Static	Used to select the type of physical interconnect for passive monitor. Legal values are one of the following
			`` `SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE ``
			`` `SVT_PCIE_PHY_INTERFACE_TYPE_SERDES ``
			The selected value must be equal to `SVT_PCIE_UI_PHY_INTERFACE_TYPE` unless `SVT_PCIE_UI_PHY_INTERFACE_TYPE` is set to `` `SVT_PCIE_UI_PHY_INTERFACE_TYPE_APP ``.

SVT_PCIE_UI_PIPE_MAX_PCLK_SUPPORTED	`SVT_PCIE_PIPE_MAX_PCLK_1000_MHZ	Static	Represents the `max_pclk` supported by PHY. Legal values are: • `` `SVT_PCIE_PIPE_MAX_PCLK_4000_MHZ `` • `` `SVT_PCIE_PIPE_MAX_PCLK_2000_MHZ `` • `` `SVT_PCIE_PIPE_MAX_PCLK_1000_MHZ `` • `` `SVT_PCIE_PIPE_MAX_PCLK_500_MHZ `` • `` `SVT_PCIE_PIPE_MAX_PCLK_250_MHZ ``

While instantiating the Unified PCIe VIP HDL module, the testbench must configure the above parameters according to testbench needs.

For more details on how to set the values for individual instances, see the example file *top.pcie_pipe_topology.sv* available in the *examples* directory.

Table 10-2 Parameters Applicable to Unified VIP Instances

Parameter	Applicable to Active Unified VIP Instances Representing			
	PIPE Master	PIPE Slave	Serial Link Partner	Application Layer
SVT_PCIE_UI_PCIE_SPEC_VER	√	√	√	√
SVT_PCIE_UI_PIPE_SPEC_VER	√	√	X	X
SVT_PCIE_UI_NUM_PHYSICAL_LANES	√	√	√	X
SVT_PCIE_UI_NUM_PMA_INTERFACE_BITS	X	X	X	X
SVT_PCIE_UI_NUM_PIPE_INTERFACE_BITS	√	√	X	X
SVT_PCIE_UI_DISPLAY_NAME	√	√	√	√
SVT_PCIE_UI_HIERARCHY_NUMBER	√	√	√	√
SVT_PCIE_UI_ENABLE_SHADOW_MEMORY_CHECKING	√	√	√	√
SVT_PCIE_UI_ENABLE_CFG_BLOCK	√	√	√	√
SVT_PCIE_UI_MPIPE	√	√	X	X
SVT_PCIE_UI_PHY_INTERFACE_TYPE	√	√	√	√
SVT_PCIE_UI_DEVICE_IS_ROOT	√	√	√	√
SVT_PCIE_UI_CONNECT_ACTIVE_VIP	√	√	√	√
SVT_PCIE_UI_CONNECT_DOWNSTREAM_PORT_MONITOR	0	0	0	0
SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR	0	0	0	0
SVT_PCIE_UI_DUT_IN_V2V_CTL_TB	0	0	0	0
SVT_PCIE_UI_PIPE_CLK_FROM_MAC	√	√	X	X
SVT_PCIE_UI_ENABLE_IO_SKEW	√	0	0	0
SVT_PCIE_UI_SETUP_PS	√	0	0	0
SVT_PCIE_UI_HOLD_PS	√	0	0	0
SVT_PCIE_UI_MON_PHY_INTERFACE_TYPE	X	X	X	X
SVT_PCIE_UI_SERIAL_CLK_TOLERANCE	X	X	√	X
SVT_PCIE_UI_PIPE_MAX_PCLK_SUPPORTED	√	√	X	X

Table 10-2 Parameters Applicable to Unified VIP Instances

Parameter	Applicable to Active Unified VIP Instances Representing			
	PIPE Master	PIPE Slave	Serial Link Partner	Application Layer
√ – Implies the parameter is valid and can accept any legal value. X – Implies its a don't care. 0 – Implies the default value.				

Note
Any signal corresponding to lanes less than NUM_PHYSICAL_LANES should not be left floating. 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 1.

10.2.4 Verilog HDL Module to SystemVerilog UVM Class Intercommunication

The svt_pcie_if interface acts as a bridge between Verilog HDL based module and the SystemVerilog UVM class-based testbench. To accommodate the testbenches and DUT topologies, you can configure svt_pcie_if. See Table 10-2 for the list of parameters available for the testbench.

By using the svt_pcie_if as a conduit to carry compile-time information from the Verilog module to UVM testbench, it eliminates the need for shared defines (for example, SVT_PCIE_MAX_LINK_WIDTH).

The parameters listed above are captured as variables in svt_pcie_if. These variables allow the testbench to tune the values of (and/or constraints on) members of the svt_pcie_device_configuration class.

Note
Irrespective of the topology of testbench (SERDES, PIPE, Single Link, or MultiLink) or the DUT type (PHY, RC, EP, Switch, and Repeater) the testbench must ensure that an svt_pcie_if instance is not shared between two Verilog VIP HDL instances.

10.2.5 Sharing Signal Connections Between Passive Monitor and the Active Agent Component

To simplify the use model for testbench using both the active and passive monitor components of the UVM PCIe Agent in the same testbench, you can use following Unified PCIe Verilog HDL VIP parameters:

- SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR
- SVT_PCIE_UI_CONNECT_DOWNSTREAM_PORT_MONITOR

  Setting these parameters to 1 within Unified PCIe VIP HDL instance result in the following:

  - Creation of an appropriate interface instance for monitoring purposes.
  - Connection of the monitor interface nodes to link partner or VIP nodes.
  - Invocation of appropriate uvm_config_db call to pass to the newly created monitor interface to passive agent.

Note

- The unified approach is limited in intent to reducing integration and testbench maintenance efforts at signal connectivity level. The testbench is still required to populate the Passive Monitor configuration object appropriately.
- `SVT_PCIE_UI_CONNECT_DOWNSTREAM_PORT_MONITOR` and `SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR` are mutually exclusive and therefore must not be set to '1' within a single unified VIP instance.

10.2.6 Using a Virtual Interface Handle

Each Unified VIP model instance implements a method called `update_if_variables(…)`. The testbench must invoke this method at time 0 (prior to invoking `run_test`). This method copies the module parameter values listed previously into corresponding variables in `svt_pcie_if` instance associated with the Unified VIP module instance. It also invokes `uvm_config_db::set` calls to pass the virtual interface handles for the active component and passive monitor. The testbench can leverage the information in the virtual interface handles to appropriately initialize the `svt_pcie_device_configuration` instance associated with the module instance. You must invoke method `svt_pcie_device_configuration::set_initial_values_via_unified_vif` to pass the handle of the virtual interface to the active VIP component.

- In one of the initial blocks, you must invoke `module_instance.update_if_variables(.....).`
- During the build phase of `uvm_test_top` instance (or its sub-environment)
  - Retrieve virtual `svt_pcie_if` (that is, `svt_pcie_vif`) interface handle by invoking `uvm_config_db::get(....).`
  - Create an instance of `svt_pcie_device_configuration` class and associate this instance with the virtual interface handle retrieved in the previous step by invoking `svt_pcie_device_configuration::set_initial_values_via_unified_vif(....).` For more details, see `svt_pcie_device_configuration` class in the HTML documentation of class reference documentation.
  - Modify the content of `svt_pcie_device_configuration` instance and finally pass it to `svt_pcie_device_agent` instance using `uvm_config_db::set(....)`.

Note

You must not modify property `svt_pcie_device_configuration::model_instance_scope`, its correct value is derived automatically during the execution of method `svt_pcie_device_configuration::set_initial_values_via_unified_vif`.

The inputs to the method `update_if_variables` are as follows:

- bit [3:0] `port_id`: It is used to form a unique string required to store the `svt_pcie_if` virtual handle in `uvm_config_db`.
- bit [7:0] `link_id`: It is used (conditionally) to form a unique string required to store the `svt_pcie_if` virtual handle in `uvm_config_db`. The use of `link_id` value to form a string to pass the handle of active component interface is dependent on argument `use_link_id_prefix_for_active_if_name`. Similarly, use of `link_id` value to form a string to pass the handle of passive component interface is dependent on argument `use_link_id_prefix_for_passive_if_name`
- string `ac_parent_class_hier`: It is used to control the visibility of `svt_pcie_if` virtual handle stored in `uvm_config_db`.

- string `pm_parent_class_hier`: It is used to control the visibility of the monitor interface stored in `uvm_config_db`.
- bit `use_link_id_prefix_for_active_if_name`: If this bit is 1, the `link_id` will be used to create a string used to pass `svt_pcie_if` type handle via `uvm_config_db`:
  - If 1, the string value used will be `link_<link_id as %0d>_vif_< port_id as %0d>`
  - If 0, the string value used will be `vif_< port_id as %0d>`
- bit `use_link_id_prefix_for_passive_if_name`: If this bit is 1, the `link_id` will be used to create a string used to pass passive monitor interfaces via `uvm_config_db`:
  - If 1, the string value used will be
    - `link_<link_id as %0d>_mpipe_x32_8g_vif OR`
    - `link_<link_id as %0d>_spipe_x32_8g_vif  OR`
    - `link_<link_id as %0d>_ serdes_x32_vif`
  - If 0, the string value used will be
    - `mpipe_x32_8g_vif OR`
    - `spipe_x32_8g_vif  OR`
    - `serdes_x32_vif`

Note
The strings mentioned in the last two bullets in the above list may change.

10.2.7 Backward Compatibility With Existing Testbench

The use model and features of the unified testbench setup does not affect the existing test cases and sequences in any way. The proposed changes affect only the connection style between VIP and DUT.

10.2.8 Installing the Unified VIP Example

To install the example, perform the following steps (assuming you have downloaded the PCIe VIP model and set up all licensing and access to the VCS simulator):

1. Install the example. At the command line, invoke the following. The location <*design_dir* > is the location you will install the example.

```
$DESIGNWARE_HOME/bin/dw_vip_setup -path <design dir> -e
pcie_svt/tb_pcie_svt_uvm_unified_vip_sys -svtb
```

2. Change working directory to the installed example:

```
cd <design dir>/>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys
```

3. The README contains a description of the example.
4. To run the example, use the following command:

```
gmake USE_SIMULATOR=vcsvlog base_multi_link_test WAVES=1
```

10.2.9 Testbench Structure

10.2.9.1 Key Files

The following are the key files in the example:

- *top.sv*: Top Level testbench module contains a generic code applicable to all tests.

- *env/pcie_device_base_test.sv*: Extension of `uvm_test`, serves as base class for individual test cases.
- Individual test cases (extension of *pcie_device_base_test.sv*): The same sequence is used to demonstrate flow of packets in different topologies of the testbench.
  - *tests/ts.base_multi_link_test.sv*
  - *tests/ts.base_pipe_test.sv*
  - *tests/ts.base_pma_test.sv*
  - *tests/ts.base_serdes_test.sv*
  - *tests/ts.base_pie8_eq_test.sv*
  - *tests/ts.base_pipe5_test.sv*
  - *tests/ts.base_serdes5_test.sv*

Note
These file names contain the name of the test for the `gmake` invocation. In this case, remove the "*ts.*" and the file extension (*.sv*).

- *hdl_interconnect_macros.sv*: Building blocks of code, used to simplify the integration effort.
- *top_test.sv*: Include file included in the *top.sv*. Contains Test/DUT topology specific code.

10.2.9.2 Topology File Structure

The example testbench demonstrates the Unified PCIe VIP as part of different test topologies. The topology variations are captured in the following files that use macros from the file *hdl_interconnect_macros.sv*:

- *top.pcie_multi_link_topology.sv*
- *top.pcie_pma_topology.sv*
- *top.pcie_serdes_topology.sv*
- *top.pcie_pie8_eq_topology.sv*
- *top.pcie_pipe_topology.sv*
- *top.pcie_pipe5_topology.sv*
- *top.pcie_serdes5_topology.sv*

The *top_test.sv* file is a soft link pointing to one of the topology files. The link is updated by the Makefile based on the test name supplied by the user. These topology files contain the code to instantiate and connect the Unified VIP instances. The DUT instance is also part of the topology file.

10.2.9.3 Helper Macros

The example testbench comes with a set of multi-line macros. These are shared as source code so that testbench can use them as starting point and enhance them. The help macros are in the *tb_pcie_svt_uvm_unified_vip_sys/hdl_interconnect_macros.sv* file.

Table 10-3 lists the available Helper macros.

Table 10-3 Helper Macros

Macro Name	Arguments	Description
SVT_PCIE_ICM_CREATE_PORT_INST	port_id	This macro requires a decimal value as input (port_id) • Creates instance of svt_pcie_if. Uses port_id to generate an unique name for the instance. • Creates instance of PCIe single port device. Uses port_id to generate an unique name for the instance. • Declares set of parameters and ties them to a PCIe single port device instance. Uses port_id to associate the parameter with PCIe single port device instance. This parameter can be updated using a defparam and can also be used in "generate" statements residing in the module instantiating PCIe single port device.
SVT_PCIE_ICM_CREATE_LINK	link_id, spd_a, spd_b	This macro accepts three inputs: • Decimal value as link_id. • Reference to instance of VIP module representing a port of the link. • Reference to instance of another VIP module representing peer port of the link. It invokes the update_if_variables method for both instances of the VIP, and if required also handles VIP application layer connection requirements. The connection requirements take effect only when the VIP has its application layer enabled, but the lower stack of PHY, Data Link, and Transaction Layer is disabled (that is, VIP instance is being used to test RTL controller implementation).
SVT_PCIE_ICM_PIPE_PIPE_LINK	link_id, spipe_inst, mpipe_inst	This macro accepts three inputs: • Decimal value as link_id. • Reference to instance of VIP module representing a SPIPE port of the link. • Reference to instance of another VIP module representing MPIPE port of the link. It is used to connect the pipe signals of two controllers (assumption is each of the pipe controller instances has a member called vip_port_if and this member is of type svt_pcie_if).

Table 10-3 Helper Macros

Macro Name	Arguments	Description
SVT_PCIE_ICM_SER_SER_LINK	link_id, ser_inst_a, ser_inst_b	This macro accepts three inputs: • Decimal value as `link_id`. • Reference to instance of VIP module representing a Serial port of the link. • Reference to instance of another VIP module representing peer Serial port of the link. It is used to connect the serial signals of two controllers (assumption is each of the controller instances has a member called `vip_port_if` and this member is of type `svt_pcie_if`).
SVT_PCIE_ICM_PMA_PMA_LINK	link_id, pma_inst_a, pma_inst_b	This macro accepts three inputs: • Decimal value as `link_id`. • Reference to instance of VIP module representing a PMA port of the link. • Reference to instance of another VIP module representing peer PMA port of the link. It is used to connect the PMA signals of two controllers (assumption is each of the controller instances has a member called `vip_port_if` and this member is of type `svt_pcie_if`).
SVT_PCIE_ICM_PIPE5_PIPE5_LINK	spipe_inst, mpipe_inst	This macro accepts two inputs: • Reference to instance of VIP module representing a SPIPE port of the link. • Reference to instance of another VIP module representing MPIPE port of the link. It is used to connect signals of MAC controller with PIPE5 interface to PIPE5 compliant PHY + MAC.

10.2.9.4 Single/Multi-Link Base Test (pcie_device_base_test.sv)

This is the top level user-defined component in the UVM hierarchy. It is extended from the `uvm_test` class. The existing setup is configured to support up to four (controlled through local macro `SVT_PCIE_MAX_NUM_LINKS`) active links running concurrently.

The code in the build method parses the `config_db` structure looking for reference to the virtual `svt_pcie_if` handles. On successful retrieval of the interface handles of the required type, and adhering to naming convention, it does some basic checks on variables stored as part of these instances. If the checks are successful, then it creates a child component of type `pcie_device_unified_vip_instances_env`. Each such instance represents a link created in the topology file.

The code in `align_vif_and_cfg_prop(....)` ensures the default values and/or constraint limits of `svt_pcie_device_configuration` members are in-line with the compile-time settings listed in topology files.

10.2.9.5 Running Test Cases

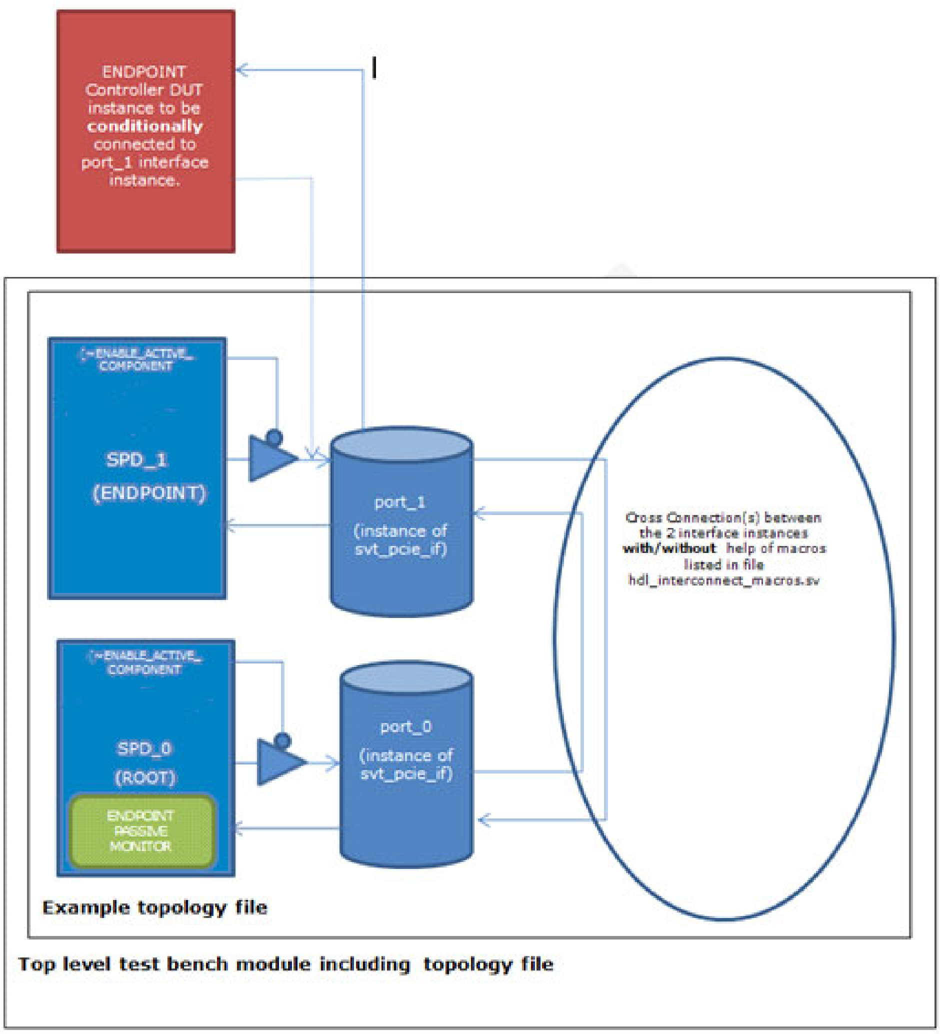
The following are the command line invocation of all the test cases:

- `gmake base_pipe_test`
- `gmake base_pma_test`
- `gmake base_serdes_test`
- `gmake base_multi_link_test`
- `gmake base_pie8_eq_test`
- `gmake base_pipe5_test`
- `gmake base_serdes5_test`

10.2.10 DUT Integration

The reference model shown in this section is a PIPE interface, but the same can be applied to PMA and Serial interfaces.

The following figure represents the content of the *top.pcie_pipe_topology.sv* (single link EP DUT) in graphical form.



If you want to replace EP DUT (SPD_1 `instance`) with the RTL implementation, then perform the following steps:

1. Create a new topology file similar to the one provided in the *example* directory. Instantiate DUT module in the topology file.
2. Ensure that the VIP instance that is being replaced by DUT does not drive the signal interface. To achieve this goal, do not exclude or comment the VIP instance. Instead follow either step 3. or step 4.
3. Disable the Active Component of Unified VIP.

   For example, if you want to disable a VIP instance with instance name as `spd_1`: `defparam SVT_PCIE_UI_CONNECT_ACTIVE_VIP_P1=0.`
4. After modifying the settings as described in step 3., connect DUT ports to the appropriate sub-interface of Unified PCIe VIP interface (`port_if_1`).
5. For the testbench setup with shared signal connections between the SVT PCIe Passive Monitor and Active VIP.

   set `defparam` `SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR_P0=1`.
6. Create extensions of tests that need to run with a DUT as new files under a tests folder. Include these tests in the newly created topology file.
7. Update the build method of the extended test to correctly populate the settings of `cust_cfg[0].endpoint_cfg`. This step is similar to non-unified or legacy testbench.

10.2.10.1 Topology and Interconnect Macros Files

The following figures are taken from *top.pcie_multi_link_topology.sv*.

- Interconnect Macro file – *hdl_interconnect_macros.sv*
  - *hdl_interconnect_macros.sv* provides macro definitions and serves a common point to set key parameters of several VIP ports. Parameters with their default values are shown in the following screenshot. These can be updated on a per VIP instance basis using `defparam` overrides in the topology files.

```
`define SVT_PCIE_ICM_CREATE_PORT_INST(port_num) \
  parameter SVT_PCIE_UI_PCIE_SPEC_VER_P``port_num`` = `SVT_PCIE_UI_PCIE_SPEC_VER_3_0; \
  parameter SVT_PCIE_UI_PIPE_SPEC_VER_P``port_num`` = `SVT_PCIE_UI_PIPE_SPEC_VER_4_3; \
  parameter SVT_PCIE_UI_NUM_PHYSICAL_LANES_P``port_num`` = 32; \
  parameter SVT_PCIE_UI_NUM_PMA_INTERFACE_BITS_P``port_num`` = 10; \
  parameter SVT_PCIE_UI_NUM_PIPE_INTERFACE_BITS_P``port_num`` = 32; \
  parameter SVT_PCIE_UI_HIERARCHY_NUMBER_P``port_num`` = 0; \
  parameter SVT_PCIE_UI_MPIPE_P``port_num`` = 1;  \
  parameter SVT_PCIE_UI_PHY_INTERFACE_TYPE_P``port_num`` = `SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES; \
  parameter SVT_PCIE_UI_DEVICE_IS_ROOT_P``port_num`` = 1;    /*behave as a root complex  */ \
  parameter SVT_PCIE_UI_ENABLE_SHADOW_MEMORY_CHECKING_P``port_num`` = 0;  /*If set, applications will check memory reads against the shadow memory*/ \
  parameter SVT_PCIE_UI_ENABLE_CFG_BLOCK_P``port_num`` = 1; \
  parameter SVT_PCIE_UI_DUT_IN_V2V_CTL_TB_P``port_num`` = 0;  /*Ignore this parameter, will be deprecated in future */ \
  parameter SVT_PCIE_UI_CONNECT_ACTIVE_VIP_P``port_num`` = 1; \
  parameter SVT_PCIE_UI_PIPE_CLK_FROM_MAC_P``port_num`` = 1'b0; \
  parameter SVT_PCIE_UI_CONNECT_DOWNSTREAM_PORT_MONITOR_P``port_num`` = 1'b0; \
  parameter SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR_P``port_num`` = 1'b0; \
  svt_pcie_if port_if_``port_num``(); \
  svt_pcie_single_port_device_agent_hdl spd_``port_num``(port_if_``port_num``); \
`ifdef VCS \
  defparam spd_``port_num``.SVT_PCIE_UI_DISPLAY_NAME = {"spd_``port_num``."}; \
`else \
  defparam spd_``port_num``.SVT_PCIE_UI_DISPLAY_NAME = $sformatf("spd_%0d.",port_num); \
`endif \
  defparam spd_``port_num``.SVT_PCIE_UI_ENABLE_CFG_BLOCK = SVT_PCIE_UI_ENABLE_CFG_BLOCK_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_PCIE_SPEC_VER = SVT_PCIE_UI_PCIE_SPEC_VER_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_PIPE_SPEC_VER = SVT_PCIE_UI_PIPE_SPEC_VER_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_NUM_PHYSICAL_LANES = SVT_PCIE_UI_NUM_PHYSICAL_LANES_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_NUM_PMA_INTERFACE_BITS = SVT_PCIE_UI_NUM_PMA_INTERFACE_BITS_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_NUM_PIPE_INTERFACE_BITS = SVT_PCIE_UI_NUM_PIPE_INTERFACE_BITS_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_HIERARCHY_NUMBER = SVT_PCIE_UI_HIERARCHY_NUMBER_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_MPIPE = SVT_PCIE_UI_MPIPE_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_PHY_INTERFACE_TYPE = SVT_PCIE_UI_PHY_INTERFACE_TYPE_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_DEVICE_IS_ROOT = SVT_PCIE_UI_DEVICE_IS_ROOT_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_ENABLE_SHADOW_MEMORY_CHECKING = SVT_PCIE_UI_ENABLE_SHADOW_MEMORY_CHECKING_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_DUT_IN_V2V_CTL_TB = SVT_PCIE_UI_DUT_IN_V2V_CTL_TB_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_CONNECT_ACTIVE_VIP = SVT_PCIE_UI_CONNECT_ACTIVE_VIP_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_PIPE_CLK_FROM_MAC = SVT_PCIE_UI_PIPE_CLK_FROM_MAC_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_CONNECT_DOWNSTREAM_PORT_MONITOR = SVT_PCIE_UI_CONNECT_DOWNSTREAM_PORT_MONITOR_P``port_num``; \
  defparam spd_``port_num``.SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR = SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR_P``port_num``; \
```

- Macro definition of link creation between two VIP instances.

```
// The following macro takes care of forming a link using 2 instances of interface objects connected to VIP.
// The link is uniquely identified using link_id argument. It also invokes the  uvm_config_db calls
// (via update_if_variables) to pass the virtual interface handles to the class world.
`define SVT_PCIE_ICM_CREATE_LINK(link_id,spd_a,spd_b) \
   tri1 link_``link_id``_clkreq_n; \
   initial begin \
      spd_a.update_if_variables(4'h0,link_id,"uvm_test_top","uvm_test_top"); \
      spd_b.update_if_variables(4'h1,link_id,"uvm_test_top","uvm_test_top"); \
   end \
   always @(*) spd_a.vip_port_if.app_if.reset   = (spd_a.vip_port_if.phy_interface_type == PHY_INTERFACE_TYPE_APP)  ? common_pwr_on_reset : 1'bz; \
   always @(*) spd_b.vip_port_if.app_if.reset   = (spd_b.vip_port_if.phy_interface_type == PHY_INTERFACE_TYPE_APP)  ? common_pwr_on_reset : 1'bz; \
   always @(*) spd_a.vip_port_if.app_if.appl_clk    = (spd_a.vip_port_if.phy_interface_type == PHY_INTERFACE_TYPE_APP)  ? spd_b.vip_port_if.app_if.appl_clk : 1'bz; \
   always @(*) spd_b.vip_port_if.app_if.appl_clk    = (spd_b.vip_port_if.phy_interface_type == PHY_INTERFACE_TYPE_APP)  ? spd_a.vip_port_if.app_if.appl_clk : 1'bz; \
```

- Lane-to-Lane connection for PIPE.

```
// The following macro is used by user to cross connect signals of spipe interface
// with that of a mpipe interface
`define SVT_PCIE_ICM_PIPE_PIPE_LINK(link_id,spipe_inst,mpipe_inst) \
      `SVT_PCIE_ICM_PIPE_PIPE_COMMON_CODE(link_id,spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 0) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 1) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 2) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 3) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 4) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 5) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 6) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 7) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 8) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 9) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 10) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 11) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 12) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 13) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 14) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 15) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 16) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 17) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 18) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 19) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 20) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 21) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 22) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 23) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 24) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 25) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 26) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 27) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 28) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 29) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 30) \
      `SVT_PCIE_ICM_PIPE_PIPE_PER_LANE_CODE(spipe_inst.vip_port_if.pipe_if , mpipe_inst.vip_port_if.pipe_if , 31) \
```

- Lane-to-Lane connection for Serial connection.

```
// The following macro is used by user to cross connect signals of one serdes interface instance
// to another serdes interface instance
`define SVT_PCIE_ICM_SER_SER_LINK(link_id,ser_inst_a,ser_inst_b) \
      assign link_``link_id``_clkreq_n = ser_inst_a.vip_port_if.ser_if.clkreq_n; \
      assign link_``link_id``_clkreq_n = ser_inst_b.vip_port_if.ser_if.clkreq_n; \
      always @(*) begin \
       ser_inst_a.vip_port_if.ser_if.reset = common_pwr_on_reset;  \
       ser_inst_b.vip_port_if.ser_if.reset = common_pwr_on_reset;  \
      end \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 0) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 1) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 2) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 3) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 4) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 5) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 6) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 7) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 8) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 9) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 10) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 11) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 12) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 13) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 14) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 15) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 16) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 17) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 18) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 19) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 20) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 21) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 22) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 23) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 24) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 25) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 26) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 27) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 28) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 29) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 30) \
     `SVT_PCIE_ICM_SER_SER_IF_PER_LANE_CODE(ser_inst_a , ser_inst_b , 31) \
```

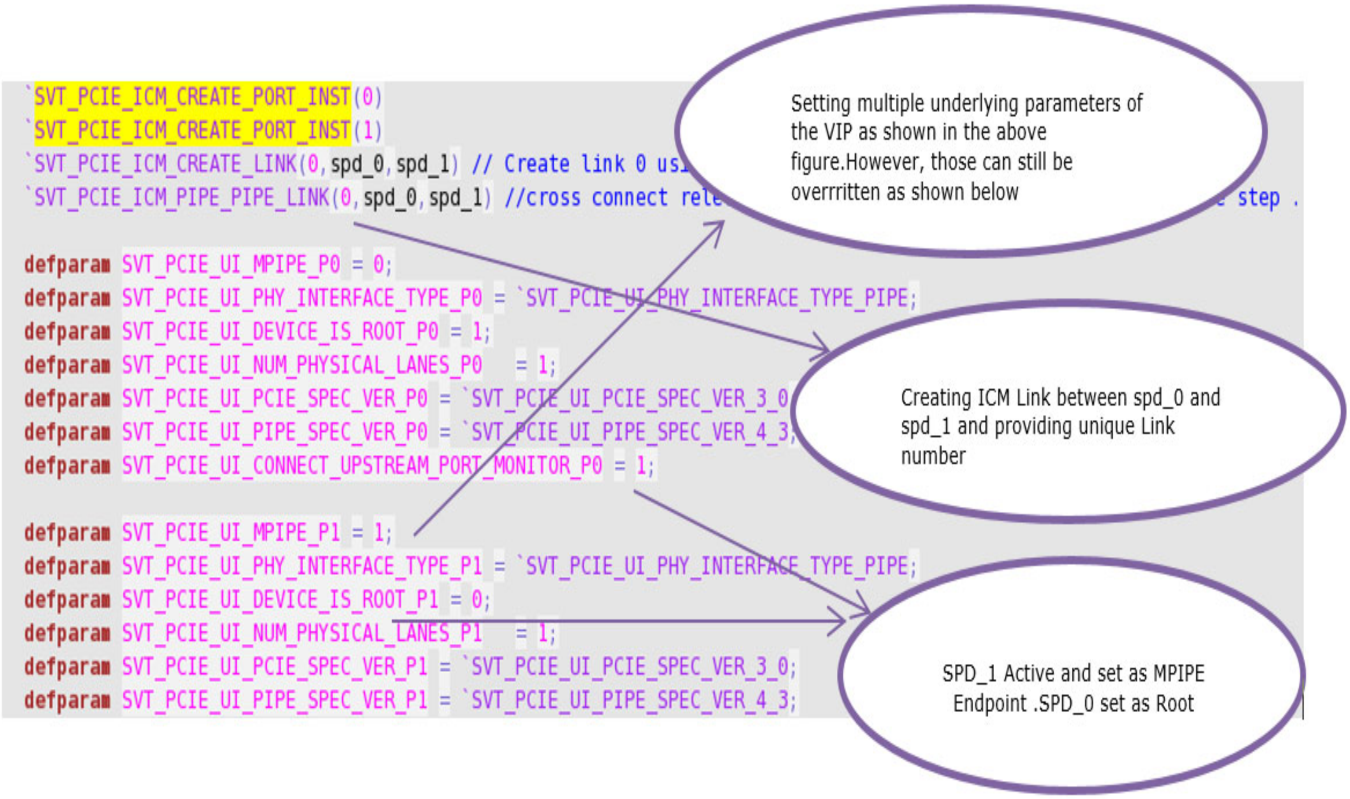
- Lane-to-Lane connection for 10-bit PMA connection.

```
// The following macro is used by user to cross connect signals of one pma interface instance
// with that of another pma interface instance
`define SVT_PCIE_ICM_PMA_PMA_LINK(link_id,pma_inst_a,pma_inst_b) \
      assign link_``link_id``_clkreq_n = pma_inst_a.vip_port_if.pma_if.clkreq_n ; \
      assign link_``link_id``_clkreq_n = pma_inst_b.vip_port_if.pma_if.clkreq_n ; \
      always @(*) begin \
         pma_inst_a.vip_port_if.pma_if.reset = common_pwr_on_reset;  \
         pma_inst_b.vip_port_if.pma_if.reset = common_pwr_on_reset;  \
      end \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 0) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 1) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 2) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 3) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 4) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 5) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 6) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 7) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 8) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 9) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 10) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 11) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 12) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 13) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 14) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 15) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 16) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 17) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 18) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 19) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 20) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 21) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 22) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 23) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 24) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 25) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 26) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 27) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 28) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 29) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 30) \
      `SVT_PCIE_ICM_PMA_PMA_IF_PER_LANE_CODE(pma_inst_a.vip_port_if.pma_if , pma_inst_b.vip_port_if.pma_if , 31) \
```

- Topology File – *top.pcie_multi_link_topology.sv*
  - Connection between `spd_0` and `spd_1` done in the topology file.

Note

Here, `spd_1` can be disabled when integration with a RTL DUT by setting the parameter `SVT_PCIE_UI_CONNECT_ACTIVE_VIP_P1` to 0.



10.2.11 Known Limitations

- VMM is not supported.
- No HTML documentation for new prototype.
- Lack of Makefile infrastructure to support command line controls for individual link properties.
- The following signals are implemented as shared signal even though later PIPE specification versions require it to be on per lane basis (except for PIPE5 signal interconnect)
  - `PowerDown`
  - `Elasticity Buffer Mode`
  - `PclkChangeOk`
  - `PclkChangeAck`
  - `AsyncPowerChangeAck`

Note

- The macros used in file *top.pcie_pipe5_topology.sv* are similar (but independent) to the ones explained in screenshots for *top.pcie_pipe_topology.sv*.
- The macros used in file *top.pcie_serdes5_topology.sv* are exactly the same as the ones used in *top.pcie_serdes_topology.sv.*

10.3 Check ‘x’ or ‘z’ on Signal

This feature checks that the signals at the interface are not driven to 'x' or 'z' state after reset de-assertion. The feature is command line controllable. It is disabled by default, it can be enabled in the Unified TB (`tb_pcie_svt_uvm_unified_vip_sys`) by adding a macro `+define+SVT_ENABLE_XCHECK=1` in the compile flow.

This check is used to check the case when a signal is not connected or if the signal is wrongly driven to tristate.

It is implemented on the principle that once Unified VIP is out of power on reset (issued via "reset" node) it can check all applicable (based on `svt_pcie_device_configuration` content) interface signals for known values.

In case of signal driven to x or z, the error signature will be as follows.

```
[Unknown Value Observed by X Checker instance] offending signal is connected to
interface instance
test_top.spd_1.mpipe.port0.pipe_if.u_xcheck_pipe_if_rx_eq_in_progress_0.unnamed$$_1.
Please refer to instance name for further details.
```

In the above example message, the VIP is complaining about interface signal `pipe_if_rx_eq_in_progress_0`.

- Limitation:
  - This check is not applicable for conventional TBs.
  - It is applicable to UVM and OVM methodologies only.

11 Using the PCIe Verification IP

This chapter discusses the following topics:

- "User-Supplied Memory Model Interface" on page 206
- "External Clocking and Per Lane Clocking for Serial Interface" on page 207
- "Callbacks" on page 209

11.1 SystemVerilog UVM Example Testbenches

This section describes SystemVerilog UVM example testbenches that show general usage for various applications. A summary of the examples is listed in Table 11-1

Table 11-1 SystemVerilog Example Summary

Example Name	Level	Description
`tb_pcie_svt_uvm_unified_vip_sys`	Advanced	The example consists of the following: • A top-level module, which includes tests, instantiates interfaces, HDL interconnect wrapper, generates system clock, and runs the tests • A base test, which is extended to create a directed and a random test • The tests create a testbench environment, which in turn creates PCIe System Env • PCIe System Env is configured with Root Complex and one Endpoint

The examples are located at:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/examples/sverilog/

Examples may be installed in your local design directory following the instructions in the installation chapter.

The tests in the unified example (`tb_pcie_svt_uvm_unified_vip_sys`) are:

- `ts.address_translation_test.sv`
- `ts.base_multi_link_test.sv`
- `ts.base_pie8_eq_test.sv`
- `ts.base_pipe_in_serdes_arch_mode_test.sv`
- `ts.base_pipe5_test.sv`
- `ts.base_pipe_test.sv`
- `ts.base_pma_test.sv`
- `ts.base_serdes5_test.sv`
- `ts.base_serdes_test.sv`
- `ts.basic_ptm_test.sv`

11.2 Installing and Running the Examples

Below are the steps for installing and running example, `tb_pcie_svt_uvm_unified_vip_sys`. Similar steps are applicable for other examples:

1. Install the example using the following command line:

```
% cd <location where example is to be installed>
% mkdir design_dir <provide any name of your choice>
% $DESIGNWARE_HOME/bin/dw_vip_setup -path ./design_dir  -e
    pcie_svt/tb_pcie_svt_uvm_unified_vip_sys -svtb
```

This installs the example under:

<design_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys

2. Use either one of the following to run the testbench:

   a. Use the Makefile:

      The following three tests are provided in the "tests" directory:

      - `ts.address_translation_test.sv`
      - `ts.base_multi_link_test.sv`
      - `ts.base_pie8_eq_test.sv`
      - `ts.base_pipe_in_serdes_arch_mode_test.sv`
      - `ts.base_pipe5_test.sv`
      - `ts.base_pipe_test.sv`
      - `ts.base_pma_test.sv`
      - `ts.base_serdes5_test.sv`
      - `ts.base_serdes_test.sv`
      - `ts.basic_ptm_test.sv`

      To run the ts.directed_test.sv test, for example, do following:

      ```
      gmake USE_SIMULATOR=vcsvlog base_serdes_test WAVES=1
      ```

      To see more options, invoke "`gmake help`".

   b. Use the sim script:

      To run the ts.random_wr_rd_test.sv test, for example, do following:

      ```
      ./run_pcie_svt_uvm_unified_vip_sys -w base_pipe_test vcsvlog
      ```

      To see more options, invoke "`./run_pcie_svt_uvm_unified_vip_sys -help`".

For more details about installing and running the example, refer to the README file in the example, located at:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/examples/sverilog/tb_pcie_svt_uvm_unified_vip_sys/README

or

<design_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/README

11.2.1 Running the Example with +incdir+

In the current setup, you install the VIP under *DESIGNWARE_HOME* followed by creation of a design directory which contains the versioned VIP files. With every newer version of the already installed VIP requires the design directory to be updated. This results in:

- Consumption of additional disk space
- Increased complexity to apply patches

The new alternative approach of directly pulling in all the files from *DESIGNWARE_HOME* eliminates the need for design directory creation. VIP version control is now in the command line invocation.

The following code snippet shows how to run the basic example from a script:

```
cd <testbench_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/

// To run the example using the generated run script with +incdir+

./run_pcie_svt_uvm_unified_vip_sys -verbose -incdir base_pipe_test vcsvlog
```

For example, the following compile log snippet shows the paths and defines set by the new flow to use VIP files right out of *DESIGNWARE_HOME* instead of *design_dir*.

```
vcs -l ./logs/compile.log -q -Mdir=./output/csrc
+define+DESIGNWARE_INCDIR=<DESIGNWARE_HOME> \
+define+SVT_LOADER_UTIL_ENABLE_DWHOME_INCDIRS
+incdir+<DESIGNWARE_HOME>/vip/svt/pcie_svt/O-2018.12/sverilog/include \
-ntb_opts uvm -full64 -sverilog +define+UVM_DISABLE_AUTO_ITEM_RECORDING
+define+UVM_PACKER_MAX_BYTES=8192 \
-debug_acc -timescale=1ns/1fs +libext+.v+.sv -y <testbench_dir>/src/verilog/vcs \
-y <testbench_dir>/src/sverilog/vcs +define+UVM_VERDI_NO_COMPWAVE \
-debug_acc -P pli.tab msglog.o +define+SVT_UVM_TECHNOLOGY +define+SYNOPSYS_SV
+incdir+<testbench_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/. \
+incdir+<testbench_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/../..
/env \
+incdir+<testbench_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/../en
v \
+incdir+<testbench_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/env \
+incdir+<testbench_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/dut \
+incdir+<testbench_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/hdl_i
nterconnect \
+incdir+<testbench_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/lib \
+incdir+<testbench_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys/tests
\
-o ./output/simvcssvlog -f top_files -f hdl_files
```

Note
For VIPs with dependency, include the `+incdir+` for each dependent VIP.

11.2.1.1 Supported Methodologies with Simulators

Table 11-2 lists the methodologies supported with simulators.

Table 11-2 Supported Methodologies with Simulators

Methodology	VCS	MTI	IUS
UVM	Supported	Supported	Not Supported
OVM	Supported	Supported	Not supported
VMM	Supported	Supported	Not supported
HDL	Not Supported	Not Supported	Not Supported

11.3 Error Message Usage

The control of check and error messages from the model is handled through the svt_err_check class instance "err_check" within the agent. class. Following is a message you might see from the model.

```
UVM_ERROR svt_err_check_stats.sv(817) @ 226895734.80 ps:
uvm_test_top.env.root.port0.dl0
[register_fail:AC_DL:PROTOCOL:dl_receive_nullified_tlp_lcrc]  - Received TLP with EDB
delimiter but bad LCRC = 0x8fb52aea, expected LCRC = 0x8fb52ae9
```

The main components of the message are:

- **Reporter**: “uvm_test_top.env.root.port0.dl0”
- **ID**: "register_fail:AC_DL:PROTOCOL:dl_receive_nullified_tlp_lcrc"
- **Message**: “Received TLP with EDB delimiter but bad …”

The check interface gives you the ability to change how the message is issued based on the unique message ID. For example, you can write the following code to demote the ERROR message to a NOTE/UVM_INFO message:

```
env.root.err_check.set_default_fail_effects("^AC_DL", "", svt_err_check_stats::NOTE,
"dl_receive_nullified_tlp_lcrc$");
```

The resulting output is now:

```
UVM_INFO svt_err_check_stats.sv(817) @ 226895734.80 ps: uvm_test_top.env.root.port0.dl0
[dl_receive_nullified_tlp_lcrc]  - Received TLP with EDB delimiter but bad LCRC =
0x8fb52aea, expected LCRC = 0x8fb52ae9
```

Users can change the behavior of the specified message by modifying the "effect" argument (argument three in the set_default_fail_effects() method), which is of type svt_err_check_stats::fail_effect_enum. The values available with the fail_effect_enum are:

- **IGNORE**. Ignore the check result.
- **VERBOSE**. Generate verbose message for the check results.
- **DEBUG**. Generate debug message for the check results.
- **NOTE**. Generate note message for the check results.
- **WARNING**. Generate warning message for the check results.
- **ERROR**. Generate error message for the check results.
- **EXPECTED**. Failure is expected.

Hint The NOTE, DEBUG, and VERBOSE settings equate to UVM_INFO verbosity settings of UVM_LOW, UVM_HIGH, and UVM_FULL respectively.

In addition to changing how messages are reported to you, the svt_err_check_stats class has additional features for you to track messages:

- **exec_count**. Tracks the number of times that a given check has been executed.
- **pass_count**. Tracks the number of times that a given check has PASSED.
- **fail_ignore_count** . Tracks the number of times the check has failed, with IGNORED effect.
- **fail_verbose_count**. Tracks the number of times the check has failed, with VERBOSE effect.

- **fail_debug_count**. Tracks the number of times the check has failed, with DEBUG effect.
- **fail_note_count**. Tracks the number of times the check has failed, with NOTE effect.
- **fail_warn_count**. Tracks the number of times the check has failed, with WARNING effect.
- **fail_err_count**. Tracks the number of times the check has failed, with ERROR or FATAL effect.
- **fail_expected_count**. Tracks the number of times the check has failed, with EXPECTED effect.

To check the statistics count, get a handle to the stats container, using the same lookup strings as with

```
svt_err_check_stats check_stats = env.root.err_check.find("^DL$", "",
"^register_fail:DL:dl_receive_nullified_tlp_lcrc$");
```

The "check_stats.fail_note_count" would now be incremented by 1.

To disable all tracking of an ID (no statistics or coverage collection), which would supersede the above call, do the following:

```
env.root.err_check. disable_checks("^DL$", "",
"^register_fail:DL:dl_receive_nullified_tlp_lcrc$");
```

The "^" and "$" in the SVT call are the regex meta-character start/end string terminators respectively. The string arguments to the SVT err_check interface are regex expressions. Blanks are considered wildcards.

The "" argument in the previous example is the sub_group. The complete ID is register_fail:[<group>:][<sub_group>:]<unique_id>. The sub_group may or may not appear in all cases. You can use the meta-characters when trying to isolate specific groups, sub_groups, and IDs.

11.4 Some Configuration Values to Set

The following steps can help you get started in configuring the model.

1. SetAllocated Credits
   - TL: Set initial credits for a VC
   - Use: svt_pcie_tl_configuration
2. SetVCEnable
   - TL: Enable / Disable VCs; disabled by default
   - Use: svt_pcie_tl_service_set_vc_en_sequence
3. SetTrafficClassMap
   - TL: Setup TC map
   - Use: svt_pcie_tl_configuration
4. AddMemAddrAppIdMapEntry
   - TL: Sets AP ID for Memory target address range
   - Use: svt_pcie_mem_target_service_mem_range_sequence
5. SetLinkEnable
   - DL: Enable / Disable the DL
   - Use: svt_pcie_dl_service_link_en_sequence
6. SetSupportedSpeeds
   - PL: Set supported speed for link training / default speed if link training is disabled

- Use: svt_pcie_pl_phy_configuration

7. SetLinkWidth
   - PL: maximum link width, also initiates LTSSM negotiation
   - Use: svt_pcie_pl_phy_configuration
8. SetLinkEnable
   - PL: Enable / Disable the PL link
   - Use: svt_pcie_pl_service_link_en_sequence

Example configuration code:

```
class pcie_shared_cfg extends uvm_object;
   rand svt_pcie_device_configuration root_cfg;
   …
   function new(string name = "pcie_shared_cfg");
      super.new(name);
      this.root_cfg = new("root_cfg");
       root_cfg.device_is_root = 1;
       root_cfg.pcie_spec_ver   = svt_pcie_device_configuration:PCIE_SPEC_VER_2_1;
       root_cfg.pcie_cfg.pl_cfg.link_width = 1;
       root_cfg.pcie_cfg.pl_cfg.target_speed = `SVT_PCIE_SPEED_2_5_G;
       root_cfg.pcie_cfg.pl_cfg.skip_polling_active = 1;
   endfunction : new
endclass : pcie_shared_cfg
```

Example to set credits.

```
class myTest extends uvm_test;
   …
    virtual function void build_phase(uvm_phase phase);
       super.build_phase(phase);
       // Setup Initial TX Credits
       cust_cfg.root_cfg.pcie_cfg.tl_cfg.init_p_hdr_tx_credits[0] = 104;
       cust_cfg.root_cfg.pcie_cfg.tl_cfg.init_p_data_tx_credits[0] = 1020;
       cust_cfg.root_cfg.pcie_cfg.tl_cfg.init_np_hdr_tx_credits[0] = 105;
       cust_cfg.root_cfg.pcie_cfg.tl_cfg.init_np_data_tx_credits[0] = 1021;
       cust_cfg.root_cfg.pcie_cfg.tl_cfg.init_cpl_hdr_tx_credits[0] = 106;
       cust_cfg.root_cfg.pcie_cfg.tl_cfg.init_cpl_data_tx_credits[0] = 1022;
      …
endclass
```

Example to enable the link:

```
class pcie_traffic_sequence extends svt_pcie_device_system_virtual_base_sequence;
  …
  task body();
      svt_pcie_dl_service_set_link_en_sequence link_en_seq;
      svt_pcie_pl_phy_service_set_phy_en_sequence phy_en_seq;
      …

        `uvm_do_on_with(link_en_seq, p_sequencer.root_virt_seqr.mac_virt_seqr.dl_seqr,
             {link_en_seq.enable == 1'b1;})
…
  endtask : body
endclass : pcie_traffic_sequence
```

11.5 UVM Reporting Levels

The UVM verbosity level for message logging is set using the +UVM_VERBOSITY runtime option. For each verbosity level, the Items that the UVM API prints to the log file are shown below.

- UVM_NONE – Only print error messages
- UVM_LOW – Print important messages that are not error messages
- UVM_MEDIUM – Drivers and monitors print transactions they transmit and receive
- UVM_HIGH – Print more detailed messages about component operation
- UVM_FULL – Print internal debug messages

11.6 Controlling Verbosity From the Command Line

The VIP can use all the UVM command line options for control of verbosity. The following table summarizes the options available. Please refer to the UVM Reference Guide for more details.

UVM Options are shown below.

Table 11-3 UVM Verbosity Options

UVM Option	Description
+UVM_VERBOSITY	setting for all components
+uvm_set_verbosity	Granular control by component and phase or time
+uvm_set_action	Action to take upon message (None, display, log, count, stop, exit, hook)
+uvm_set_severity	Severity override (upgrade or downgrade)

The remainder of this section will focus on verbosity control from the command line.

Globally

The PCIe VIP is compliant with the verbosity options within UVM. The uvm_cmdline_processor looks for the +UVM_VERBOSITY option on the simulator command line, and will set the initial verbosity for all UVM components to the supplied level.

Examples:

```
// Display only UVM_FATAL, UVM_ERROR, UVM_WARNING
simv +UVM_TESTNAME=base_pipe_test +UVM_VERBOSITY=UVM_NONE
// Display all messages
simv +UVM_TESTNAME=base_pipe_test +UVM_VERBOSITY=UVM_FULL
```

Per Component:

Also supported is the +uvm_set_verbosity which allows for more granular control. The command breakdown is as follows:

+uvm_set_verbosity=<component_name>, <id>, <verbosity>, <phase_name>

or

+uvm_set_verbosity=<component_name>, <id>, <verbosity>, time, <time>

Note, <id> can either be all, _ALL_ or a specific message id.

Example:

```
// Set all components to UVM_NONE except for the tl which is at UVM_LOW
simv +UVM_TESTNAME=base_pipe_test +UVM_VERBOSITY=UVM_NONE \
+uvm_set_verbosity=uvm_test_top.env.root.port0.tl0,_ALL_,UVM_LOW,time,0
// Set all components to UVM_NONE except for root
simv +UVM_TESTNAME=base_pipe_test +UVM_VERBOSITY=UVM_NONE \
+uvm_set_verbosity=uvm_test_top.env.root.*",_ALL_,UVM_LOW,time,0
```

Another option for component control which applies only to SVT based VIPs is the +vip_verbosity option:

Example:

```
// UVM_NONE for all with the exception that all instances of the dl are at UVM_LOW and
// all instances of the tl are at UVM_FULL
simv +vip_verbosity=svt_pcie_dl:UVM_LOW,svt_pcie_tl:UVM_FULL
```

SVT Verbosity relationship to UVM Severity Levels

`define SVT_FATAL_VERBOSITY UVM_NONE

`define SVT_ERROR_VERBOSITY UVM_NONE

`define SVT_WARNING_VERBOSITY UVM_NONE

`define SVT_NORMAL_VERBOSITY UVM_LOW

`define SVT_TRACE_VERBOSITY UVM_MEDIUM

`define SVT_DEBUG_VERBOSITY UVM_HIGH

`define SVT_VERBOSE_VERBOSITY UVM_FULL

11.7 Resetting the PCIe VIP

Power-on reset to the VIP model uses the reset signal which is active high. Reset signal must be de-asserted at time 0 such that a posedge is seen by the VIP. Reset should be asserted for at least 100 ns. 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.

When VIP model is in SPIPE mode, the `attached_pipe_reset_n` signal needs to be de-asserted (0->1) after the above power-on reset sequence has occurred. Also, `pipe_pclk` signal has to be stable (that is, toggling) before both reset and `attached_pipe_reset_n` signals are de-asserted.

To perform a reset of the DUT in mid-simulation, only the DUT should be reset. The VIP will detect the change on the bus and move to Detect. Training will resume and the bus will recover. Thus, for proper DUT verification, it is advisable to separate the SVC model reset from the DUT reset.

In a typical scenario, the following will happen:

1. Initialize and bring up link to stable after reset (normal config sequence)
2. Run traffic
3. Take down the link and check both IP/VIP are in link-down state
4. Bring link-up and check both IP/VIP are in a link-up state
5. Run traffic and verify all is well.

The svt_pcie_device_virtual_reset_sequence class will perform a mid simulation reset as described in the previous steps. This sequence implements Reset. This class resets the VIP by doing the following:

- Sets the hotplug mode to unplugged in the PL
- Calls RESET_APP on all of the applications.

Note that in order to start up the LTSSM again the user will have to use a PL service call to set the hotplug mode to HOTPLUG_DETECT

The model reset supports the following actions:

- Clears all of the Tx and Rx packet queues in the Transaction Layer. This deletes any completions in progress.
- Clears all packets queued in the driver_app, requester_app, target_app, and svt_pcie_tlp.
- All storage elements are cleared or garbage collected as appropriate.
- Deletes all packets that are in transit (for example, in the link layer or in the phy layer).
- Kills all completion timeouts in transaction layer
- Terminates compliance checks
- Resets the LTSSM back to its default initial state (Detect).

Note
The test needs to wait for the VIP to be in an idle state (that is, no PIPE signal handshaking in progress) when changing VIP configuration during UNPLUG. The test can use the `is_pipe_idle` attribute in the `svt_pcie_pl_status` class to determine when the VIP is idle.

11.8 Creating and Using Custom Applications

Custom applications and test sequences can be developed for the PCIe VIP analogous to that of the Driver, Requester, and Target. You can create applications that enable specific functionality not available through the built-in applications. Custom applications allow the user to interface to the TL with TLPs enabling end-user specific functionality.

Applications examples include SRIOV and address translation. User applications are instantiated as classes in the SVT testbench. User applications that send TLPs should communicate with the VIP's tlp_seqr TLP sequencer using a TLP sequence that generates TLPs from svt_pcie_agent::tlp_seqr.

User applications co-exist and run in parallel to the Synopsys-supplied applications. Alternatively, testbenches can emulate user applications using sequences that generate TLPs with unique application IDs. The sequences that generate these TLPs run on the svt_pcie_agent::tlp_seqr sequencer.

The application_id attribute of the TLP objects generated by that application is identified by this value. The application id is available in svt_pcie_tlp::application_id. Application IDs in the range 0 – 19h are reserved for VIP internal use. The testbench should ensure that the application_id used by the applications are unique.

11.8.1 Setting Up Application ID Maps

For traffic to be directed between the MAC and the user application, a routing map must be set up. In particular, for the TLP routing to work, application IDs must be mapped to specific memory/IO address ranges, message codes and so on. This can be accomplished using the svt_pcie_tl_service transactions. A sequence of this type will run on the tl_seqr of the device agent, as shown in Example 11-1. Alternatively, the application_id to requester ID map is set up automatically by the VIP whenever a TLP with a specific RID and application ID is sent for the first time.

The following service transaction types can be used to set up application id maps:

ADD_MEM_ADDR_APPL_ID_MAP_ENTRY, ADD_IO_ADDR_APPL_ID_MAP_ENTRY

ADD_AT_ADDR_APPL_ID_MAP_ENTRY, ADD_RID_MSG_CODE_APPL_ID_MAP_ENTRY

ADD_RID_APPL_ID_MAP_ENTRY, ADD_CFG_BDF_APPL_ID_MAP_ENTRY

SEE the HTML reference documentation for more information on these service types.

Example 11-1

```
svt_pcie_tl_service add_mem_add_req;
`uvm_do_with(add_mem_add_req,{service_type ==
    svt_pcie_tl_service::ADD_MEM_ADDR_APPL_ID_MAP_ENTRY;
    appl_id == 32’h21;
    memory_addr == 0;
    memory_window == 32'h1000;})
```

11.8.2 Using Testbench Sequences to Emulate User Applications

Testbenches can add sequences that generate TLPs with unique application IDs to emulate user applications. A sequence is created that generates TLPs with unique application IDs and is run on the tlp_seqr of the PCIe agent contained in the PCI device agent. This sequencer can be accessed via the top-level virtual sequencer of type svt_pcie_device_system_virtual_sequencer:

```
`uvm_do_on(tlp_directed_seq, p_sequencer.root_virt_seqr.pcie_virt_seqr.tlp_seqr);
```

The output TLM port of this sequencer is internally connected to the sequence_item_port of the VIP’s transaction layer.

The TLPs are set up with the appropriate application ID and requester ID and pushed into the seq_item_port, as indicated in Example 11-2.

Example 11-2

```
svt_pcie_tlp mem_rd_request
`uvm_create(mem_rd_request);
mem_rd_request.cfg = cfg;
mem_rd_request.tlp_type = svt_pcie_tlp::MEM_REQ;
mem_rd_request.fmt = svt_pcie_tlp::NO_DATA_3_DWORD;
mem_rd_request.length = 2 + i;
mem_rd_request.ep = 0;
mem_rd_request.at = svt_pcie_tlp::UNTRANSLATED;
mem_rd_request.first_dw_be = 4'b1111;
mem_rd_request.last_dw_be = 4'b1111;
mem_rd_request.application_id = 32‘h21;
mem_rd_request.address = 32'h0000_4000 | ('h100 << i);
mem_rd_request.requester_id = 4;
`uvm_send(mem_rd_request)
```

11.8.3 Waiting for Completions

Completions are routed to the appropriate application_id using the application ID-to-requester ID map. Using code like the following the completions can be accessed from the rx_tlp_peek port whenever they are available:

```
`uvm_send(mem_rd_request)
root.tl.EVENT_RECEIVED_TLP.wait_trigger();
```

```
root.tl.rx_tlp_peek_port.peek(resp);
```

11.9 Backdoor Access to 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 allocates a Configuration Pointer Table which contains pointers to all of the Configuration Blocks allocated (one for each function in the device).

11.9.1 Setting up the Configuration Space for Backdoor Access

The VIP model behavior is not defined by the configuration space as in a real device. The model behavior is defined by the attributes in the configuration and service classes. Though setting up the configuration space does not define the VIP behavior, the model can be set up to respond to any incoming configuration TLP. A user can program the configuration space of the model though the backdoor using the APIs on the cfg_database of the VIP model.

The VIP does not have a real configuration space like a RTL module. However, it has an internal memory that it uses for CfgWr/Rd transactions. The VIP stores the write value for the incoming CfgWr transactions, and use the return data from this memory while completing CfgRd TLPs.

There is a backdoor way to write/read this internal memory used by the VIP for configuration TLPs. Please note that you do not have to pre-load the configuration database to be able to use it. The VIP will respond to any configuration request, but it will have a default value of 0 in all of the registers.

To set up the configuration space in the pcie vip model without having to perform configuration write/read cycles, use the svt_pcie_cfg_database_service class. This class contains these 3 service types:

- GET_NUM_FUNCTIONS(`SVT_PCIE_CFG_DB_SERVICE_GET_NUM_FUNCTIONS) Get maximum number of functions.
- READ_CFG_DWORD(`SVT_PCIE_CFG_DB_SERVICE_READ_CFG_DWORD) Read configuration DWORD.
- WRITE_CFG_DWORD(`SVT_PCIE_CFG_DB_SERVICE_WRITE_CFG_DWORD) Write configuration DWORD.

These services also use these attributes to set up the configuration space.

- dword_addr
- dword_data
- function_num

Following shows example code on backdoor access.

```
//  This sequence creates backdoor then frontdoor (TLP) traffic sequences
class pcie_cfg_seq extends pcie_device_system_test_base_sequence;

  // Factory Registration.
  `svt_uvm_object_utils(pcie_cfg_seq)

  // Constructs the pcie_cfg_seq sequence
  //  @param name string to name the instance.
```

```
  function new(string name = "pcie_cfg_seq");
    int err_status;
    super.new(name);

  endfunction

  // Executes PCIE configuration sequences to demonstrate backdoor/frontdoor accesses

  task body();
    begin
      pcie_device_system_link_up_sequence link_up_seq;
      svt_pcie_driver_app_service_wait_until_idle_sequence wait_until_driver_idle_seq;

      cfg_read_sequence read_cfg_seq;
      //cfg_write_sequence write_cfg_seq;

       svt_pcie_cfg_database_service cfg_database_seq;
      svt_pcie_device_agent endpoint_device;
      bit [7:0]  bus, func;
      bit [15:0] remote_bdf;
      bit [31:0] cfg_rdata, cfg_wdata;

      bit [7:0]  function_num;             // Function Numberb
      bit [31:0]  cpt_ptr;                 // Configuration Pointer Table, CPT, which
                                           // contains pointers to all the
                                           //  Configuration Blocks allocated (one per
                                           // device)
      bit        cfg_space_type = 0;       // Type 0 or Type 1
      bit [3:0]  function_type = 0 ;       // PF, BF, VF, etc.
      bit [7:0]  sriov_physical_function = 0;  // The PF that is the parent Physical
                                               // function of the VF
      bit [7:0]  mriov_base_function = 0;   // The BF that is the parent Base Function
                                            // of this function
      bit [31:0] command_status;            // Returned status for allocate
      bit [15:0] req_cap_id;
      int         err_status;
      int         remote_register_num;

      int         test_data, test_addr;

      super.body();

      test_data = 32'habcd_1234;
      test_addr = $urandom_range(0, 1023);    // Scribble randomly in the cfg database

      // Housekeeping:
      // bring up link
      `svt_uvm_do(link_up_seq);

      // Can we get the BDF via these xx_device agents?  Maybe in an
      // enumerator that's considered 'cheating'?
      if(!$cast(root_device, p_sequencer.find_root_agent(this))) begin
        `uvm_fatal(get_full_name(), "Failed attempting to obtain handle to Root Device
agent.");
```

```
      end

      if(!$cast(endpoint_device, p_sequencer.find_endpoint_agent(this))) begin
        `uvm_fatal(get_full_name(), "Failed attempting to obtain handle to Endpoint
Device agent.");
      end

      // Backdoor fill in the cfg database
`define BACKDOOR_CFG_ACCESS_WORKS    1
`ifdef BACKDOOR_CFG_ACCESS_WORKS    // currently it doesn't...

      `svt_note("body", "Backdoor write started");

      function_num = 0;
      `uvm_create_on(cfg_database_seq,
                     p_sequencer.endpoint_virt_seqr.cfg_database_seqr);
       cfg_database_seq.service_type = svt_pcie_cfg_database_service::WRITE_CFG_DWORD;
       cfg_database_seq.function_num = function_num;
       cfg_database_seq.dword_addr = test_addr;
       cfg_database_seq.byte_enables = 4'b1111;
       cfg_database_seq.dword_data = test_data;   // 32'habcd_1234
       `uvm_send(cfg_database_seq);

       if(cfg_database_seq.command_status != `SVT_PCIE_CFG_DATABASE_STATUS_SUCCESSFUL)
         `uvm_error("body", $sformatf("Command status not SUCCESSFUL(0x%h) received
          0x%h", `SVT_PCIE_CFG_DATABASE_STATUS_SUCCESSFUL,
           cfg_database_seq.command_status))
       `uvm_info("body",$sformatf("Config_reg 0 is backdoor written with 0x%x",
            cfg_database_seq.dword_data), UVM_LOW);

      `uvm_create_on(cfg_database_seq,
           p_sequencer.endpoint_virt_seqr.cfg_database_seqr);
       cfg_database_seq.service_type = svt_pcie_cfg_database_service::READ_CFG_DWORD;
       cfg_database_seq.function_num = function_num;
       cfg_database_seq.dword_addr = test_addr;
       cfg_database_seq.byte_enables = 4'b1111;
       cfg_database_seq.dword_data = 32'hffff_ffff;
       `uvm_send(cfg_database_seq);
       if(cfg_database_seq.command_status != `SVT_PCIE_CFG_DATABASE_STATUS_SUCCESSFUL)
         `uvm_error("body", $sformatf("Command status not SUCCESSFUL(0x%h) received
          0x%h", `SVT_PCIE_CFG_DATABASE_STATUS_SUCCESSFUL,
          cfg_database_seq.command_status))
       `uvm_info("body",$sformatf("Config_reg 0 is backdoor read data=0x%x",
       cfg_database_seq.dword_data), UVM_LOW);

       if (cfg_database_seq.dword_data != test_data)
         begin
           `svt_error("body", $sformatf("Backdoor read of cfg addr %0d returned 0x%x,
              expected 0x%x", test_addr, cfg_database_seq.dword_data, test_data));
         end
 `endif // BACKDOOR_CFG_ACCESS_WORKS
```

The svt_pcie_* sequences that refers to the configuration space register number. The numbering is with regard to a dword address:

- SVT register number = 0 => [Device ID | Vendor ID] => PCIE registers [[3,2],[1,0]]
- SVT register number = 1 => [Status | Command] => PCIE registers [[7,6],[5,4]]

Note, most Firmware uses as a convention register numbers 0, 1, 2, 3, 4, 5, 6, and 7 as byte offset numbers as defined in the specification.

11.10 Setting VIP Lanes for Receiver Detect

To set which lanes the VIP will see as present from the DUT when the VIP performs a receiver detect in the detect.active state, use the following configuration member:

svt_pcie_pl_configuration::dut_receiver_present = 32'hffff_ffff

Each bit corresponds to a lane, with bit 0 corresponding to lane 0, and with bit 1 corresponding to lane 1, and so on.

For example, take the case of the VIP as an SPIPE configured to a width of x4. If you set dut_receiver_present to 32'h000_0007, then the VIP will behave as if it only detected a receiver on lanes 0-2. As a result of this, the VIP will try to negotiate to a width of x2. Note that this controls what lanes the VIP sees as present, not which lanes the DUT sees as present. Use dut_receiver_present for serial and SPIPE models only. MPIPE models will use the mechanism defined in the PIPE interface to determine which receivers are present.

11.11 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.

11.11.1 Transaction Layer ASCII Signals

The ASCII signals listed in Table 11-4 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 11-4 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_xId	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_xId	Sent TLP transaction ID.

11.11.2 Data Link Layer ASCII Signals

ASCII signals on the Data Link Layer are listed in Table 11-5.

Table 11-5 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

11.11.3 Physical Layer ASCII Signals

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

Table 11-6 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 11-6 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.

11.12 Using the Ordering Application

This component is provided as an optional feature which may be utilized by testbenches explicitly. The Ordering Application is implemented as a uvm_component. The agent components that ship with the VIP do not use this component by default. It has following functions:

- Validates ordering rules implementation of a DUT.
- It can optionally also be used to re-order outbound TLPs to act as application layer logic for a DUT that does not implement the rules in RTL.

It has following parts:

1. svt_pcie_ordering_app_configuration: This configuration class is used to configure the application component.
2. tx_tlp_in_port: This is a sequence item pull port (SIPP) that processes all transactions that are scheduled for transmission by DUT. The application expects transactions of type svt_pcie_tlp and hence it may be connected to a sequencer of type svt_pcie_tlp_sequencer.
3. rx_tlp_in_export: This is an analysis implementation port. The testbench must connect this to an analysis port that broadcasts transactions received by the VIP.
4. tx_tlp_out_port: This is a TLM put port onto which the application pushes all re-ordered TLPs (if enabled).
5. tl_status: This is a reference to the TL status the VIP maintains.

11.12.1 Steps to Use the Ordering Application:

1. Create a new configuration object of type svt_pcie_ordering_app_configuration and set the desired values to the properties.

   ```
   svt_pcie_ordering_app_configuration  ordering_app_cfg = new();
   ```
2. Create the Ordering application in build_phase() of a containing component (typically an environment or test component).

   ```
   ordering_app = svt_pcie_ordering_app::type_id::create("ordering_app", this);
   ```
3. In the build phase, set the reference of the configuration object in the uvm_config_db so it can be obtained by the application.

```
uvm_config_db#( svt_pcie_ordering_app_configuration )::set(this, "ordering_app", "cfg",
ordering_app_cfg);
```

4. Create a new sequencer instance of type svt_pcie_tlp_sequencer.

```
uvm_config_db #( svt_pcie_tl_configuration )::set( this, "ordering_app_seqr", "cfg",
dut_cfg.pcie_cfg.tl_cfg );

ordering_app_seqr = svt_pcie_tlp_sequencer::type_id::create("ordering_app_seqr", this);
```

5. In the connect_phase(), connect the ordering app with the sequencer created in step 4.

```
ordering_app.tx_tlp_in_port.connect(ordering_app_seqr.seq_item_export);
```

6. Set the reference of the svt_pcie_tl_status object (that the VIP maintains) in the uvm_config_db so it can be obtained by the application in end_of_elaboration_phase(). The application has need of this to obtain TCVC mapping and credit information.

```
uvm_config_db#( svt_pcie_tl_status)::set(this, "ordering_app", "tl_status",
<vip_agent>.pcie_agent.pcie_status.tl_status);
```

7. Connect rx_tlp_in_export port with an analysis port that broadcasts TLPs received by the VIP.

```
<analysis_port>.connect(ordering_app.rx_tlp_in_export);
```

8. Optionally connect tx_tlp_out_port with TLM blocking_put_imp of a downstream component. This will typically be responsible to send outbound transactions through DUT's application interface. This step is required if re-ordering of TLPs is enabled in the application (in case DUT does not support the Ordering rules in RTL).

```
ordering_app.tx_tlp_out_port.connect(<dut_driver_component>.<name_of_put_imp_port>);
```

9. Now, use the ordering_app_seqr created in step 4 to schedule all transactions to be transmitted by DUT.

11.13 Using the reconfigure_via_task Call

Please note, the reconfigure_via_task and the get_cfg_via_task calls have been depreciated. Use get_cfg and reconfigure functions in their place.

11.14 Configuring Trace File Output

You can configure how the trace file displays information using the svt_pcie_configuration::dl_trace_options member. These options apply to the default DL tracing format options.

- dl_trace_options[1] bit when set to 1'b1 enables optional printing of Cfg TLP Register Offset address as "O:0x???"
- dl_trace_options[1] bit when set to 1'b0 enables default printing of Cfg TLP Register Number as "R:0x???"
- dl_trace_options[0] bit when set to 1'b1 enables default printing of Cfg Access TLP Payload in Little Endian format
- dl_trace_options[0] bit when set to 1'b0 enables default printing of Cfg Access TLP Payload in Big Endian format

This first instance has the Cfg TLP Register Offset address option enabled (svt_pcie_configuration::dl_trace_options[1]=1) and printing of the CFG access payload as bigendian(svt_pcie_configuration::dl_trace_options[0]=0). See the following:

```
endpoint0  18128.000 18236.000 R CfgWr0  0x0000/06 ... BDF:0x0107 O:0x010   0 c      1 H44008001 ...
```

```
                                                                            0 fecacefa
endpoint0  18448.000 18540.000 R CfgRd0  0x0000/07 ... BDF:0x0107 O:0x010   0 f      1 H04008001 ...
endpoint0  18588.000 18696.000 T CplD    0x0000/07 ... ID:0x0107 Stat:SC  BC:0004    1 H4a008001 ...
                                                                            0 efbecefa
```

This second instance has the Cfg TLP Register Offset address option disabled (svt_pcie_configuration::dl_trace_options[1]=0) and printing of the CFG access payload as LittleEndian(svt_pcie_configuration::dl_trace_options[0]=1). See the following:

```
root0 18124.000 18232.000 T CfgWr0    0x0000/06 ... BDF:0x0107 R:0x004   0 c     1 H44008001 H0000060c ...
                                                                             0  facecafe LittleEndian
root0 18444.000 18536.000 T CfgRd0    0x0000/07 ... BDF:0x0107 R:0x004   0 f     1 H04008001 H0000070f ...
root0 18592.000 18700.000 R CplD      0x0000/07 ... ID:0x0107 Stat:SC  BC:0004   1 H4a008001 H01070004 ...
                                                                             0  facebeef LittleEndian
```

11.15 Target Application

The Target Completer Application provides the following features:

- Provides completer services by responding to various inbound requests: CFG, Memory, IO and Interrupts
- Will break up reads into multiple completions
- Interleaved with other read completions
- Highly configurable
    - min:max read data size
    - Completion boundary (align)
    - Max payload size
    - min:max latency (mem, io, cfg ; all independent)
    - Un-Initialized mem mode: 0, completer abort

To configure the Target Completer Application you use the svt_pcie_target_app_configuration class. Consult the HTML class reference for additional information. Following are some useful settings:

- completer_id. Default Completer ID used by the Target application in the generated completions until Configuration Write reuqests are received on the link to program the completer ID. This ID is concatenation of Bus number, Device number and a Function number.
- max_io_cpl_latency. The variable represents maximum latency in ns for each completion packet generated by the application in response to inbound IO requests.
- min_cfg_cpl_latency. The variable represents minimum latency in ns for the completion packet generated by the application in response to inbound Configuration requests.
- read_completion_boundary_in_bytes. The variable read_completion_boundary_in_bytes specifies the RCB value. The Target application uses this while creating completions.
- max_payload_size_in_bytes. The variable specifies maximum payload size in bytes. Any TLP payload cannot exceed this value in size.

11.15.1 Target Application Callbacks

The target application is the component responsible for handling the auto-generated completions in the VIP model. The model has two callbacks defined at this application layer namely:

1. post_rx_tlp_get(): Called by the component after recognizing a TLP transaction received immediately from the link.

2. pre_tx_tlp_put(): Called by the component after scheduling a TLP transaction for transmission on the link, just prior to framing.

These callback can be used to inject errors into transactions using exception objects. The following example illustrates how to set the Error Poison (EP) bit in a completion TLP generated by the target application.

```
// Callback Class
class set_ep_target_app_callback extends svt_pcie_target_app_callback;

  `svt_uvm_object_utils(set_ep_target_app_callback)

  // Exception List and Exception class objects
  svt_pcie_tlp_exception_list my_tlp_exc_list = new("my_tlp_exc_list");
  svt_pcie_tlp_transaction_exception my_tlp_exc = new("my_tlp_exc");

  function new(string name = "set_ep_target_app_callback");
    super.new();
  endfunction

// Callback Function Implementation
virtual function void pre_tx_tlp_put(svt_pcie_target_app target_app, svt_pcie_tlp
transaction, ref bit drop);
 // Add any conditional statement here to look for a specific TL packet (if necessary).
 // The illustration below is unconditional so it would end up setting the EP bit on all
 // completion packets being transmitted by the target application.
    my_tlp_exc.error_kind = svt_pcie_tlp_transaction_exception::CORRUPT_EP;
    my_tlp_exc.corrupted_data = 0;
    my_tlp_exc.corrupted_data[0] = 1;
    my_tlp_exc_list.add_exception(my_tlp_exc);
    `svt_note("pre_tx_tlp_get",$sformatf("ERP - pre_tx_tlp_put: Attaching exception
      list - corrupting TLP EP field (was=1'b%b now=1'b%b).\n", transaction.ep,
    my_tlp_exc.corrupted_data[0]));
    $cast(transaction.exception_list, my_tlp_exc_list.`SVT_DATA_COPY());
  endfunction

endclass

// UVM Test Class
class base_pipe_test extends pcie_device_base_test ;

  set_ep_target_app_callback set_ep_target_cb;
…
  virtual function void end_of_elaboration_phase(uvm_phase phase);
    super.end_of_elaboration_phase(phase);

    set_ep_target_cb = new("set_ep_target_cb");

uvm_callbacks#(svt_pcie_target_app,svt_pcie_target_app_callback)::add(env.endpoint.targ
et[0], target_cb);
  endfunction

endclass
```

The same approach can be used to attach other errors on completions generated by the target application. To check the list of errors supported by the model, see HTML class reference documentation:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_tlp_exception.html#item_error_kind_enum

11.16 Requester Application

Requester Application is used for generating background traffic. The Application generates PCIe Read/Write transactions to a given target. You can randomize the following:

- [minimum:maximum] address range(s)
- Number or writes
- Number of reads
- [minimum:maximum] data length
- Configure bandwidth
- Time between packets
- # requests per second

It performs a synchronize when it is finished.

You use the class svt_pcie_requester_app_configuration for the Target Application. For additional information on configuration members, consult the HTML Class Reference. Following are some configuration members.

- bandwidth_mode. Specifies the mode which controls the read/write request generation mechanism. BANDWIDTH_MODE_REQUESTS_PER_SEC mode specifies the read/write request generation rate as number of requests to be generated per second. BANDWIDTH_MODE_NS_DELAY_IN_REQUESTS mode specifies the read/write request generation rate in terms of delay between successive requests generated.
- completion_timeout_ns. Completion timeout in nanoseconds.
- num_mem_read. Number of memory read transactions to be transmitted.
- max_time_between_packets. The variable is applicable when bandwidth_mode is set to BANDWIDTH_MODE_NS_DELAY_IN_REQUESTS. The value of this variable specifies maximum delay in NS in the successive packets to be generated.

11.17 What Are Blocking and Non-blocking Reads in PCIe SVT?

'Blocking' read prevents other transactions from being queued. Whereas in case of non-blocking read, a process does not wait for the completion. As a result the transaction is queued.

The driver application layer uses the block attribute to control when the driver queues the next transaction. When it is set to 1, the driver will wait until the transaction is completed before the next transaction is queued.

When set to 0, the driver does not wait for transaction to complete and drives the sub-sequent transaction. The function of the block bit is to not return control to the user until the completion is received in the case of a read.

```
`* block = 1, block until completion is received.
* block = 0, non-block.
For cfg_rd request:
```

```
            `uvm_do_on_with(cfg_rd_rq,p_sequencer.root_virt_seqr.driver_transaction_seq
r[0],{
            bdf == 16'h0100;
            block == 1'b1; // block until completion is received.
            first_dw_be == 4'hf;
            register_number inside {[0:50]};
            });
```

For Mem_rd request:

```
`uvm_do_on_with(mem_rd_request_seq,vip_seqr.driver_transaction_seqr[0],{
address == mem_wr_request_seq.address;
traffic_class == mem_wr_request_seq.traffic_class;
length == mem_wr_request_seq.length;
block == 1'b0; // it will not wait for completion
first_dw_be == mem_wr_request_seq.first_dw_be;
last_dw_be == mem_wr_request_seq.last_dw_be;
});
```

11.18 Using Service Class Reset App

The VC VIP provides functionality to support mid-simulation reset. The scenario is that the VIP is connected to the DUT, and that the DUT is reset sometime into the test after the initial reset of the VIP and DUT. The purpose of the test is to ensure the DUT recovers and that the link retrains. During this time you want to mimic that a reset also happening on the VIP. This means all activity should cease, all buffers should be cleared, and you should go back into our initial state waiting for training sets.

In terms of implementation note that the VIP has its own reset, and that it is only allowed to toggle once, typically at the beginning of a simulation. Further, the VIP reset should never be connected to the reset of the DUT.

Since the VIP reset doesn't toggle, a mid-sim reset is performed with a combination of hot plug control and application resets. The PL provides a mechanism to 'unplug' and then 'plug' the svt_pcie_pl_service_request_hot_plug_mode_sequence. All applications provide a mechanism which resets the apps meaning they are re-initialized, svt_pcie_*_reset_app_sequence. Table 11-7 on page 193 lists each one for each service class.

Synopsys also provides a sequence which wraps the hot plug and reset calls: svt_pcie_device_virtual_reset_sequence.

The following outlines the steps in a mid-sim reset.

1. Unplug VIP from bus (HOTPLUG_UNPLUG)
2. Assert Reset on the DUT
3. Reset apps (all, some, or none – user choice)
4. Re-enable the VIP on the bus (HOTPLUG_DETECT)
5. De-assert Reset on the DUT
6. Continue with the test
   - Suggest monitoring for a change in LTSSM state; PL : WaitForLtssmStateChange()
7. Initialize DUT and VIP, run to a point in the test where a mid-sim reset is to be performed

Option 1:

- Unplug VIP from bus, svt_pcie_pl_service_request_hot_plug_mode_sequence with HOTPLUG_UNPLUG
- Assert reset on DUT
- foreach (app) ResetApp (call on apps to reset; not necessary to reset all apps)
- execute svt_pcie_pl_service_request_hot_plug_mode_sequence with HOTPLUG_DETECT
- De-assert reset on DUT

Option 2:

- In parallel, assert DUT's reset line and execute: svt_pcie_device_virtual_reset_sequence
- After completion of sequence, reassert DUT's reset line

8. Continue with the test
   - Suggest monitoring for a change in LTSSM state, or L0

```
wait (agent.status.pcie_status.pl_status.ltssm_state == svt_pcie_types::L0)
```

Table 11-7 Service Class App Sequence Resets

Reset Sequence for Application	Description
svt_pcie_requester_app_service_reset_app_sequence	• This sequence implements ResetApp • RESET_APP is a reset for the Driver • The effect of resetting this application is to drop all queued and partially completed requests. After a reset the driver will check for completions on sent requests or check for timeouts. • It is not necessary to call Reset App at the beginning of simulation
svt_pcie_io_target_service_reset_app_sequence	• This sequence implements Reset App • RESET_APP resets the application back to its initial state. All data will be lost. • It is not necessary to call this at start of sim.
svt_pcie_mem_target_service_reset_app_sequence	• This sequence implements Reset App • RESET_APP resets the memory target back to its initial state. All data will be lost. • It is not necessary to call this at start of sim
svt_pcie_requester_app_service_reset_app_sequence	• This sequence implements Reset App • RESET_APP is a reset for the requester. Any outstanding requests will be dropped, and there will be no timeouts for these dropped requests. If completions come in for the dropped requests they will be treated as unexpected completions. • It is not necessary to call RESET_APP at the start of simulation.

Table 11-7 Service Class App Sequence Resets (Continued)

Reset Sequence for Application	Description
svt_pcie_target_app_service_reset_app_sequence	• This sequence implements ResetApp • RESET_APP resets the target application. All outstanding requests are dropped and will not be completed. • It is not necessary to call RESET_APP at the start of simulation.

Following is a code fragment showing midsim reset.

```
task midsim_reset_sequence::body();

    pcie_device_system_link_up_sequence link_up_seq;

    svt_pcie_requester_app_service_mem_range_sequence req_mem_range_seq;
    svt_pcie_requester_app_service_app_sequence req_app_seq;
    svt_pcie_requester_app_service_clr_stats_sequence req_clr_stats_seq;
    svt_pcie_requester_app_service_disp_stats_sequence req_disp_stats_seq;
    svt_pcie_requester_app_service_reset_app_sequence reset_requester_app_seq;
    svt_pcie_target_app_service_reset_app_sequence reset_target_app_seq;
    svt_pcie_pl_service_request_hot_plug_mode_sequence request_hot_plug_mode_seq;
    svt_pcie_device_virtual_reset_sequence  device_reset_vseq;

...
    // bring up link
    `svt_uvm_do(link_up_seq);

// Add memory ranges to Requester application

`svt_uvm_do_on_with(req_mem_range_seq,p_sequencer.endpoint_virt_seqr.requester_seqr, {
req_mem_range_seq.service_type == svt_pcie_requester_app_service::ADD_MEM_RANGE;
                                                                                  ...//
Reset after a few TLPs have been sent
     wait(ep_agent.status.pcie_status.tl_status.num_tlps_sent == 3);


     // Now reset both sides of the link
     /*
    // Note that this code has been left in intentionally to serve as an example on how
to reset only certain parts of the VIP
     // The reset virtual sequence called below will reset all app layers.

`svt_uvm_do_on_with(request_hot_plug_mode_seq,p_sequencer.endpoint_virt_seqr.pcie_virt_
seqr.pl_seqr,{mode == svt_pcie_pl_service::HOT_PLUG_UNPLUG;});


`svt_uvm_do_on_with(request_hot_plug_mode_seq,p_sequencer.root_virt_seqr.pcie_virt_seqr
.pl_seqr,{mode == svt_pcie_pl_service::HOT_PLUG_UNPLUG;});

     // App resets are optional, and are reset individually with a service call
     `svt_uvm_do_on_with(req_app_seq,p_sequencer.endpoint_virt_seqr.requester_seqr, {
req_app_seq.service_type == svt_pcie_requester_app_service::RESET_APP; });
```

```
      `svt_uvm_do_on(reset_target_app_seq,p_sequencer.root_virt_seqr.target_seqr[0]);
      */

      ->seq_to_test1;
      uvm_report_info( "TEST", "Triggering seq_to_test1 event.", UVM_NONE );

      // Just use the reset virtual sequence
      `svt_uvm_do_on(device_reset_vseq, p_sequencer.root_virt_seqr);

      // EP should dected the EIOS and automatically enter detect.
      wait(ep_agent.status.pcie_status.pl_status.ltssm_state ==
  svt_pcie_types::RECOVERY_RCVRLOCK);

      ->seq_to_test2;
      uvm_report_info( "TEST", "Triggering seq_to_test2 event.", UVM_NONE );

      `svt_uvm_do_on(device_reset_vseq, p_sequencer.endpoint_virt_seqr);

      // Wait some time and start everything back up
      #1000;

  `svt_uvm_do_on_with(request_hot_plug_mode_seq,p_sequencer.endpoint_virt_seqr.pcie_virt_
  seqr.pl_seqr,{mode == svt_pcie_pl_service::HOT_PLUG_DETECT;});


  `svt_uvm_do_on_with(request_hot_plug_mode_seq,p_sequencer.root_virt_seqr.pcie_virt_seqr
  .pl_seqr,{mode == svt_pcie_pl_service::HOT_PLUG_DETECT;});

      wait(ep_agent.status.pcie_status.pl_status.ltssm_state == svt_pcie_types::L0);

      `svt_uvm_do_on_with(req_app_seq,p_sequencer.endpoint_virt_seqr.requester_seqr, {
  req_app_seq.service_type == svt_pcie_requester_app_service::START_APP; });


      `svt_uvm_do_on_with(req_app_seq,p_sequencer.endpoint_virt_seqr.requester_seqr, {
  req_app_seq.service_type == svt_pcie_requester_app_service::IS_APP_FINISHED; });

      // Wait until Requester application has completed pumping in specified memory
  requests
      if(req_app_seq.is_finished == 'b0) begin
        // wait until requester application is finished.
       `svt_uvm_do_on_with(req_app_seq,p_sequencer.endpoint_virt_seqr.requester_seqr, {
  req_app_seq.service_type == svt_pcie_requester_app_service::WAIT_UNTIL_FINISHED; });
      end
  ...
     // Now reset both sides of the link
     /*
     // Note that this code has been left in intentionally to serve as an example on how
  to reset only certain parts of the VIP
     // The reset virtual sequence called below will reset all app layers.

  `svt_uvm_do_on_with(request_hot_plug_mode_seq,p_sequencer.endpoint_virt_seqr.pcie_virt_
  seqr.pl_seqr,{mode == svt_pcie_pl_service::HOT_PLUG_UNPLUG;});
```

```
`svt_uvm_do_on_with(request_hot_plug_mode_seq,p_sequencer.root_virt_seqr.pcie_virt_seqr
.pl_seqr,{mode == svt_pcie_pl_service::HOT_PLUG_UNPLUG;});

    // App resets are optional, and are reset individually with a service call
    `svt_uvm_do_on_with(req_app_seq,p_sequencer.endpoint_virt_seqr.requester_seqr, {
req_app_seq.service_type == svt_pcie_requester_app_service::RESET_APP; });

    `svt_uvm_do_on(reset_target_app_seq,p_sequencer.root_virt_seqr.target_seqr[0]);
    */

     ->seq_to_test1;
     uvm_report_info( "TEST", "Triggering seq_to_test1 event.", UVM_NONE );

     // Just use the reset virtual sequence
     `svt_uvm_do_on(device_reset_vseq, p_sequencer.root_virt_seqr);
...

`svt_uvm_do_on_with(request_hot_plug_mode_seq,p_sequencer.endpoint_virt_seqr.pcie_virt_
seqr.pl_seqr,{mode == svt_pcie_pl_service::HOT_PLUG_DETECT;});


`svt_uvm_do_on_with(request_hot_plug_mode_seq,p_sequencer.root_virt_seqr.pcie_virt_seqr
.pl_seqr,{mode == svt_pcie_pl_service::HOT_PLUG_DETECT;});

     wait(ep_agent.status.pcie_status.pl_status.ltssm_state == svt_pcie_types::L0);

     `svt_uvm_do_on_with(req_app_seq,p_sequencer.endpoint_virt_seqr.requester_seqr, {
req_app_seq.service_type == svt_pcie_requester_app_service::START_APP; });

     `svt_uvm_do_on_with(req_app_seq,p_sequencer.endpoint_virt_seqr.requester_seqr, {
req_app_seq.service_type == svt_pcie_requester_app_service::IS_APP_FINISHED; });

     // Wait until Requester application has completed pumping in specified memory
requests
     if(req_app_seq.is_finished == 'b0) begin
       // wait until requester application is finished.
      `svt_uvm_do_on_with(req_app_seq,p_sequencer.endpoint_virt_seqr.requester_seqr, {
req_app_seq.service_type == svt_pcie_requester_app_service::WAIT_UNTIL_FINISHED; });
     end

     `svt_uvm_do_on(req_disp_stats_seq,p_sequencer.endpoint_virt_seqr.requester_seqr);
     `svt_uvm_do_on(req_clr_stats_seq,p_sequencer.endpoint_virt_seqr.requester_seqr);
     `svt_uvm_do_on(req_disp_stats_seq,p_sequencer.endpoint_virt_seqr.requester_seqr);
endtask
```

11.19 Using FLR

Perform the following steps to enable FLR support:

1. Initiate `cfg_wr` in the Device Control Register (Offset 08h) to initiate FLR

2. Wait until the device enters `flr_active`—that is, state `max_expected_time_to_enter_flr_active_in_ns`
3. Initiate the traffic from VIP
4. The function must complete the FLR within 100 ms. After 100 ms, there will be no traffic pending for that function.

FLR support can be enabled using RC instance of the VIP—that is, by Config Read/Write sequences on the Device Control Register.

Example 11-3 Sample code

Driver App Transaction's Sequences:

```
svt_pcie_driver_app_transaction_cfg_wr_sequence       cfg_wr_sequence;
svt_pcie_driver_app_transaction_cfg_rd_sequence         cfg_rd_sequence;
```

Read the control register's content:

```
  `uvm_do_on_with(cfg_rd_sequence,
p_sequencer.root_virt_seqr.driver_transaction_seqr[0], {cfg_rd_sequence.address ==
16'h0100;
          cfg_rd_sequence.block == 1'b1;
          cfg_rd_sequence.register_number == 'h08;})

// Set FLR bit to 1
tmp_data = cfg_rd_sequence.req.payload[0] | 32'h0000_8000;
//
```

Now update the Register:

```
        `svt_xvm_do_on_with(cfg_wr_sequence,
p_sequencer.root_virt_seqr.driver_transaction_seqr[0], {cfg_wr_sequence.address ==
16'h0100;
          cfg_wr_sequence.block == 1'b1;
          cfg_wr_sequence.payload == tmp_data;
          cfg_wr_sequence.register_number == 'h08;})
```

For Config commands:

- *Address* field must be the Base address of the DUT EP
- *Register number* field must be the offset

11.20 Programming Hints and Tips

11.20.1 PIPE Polarity

You can use the svt_pcie_pl_configuration::invert_tx_polarity configuration member to programs polarity inversion on all lanes. It is a 32-bit vector where bit 0 control polarity inversion on lane 0. When a bit is set, the corresponding lane will invert polarity on all outgoing data. Note that it only works in serial mode.

11.20.2 Calls For Analysis Port Set Up and Usage

The following configuration members help you setup analysis ports on the monitor.

- rand int unsigned attribute svt_pcie_dl_configuration::received_dllp_interface_mode = 0.
  Select DLLPs available at Receive DLLP analysis port. DLLPs are filtered based on th bits enabled in received_dllp_interface_mode bit vector. See svt_pcie_dl::received_dllp_observed_port for accessing received DLLPs.

  Bits are defined in include/svt_pcie_common_defines.v as SVT_PCIE_SENT_DLLP_INTERFACE_MODE_[sel]_BIT where [sel] is defined as:

  - GOOD_PACKETS_BIT

  - ERR_PACKETS_BIT

- rand int unsigned attribute svt_pcie_dl_configuration::received_tlp_interface_mode = 0
  Select TLPs available at Receive TLP analysis port. TLPs are filtered based on the bits enabled in received_tlp_interface_mode bit vector. See svt_pcie_dl::received_tlp_observed_port for accessing received TLPs.

  Bits are defined in include/svt_pcie_common_defines.v as SVT_PCIE_RECEIVED_TLP_INTERFACE_MODE_[sel]_BIT where [sel] is defined as:

  - GOOD_PACKETS_BIT

  - ERR_PACKETS_BIT

- rand int unsigned attribute svt_pcie_dl_configuration::replay_timeout
  Length of the replay timer in symbols. If called, timeout value is sticky. Setting to value = 0 will enable automatic updates.

- rand int unsigned attribute svt_pcie_dl_configuration::sent_dllp_interface_mode = 0
  Select DLLPs available at Sent DLLP analysis port. DLLPs are filtered based on the bits enabled in sent_dllp_interface_mode bit vector. See svt_pcie_dl::sent_dllp_observed_port for accessing sent DLLPs.

  Bits are defined in include/svt_pcie_common_defines.v as SVT_PCIE_SENT_DLLP_INTERFACE_MODE_[sel]_BIT where [sel] is defined as:

  - ALL_PACKETS_BIT

- rand int unsigned attribute svt_pcie_dl_configuration::sent_tlp_interface_mode = SVT_PCIE_SENT_TLP_INTERFACE_MODE_DEFAULT
  Select TLPs available at Sent TLP analysis port. TLPs are filtered based on the bits enabled in sent_tlp_interface_mode bit vector. See svt_pcie_dl::sent_tlp_observed_port for accessing sent TLPs.

  Bits are defined in include/svt_pcie_common_defines.v as SVT_PCIE_SENT_TLP_INTERFACE_MODE_[sel]_BIT where [sel] is defined as:

- ALL_PACKETS_BIT.

You must set the sent_tlp_interface_mode and received_tlp_interface_mode members to enable the analysis ports--otherwise, no transactions appear.

The sent_tlp_interface_mode parameter is for enabling the analysis port. The following code hows the enabling of the analysis ports:

```
// Enable analysis ports.
cust_cfg.root_cfg.pcie_cfg.dl_cfg.sent_tlp_interface_mode = 1;
cust_cfg.root_cfg.pcie_cfg.dl_cfg.received_tlp_interface_mode = 1;
cust_cfg.root_cfg.pcie_cfg.dl_cfg.sent_dllp_interface_mode = 1;
cust_cfg.root_cfg.pcie_cfg.dl_cfg.received_dllp_interface_mode = 1;
```

If these configuration variables (sent_tlp_interface_mode and received_tlp_interface_mode) are set to 1, then you can use the class svt_pcie_dl_dllp_monitor_transaction to obtain sent and received TLPs using the analysis ports.

Following are the steps to set up the dl_monitors:

1. Use these connections.

   ```
   ep_agent.pcie_agent.dl.sent_tlp_observed_port.connect(sent_tlp_port);
   ep_agent.pcie_agent.dl.received_tlp_observed_port.connect(received_tlp_port);
   ```

   Refer to the following example file: ts.directed_pipe_test.sv

   ```
   env.root.pcie_agent.dl.sent_tlp_observed_port.connect(
    sent_tlp_subscriber.analysis_export );
   env.root.pcie_agent.dl.received_tlp_observed_port.connect(
    rcvd_tlp_subscriber.analysis_export );
   env.root.pcie_agent.dl.sent_dllp_observed_port.connect(
    sent_dllp_subscriber.analysis_export );
   env.root.pcie_agent.dl.received_dllp_observed_port.connect(
    rcvd_dllp_subscriber.analysis_export );
   ```

2. The following connections need to be enabled:

   svt_pcie_device_configuration -> svt_pcie_configuration -> svt_pcie_dl_configuration

   Use the following code:

   ```
   root_cfg.pcie_cfg.dl_cfg.sent_tlp_interface_mode = 1;
   root_cfg.pcie_cfg.dl_cfg.received_tlp_interface_mode = 1;
   root_cfg.pcie_cfg.dl_cfg.sent_dllp_interface_mode = 1;
   root_cfg.pcie_cfg.dl_cfg.received_dllp_interface_mode = 1;
   ```

11.20.3 Sequences and the uvm_sequence :: get_response Task

The driver does not know at what time a transaction is queued, and what its tag ID will be. But for every request queued into the driver, the driver issues a unique command number. You access the command number by referring to svt_pcie_driver_app_transaction::**command_num.** The command_num attribute is assigned when the transaction is queued**.**

Every sequence that is executed will generate a response, and thus uvm_sequence ::get_response should always be called for every request that has been queued. For posted commands, it still needs to be called. For nonposted commands, if a block is called, then when you call uvm_sequence ::get_response, you will have the completion information available.

If a block is set to '0', then you want to save the command_num. You would pick up commands later on the source_rx_transaction_out_port,. You can match the completion by checking command_num. If you want to wait for a completion or check if a request has completed, then there are service calls for that, and they all use command_num.

11.20.4 Setting the TH and PH Bits Using the Driver Application Class

You can set the TH and PH bits using the Driver App interface class and transaction class: svt_pcie_driver_app_transaction. Note the following members to implement this capability:

```
/**
   * Transaction Hint bit, indicates presence of TLP Processing Hints.
   */
 rand bit th    = 0;

/**
   *  Processing Hints field.
   */
 rand ph_enum    ph = BIDIRECTIONAL;

/**
Steering tag used when TLP processing hint is present. Bits [7:0] are part of the
request header. If the TH bit is set, then the steering tag field will always be
substituted for the tag field for memory writes. In addition, the steering tag field
will always be substituted for the byte enables for memory reads/atomic operations.
  */
 rand bit [7:0] st;
```

11.20.5 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 configuration member:

> rand int unsigned attribute svt_pcie_pl_configuration::*num_tx_ts1_in_polling_active* = SVT_PCIE_NUM_TX_S1_IN_POLLING_ACTIVE_DEFAULT

This member sets the number of training sets the LTSSM must transmit in Polling.Active before exiting this state. 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.

11.20.6 When to Invoke Service Calls

You should not make any service calls until after the VIP is properly configured and initialized. For example, you should not call the hot unplug service call while the LTSSM is still in it's initial state. The hot unplug call may immediately kick the LTSSM into detect before it has a chance to finish initializing.

11.20.7 Exceptions and Scrambler Control Bits

The svt_pcie_pl_proxy code has been implemented so that if there is a svt_pcie_symbol_exception, then the svt_pcie_symbol_exception->scrambler_control bits are used (except when error_kind == "NO_ERROR").

You must set the scrambler_control bits for all symbols when using svt_pcie_symbol_exception class. The following table shows the values available to you with the scrambler_control enum:

Table 11-8 Values for Setting Control Bits of Scrambler

Name	Value	Description
NONE	b'00	Disables scrambler for the specified symbol.
FORCE_SCRAMBLE	b'01	Forces the scrambler to scramble the symbol to be transmitted (even if it is not supposed to be scrambled as per PCIe rules).
INIT_SCRAMBLER	b'10	Resets and initializes scrambler. The specified symbol is not scrambled.
INIT_AND_FORCE_SCRAMBLER	b'11	Resets, initializes scrambler and forces the scrambler to scramble the symbol to be transmitted (even if it is not supposed to be scrambled as per PCIE rules).

Summary: when a symbol is changed then the scrambler_control needs to reflect what you wants to occur for this symbol.

11.20.8 User TS1 Ordered Set Notes

User TS1s Ordered Sets can only be used in legitimate LTSSM states. You can formulate and send user TS OS only in those LTSSM states where it is legitimate to send a TS OS. The Ordered Set creation task would simply stop those default TS Ordered Sets, and let the user TS go on the bus.

Note the following:

- User TS1 OS replace outgoing TS1's, but they do not override all outgoing data.
- Users will never see a user TS1 OS in Recovery.Idle because the LTSSM is required to send idles.
- For users requiring TS1 to be sent in Recovery.Idle, the best way to do this is to start up the user TS in the state before Recovery.Idle (recovery.rcvrcfg), and turn on user TS2. The LTSSM will not make a state transition while the user TS are turned on.

```
`uvm_info(get_type_name(),$sformatf("Begin process set_tx_ts1_pattern_seq 2nd
time\n"),UVM_NONE);
   set_tx_ts1_pattern_seq.randomize() with {
        set_tx_ts1_pattern_seq.user_tx_ts_enable         == 1'b1;
        set_tx_ts1_pattern_seq.min_user_tx_ts_burst      == 1;
        set_tx_ts1_pattern_seq.max_user_tx_ts_burst      == 1;
        set_tx_ts1_pattern_seq.min_user_tx_ts_spacing    == 2;
        set_tx_ts1_pattern_seq.max_user_tx_ts_spacing    == 2;
};
```

11.21 Up/Down Configure

As part of bandwidth management and means to optimize power consumption, PCIe protocol supports changing the link width even after the TLP traffic has started (that is, Up/Down configure implies changing link width when link_up is 1). RC and EP VIP instances support this protocol feature and can act an initiator or target of the link width change request.

The following methods, service requests, and class variables are required to achieve the desired functionality. For more details, see HTML class reference documentation.

Service Requests

```
svt_pcie_pl_service
```

Control Methods

```
svt_pcie_pl_configuration::set_link_width_values( , ,)
svt_pcie_device_agent::reconfigure_via_task(). Alternatively one can use
svt_pcie_device_agent_service  requests with svt_pcie_device_agent_service::service_type
= svt_pcie_device_agent_service::REFRESH_CFG
```

Control Properties

```
svt_pcie_pl_configuration::upconfigure_capable
svt_pcie_pl_configuration::link_width
svt_pcie_pl_configuration::mpipe_turn_off_unused_lanes_after_initial_link_training
svt_pcie_pl_service::service_type = svt_pcie_pl_service::INITIATE_LINK_WIDTH_CHANGE
```

Status Properties

```
svt_pcie_pl_status::initial_negotiated_link_width
svt_pcie_pl_status::negotiated_link_width
```

Use Model for SVT PHY Layer Service Request Initiate_Link_Width_Change

The service request directs the LTSSM to change the link width of a link that is already in link up state. The VIP LTSSM shall service this request from states L0/TX_L0_Rx_L0s/L1. Issuing the service request when VIP LTSSM is not in one of the above listed states will result in the service request being ignored. This service request must be used in conjunction with the properties in `svt_pcie_pl_configuration` class.

1. Set the desired link width, supported link width and expected link width properties using the method `set_link_width_values` provided in class `svt_pcie_pl_configuration`.
2. Invoke `reconfigure_via_task` or issue `REFRESH_CFG` service request so that the values communicated via above step percolate to VIP's local copy of the configuration object.
3. Issue `svt_pcie_pl_service::INITIATE_LINK_WIDTH` change service request and wait for the LTSSM to transition from L0/L1/RX.L0s -> Recovery ->Config -> L0.
4. In case the VIP instance is the target of link width change request initiated by its partner (that is, the DUT), the testbench can set the appropriate value of `expected_link_width` by using `set_link_width_values`.

Note

If `set_link_width_values` method takes one input—that is, maximum width of the DUT, then VIP auto-computes supported and expected widths based on the value provided.

For correct setup, set the width to be attempted—that is, configured or number of physical lanes available and should not be changed during the simulation (supported link widths and the final expected width). The supported widths input has to be a superset of expected link width, otherwise VIP may issue link width not supported warnings as shown in the following example:

`If set_link_width_values (16, 32'h0F, 12)` *// Here, maximum width is 16 and supported link width vector 32'h0F (that is, 0000_1111 implies only supported upto link width 8) and expected link width is 12.*

```
UVM_WARNING @ 0.000 ns: reporter [is_valid] Invalid expected_link_width
of 12 provided, must be <= 8.
```

To avoid this warning, set `set_link_width_values` to 16, 32'h3F, 12.

11.22 Lane Reversal

The order of lanes in a multi-lane PCIe link may require a change as part of the physical link training. This feature is known as lane reversal and is traditionally verified by redoing the testbench connections between VIP instance and DUT instance thus adding a compile dependency. PCIe SVT VIP offers run time control to select the order of lanes.

Control Properties

svt_pcie_pl_configuration::lane_reversal_mode can take any of the four enumerated values UNSUPPORTED,FORCED,SUPPORTED, FORCED_AND_SUPPORTED. For more details, see HTML class reference documentation.

Use Model for Lane Reversal Feature

- svt_pcie_pl_configuration::lane_reversal_mode must be modified while the VIP LTSSM is in DETECT state (before the training is initiated).
- svt_pcie_pl_configuration::lane_reversal_mode = FORCED_AND_SUPPORTED is not applicable to EP VIP instance.
- Note that the lane number value passed to other VIP API (equalization coefficients control, receiver present control, lane polarity control, and so on) use logical lane number, therefore enabling lane reversal will not have any impact on the existing tests.

11.23 Lane Reversal with Different Link Width Configurations

Usage notes for supported link width with lane reversal enabled.

- Set the MAX_SUPPORTED_DUT_LINK_WIDTH = <user_link_width>
- Set svt_pcie_pl_configuartion attribute lane_reversal_mode to FORCED.
- Call the set_link_width_values function for changing the link width

  The set_link_width_values function accepts the following three inputs:

- link_width_value (svt_pcie_pl_configuartion: link_width_value)
- supported_link_width_vector_value (svt_pcie_pl_configuartion: supported_link_width_vector_value)
- expected_link_width_value (svt_pcie_pl_configuartion: expected_link_width_value)
  - The link_width_value (svt_pcie_pl_configuartion: link_width_value) must be same as DUT link width value.
  - The supported_link_width_vector_value (svt_pcie_pl_configuartion: supported_link_width_vector_value) must have all the possible link widths a VIP can support from 1 up to link_width_value value.

  i. When unset (second argument) in the function call, it prompts VIP to set supported_link_width_vector_value to support all the possible link widths from 1 up to link_width_value value.

  The controls for supported link width vector are as follows:

Table 11-9 Controls for Supported Link Width Vector

Link Width	supported_link_width_vector
1	32'h01
2	32'h03
4	32'h07
8	32'h0F
12	32'h1F
16	32'h3F;
32	32'h7F;

  - expected_link_width_value (svt_pcie_pl_configuartion: expected_link_width_value): The expected negotiated link width value. This value must be same as supported link width vector's link width value.

  i. When unset in the function call, it prompts VIP to set expected_link_width value same as the value of link_width_value argument.

Example 11-4 VIP-DUT Setup

- MAX_SUPPORTED_DUT_LINK_WIDTH = 16
- Set the set_link_width_values (16, 32'h03, 2) for VIP // Here Supported link width vector 32'h03 and expected link width is 2
- Set the lane_reversal_mode to FORCED for VIP
- The link up happens at X2 on [Lane 3, Lane 2]

Example 11-5 VIP-DUT Setup

- MAX_SUPPORTED_DUT_LINK_WIDTH = 32

- Set the set_link_width_values (32, 32'h01, 1) for VIP // Here Supported link width vector 32'h01 and expected link width is 1
- Set the lane_reversal_mode to FORCED for VIP
- The link up happens at X1 on [Lane 31].

Example 11-6 VIP-DUT Setup

- MAX_SUPPORTED_DUT_LINK_WIDTH = 8
- Set the set_link_width_values (8, 32'h07, 4) for VIP // Here Supported link width vector 32'h04 and expected link width is 4
- Set the lane_reversal_mode to FORCED for VIP
- The link up happens at X4 [Lane7, Lane6, Lane5, Lane4]

Example 11-7 VIP-VIP Setup or PHY-DUT Setup

- MAX_SUPPORTED_DUT_LINK_WIDTH = 8
- Set the set_link_width_values (8, 32'h01, 1) for VIP(RC) // Here Supported link width vector 32'h04 and expected link width is 1
- Set the lane_reversal_mode for VIP(RC) to FORCED
- Set set_link_width_values (8, 32'h07, 4) for VIP(EP) // Here Supported link width vector 32'h04 and expected link width is 4
- Set the lane_reversal_mode for VIP(EP) to SUPPORTED
- The link up happens at X1 [Lane7]

Example 11-8 VIP-VIP Setup or PHY-DUT Setup

- MAX_SUPPORTED_DUT_LINK_WIDTH = 8, Lane Reversal ENABLED
- Set the set_link_width_values (8) for VIP(RC) // Here Supported link width vector 32'h0F (depends upon the first Argument) and expected link width is 8(depends upon the first argument)
- Set the lane_reversal_mode for VIP(RC) to FORCED
- Set the set_link_width_values (8) for VIP(EP) // Here Supported link width vector 32'h0F (depends upon the first Argument) and expected link width is 8 (depends upon the first argument)
- Set the lane_reversal_mode for VIP(EP) to SUPPORTED
- The link up happens at X8 [Lane7, Lane6, Lane5, Lane4, Lane3, Lane2, Lane1, Lane0]

Note

- For detailed description of lane_reversal_mode and set_link_width_values method, see svt_pcie_pl_configuartion class in the HTML class reference documentation.
- The description in the previous section shows the use of set_link_width_values to control the initial link width (before physical link up transitions from 0 to 1). In subsequent course of simulation (that is, post link up is set to 1) if test intends to change the link width, then link_width_value (first argument of method set_link_width_values) value must be modified and it is recommended to retain supported_link_width_vector_value as is by resupplying its current value as input to the method.

11.24 User-Supplied Memory Model Interface

The user-supplied memory interface allows you to direct the SVT PCIe VIP application layer to utilize an external memory model to store TLP payloads. This can be useful in systems where memory is allocated from a central resource.

The package class `svt_mem_backdoor_base` provides the base API between the `svt_pcie_mem_target` and a desired memory model. `svt_pcie_mem_target_gmem_model` (generic memory model) is the default implementation of this interface and provides the base memory model of the application layer `mem_target` component. The model provides an example of the currently utilized and required minimum features of the `svt_mem_backdoor_base` API by the mem_target.

You can extend this class to implement linkage to your memory model and map the interface through the config_db for mem_target use. Required functions to overload in the extended user memory interface class are `peek_base`, `poke_base` and `free_base`.

The config_db must be set during initial configuration in the build phase. Overrides of the user memory model will be ignored during subsequent phases and reconfiguration operations.

Perform the following steps to connect a user memory interface object:

1. Extend the model `svt_pcie_mem_target_gmem_model` and overload the functions `poke_base`, `peek_base` and `free_base` to communicate with your memory model.
2. In test `build_phase` of the test or environment class, construct an extended `mem` model object.
3. Pass the memory interface object handle to `mem_target` instance through `config_db` as `user_gmem_model`.

Example override code:

```
class user_gmem_model extends svt_pcie_mem_target_gmem_model;
 …
endclass

…
virtual function void build_phase(uvm_phase phase);
  // handle to the user's gmem implementation
  user_gmem_model user_model;

  // Build up default test and environment
  super.build_phase(phase);

  /*
   * Create and assign a user override model IN BUILD PHASE
   */
  user_model = new("user_model", this);
  svt_config_object_db#(svt_mem_backdoor_base)::set(this, "<relative path to pcie
device>.mem_target", "user_gmem_model", user_model);

endfunction
```

`mem_target` can return the memory interface object handle at any time with `svt_pcie_mem_target::get_backdoor()`:

```
memory interface handle = <svt_pcie_vip_instance>.mem_target.get_backdoor();
```

For more information about the API features of `svt_pcie_mem_target_gmem_model` and `svt_mem_backdoor_base`, see HTML class reference documentation.

11.25 External Clocking and Per Lane Clocking for Serial Interface

The PCIe VIP supports the following clocking modes:

- Internal transmit bit clock mode - VIP serial transmission depends on internally generated clock.
  - Common clock for all lanes - Internally generated clock is common for all lanes.
    - Enabled by setting `ENABLE_PER_LANE_CLOCKING = 0` and `TRANSMIT_BIT_CLOCK_MODE = 0`.
  - Per lane clocking - Internally generated clocks are on per lane basis.
    - Enabled by setting `ENABLE_PER_LANE_CLOCKING = 1` and `TRANSMIT_BIT_CLOCK_MODE = 0`.
- External transmit bit clock mode - VIP expects external bit clock at physical transmission rate fed to the VIP as input. In this mode, VIP assumes that the jitter if any present is applied to the externally supplied clock.
  - Common clock for all lanes - Externally provided clock is expected to be common for all lanes.
    - Enabled by setting `ENABLE_PER_LANE_CLOCKING = 0` and `TRANSMIT_BIT_CLOCK_MODE = 1`.
  - Per lane clocking - Externally provided clocks are expected to be on per lane basis.
    - Enabled by setting `ENABLE_PER_LANE_CLOCKING = 1` and `TRANSMIT_BIT_CLOCK_MODE = 1`.

11.25.1 Enabling External Clocking and Per Lane Clocking Modes

- External Clocking mode: External clocking is disabled by default in PCIe VIP. You can use the following Verilog parameter to enable external transmit bit clock mode.

Attribute	Type	Description	Comments
TRANSMIT_BIT_CLOCK_MODE	Verilog Parameter	Verilog parameter to specify clocking mode. 0 => internal bit clocks are used to transmit serial data. 1 => external bit clocks are used to transmit serial data.	Attribute is applicable for all lanes.

For example,

Set up for configuring external clocking mode from RC:

```
spd_0.SVT_PCIE_UI_TRANSMIT_BIT_CLOCK_MODE  = 1;
```

- External clocking signaling interface: The VIP model has per lane per link speed clocking speed inputs for external bit transmit clock mode. This gives you an option to connect to the VIP model in external transmit bit clock mode when you do not have link speed multiplexed clocking pin. If you have a single output wire per lane for gen1/gen2/gen3/gen4 link speed, then you can connect the VIP model per lane per link speed pin with the link speed multiplexed on per lane basis. You can use the following signals to connect for external clocking mode.

Attribute	Type	Description	Comments

svt_pcie_ext_clk_intf::logic [31:0] tx_clk_2_5g	Input	Per lane external bit clock at Gen1(2.5GTS) rate. The interface signal is defined for all 32 lanes but you must connect the maximum number of physical lanes your design supports	Attribute is applicable for each lane.
svt_pcie_ext_clk_intf::logic [31:0] tx_clk_5g	Input	Per lane external bit clock at Gen2(5GTS) rate. The interface signal is defined for all 32 lanes but you must connect the maximum number of physical lanes your design supports.	Attribute is applicable for each lane.
svt_pcie_ext_clk_intf::logic [31:0] tx_clk_8g	Input	Per lane external bit clock at Gen3(8GTS) rate. The interface signal is defined for all 32 lanes but you must connect the maximum number of physical lanes your design supports.	Attribute is applicable for each lane.
svt_pcie_ext_clk_intf::logic [31:0] tx_clk_16g	Input	Per lane external bit clock at Gen4(16GTS) rate. The interface signal is defined for all 32 lanes but you must connect the maximum number of physical lanes your design supports.	Attribute is applicable for each lane.

For example,

Set up for configuring external clocking mode from RC:

```
spd_0.SVT_PCIE_UI_TRANSMIT_BIT_CLOCK_MODE  = 1;
assign port_if_0.ext_clk_if.tx_clk_2_5g = spd_1.m_ser.port0.tx_bit_clk;
assign port_if_0.ext_clk_if.tx_clk_5g = spd_1.m_ser.port0.tx_bit_clk;
assign port_if_0.ext_clk_if.tx_clk_8g = spd_1.m_ser.port0.tx_bit_clk;
assign port_if_0.ext_clk_if.tx_clk_16g = spd_1.m_ser.port0.tx_bit_clk;
```

- Per lane clocking mode: Per lane clocking is disabled by default in PCIe VIP. You can use following Verilog parameter to enable per lane clocking in conjunction with enabling external clocking mode.

Attribute	Type	Description	Comments
ENABLE_PER_LANE_CLOCKING	Verilog Parameter	Verilog parameter to enable per lane clocking in vip model.	Attribute is applicable for all lanes.

For example,

```
spd_0.SVT_PCIE_UI_ENABLE_PER_LANE_CLOCKING  = 1;
spd_1.SVT_PCIE_UI_ENABLE_PER_LANE_CLOCKING  = 1;
```

Note

- If you are not running with external transmit bit clock mode, then it is not required to connect to the VIP model ext_*_tx_bit_clk clocking signals.
- If per lane clocking is disabled and VIP is running in external clocking mode then you only need to connect to VIP lane 0 clock (tx_clk_2_5g[0], tx_clk_8g[0], tx_clk_8g[0], tx_clk_16g[0]).
- VIP is required to have correct external clocks even in low power states where reference clock might be switched off, so the onus on providing the correct clock even in case of low power scenario is on the testbench.

11.26 Callbacks

The callback used to alter transmitting symbols is `pre_symbol_out_put` (`svt_pcie_pl` PL, `svt_pcie_symbol` symbols[]). This symbol level callback is issued by the component at every symbol time after gathering all the symbols to be transmitted on the PCIe link. This is the last chance to the user to corrupt any symbol before it goes on the link. The events associated with this callback are as follows:

- `TLP_STARTED` – TLP started with this symbol.
- `DLLP_STARTED` – DLLP started with this symbol.
- `SKP_STARTED` – SKP started with this symbol.
- `TS1_STARTED` – TS1 started with this symbol.
- `TS2_STARTED` – TS2 started with this symbol.
- `EIOS_STARTED` – EIOS started with this symbol.
- `EIEOS_STARTED` – EIEOS started with this symbol.
- `SDS_STARTED` – SDS started with this symbol.
- `EDS_STARTED` – EDS started with this symbol.
- `EDB_TOKEN_STARTED` – EDB token started with this symbol
- `CTRL_SKP_STARTED` – CTRL-SKP started with this symbol.

Note that all the above events are triggered when the symbol is ready to be transmitted and will not remain triggered while transmitting the related ordered set.

Example 11-9 Usage Example

In the following example, a SKP-END of the transmitting Ctrl-SKP OS is being changed from 78h to FFh.

```
virtual function void pre_symbol_out_put(svt_pcie_pl pl, svt_pcie_symbol symbols[]);
    if(error_count < 1) begin
      if(symbols[0].symbol_event == svt_pcie_symbol::CTRL_SKP_STARTED &&
pl.status.current_speed == svt_pcie_pl_status::SPEED_16_0G) begin
        `svt_xvm_note("TEST: pre_symbol_out_put_callback",$sformatf("\n found event
svt_symbol::CTRL_SKP_STARTED.\n"))
        ctrl_skp_found = 1;
      end
      if(ctrl_skp_found) begin
        if((ordered_set_idx == 0 || ordered_set_idx == 0 || ordered_set_idx == 4 ||
ordered_set_idx == 8 || ordered_set_idx == 12 || ordered_set_idx == 16 ||
ordered_set_idx == 20) && symbols[0].data == 8'h78) begin // catching CTRL-SKP End 78h
          for(int i=0; i< symbols.size(); i++) begin
            svt_pcie_symbol_exception_list sym_exc_list = new();
            svt_pcie_symbol_exception sym_exc = new();
```

```
            `svt_xvm_debug("TEST: pre_symbol_out_put_callback",$sformatf("\n Corrupting
Ctrl-SKp End in lane %0d.\n", ordered_set_idx))
            sym_exc.error_kind = svt_pcie_symbol_exception::CORRUPT_DATA_VALUE_ONLY;
            sym_exc.corrupted_data = 'hFF;
            sym_exc.scrambler_control = svt_pcie_symbol_exception::FORCE_SCRAMBLE;
            sym_exc_list.add_exception(sym_exc);
            symbols[i].exception_list = sym_exc_list;
          end
          error_count++;
        end
        ordered_set_idx++;
      end
    end
  endfunction
```

11.26.1 Rx Symbol Callback

The callback used to get the received symbols information is `post_symbol_in_get` (`svt_pcie_pl` PL, `svt_pcie_symbol` symbols[]). This symbol level callback is issued by the component at every symbol time after gathering all the symbols being received on the PCIe link. The event associated with this callback are as follows:

- `SKP_ENDED` – SKP ended with this symbol.
- `TS1_ENDED` – TS1 ended with this symbol.
- `TS2_ENDED` – TS2 ended with this symbol.
- `EIOS_ENDED` – EIOS ended with this symbol.
- `EIEOS_ENDED` – EIEOS ended with this symbol.
- `SDS_ENDED` – SDS ended with this symbol.
- `CTRL_SKP_ENDED` – CTRL-SKP ended with this symbol.

Note that all the above events are triggered when the symbol is completely received and will not remain triggered while the related ordered set is being received.

Example 11-10 Usage Example

The following example increments a counter whenever a SKP OS and SDS OS is received.

```
virtual function void post_symbol_in_get(svt_pcie_pl pl, svt_pcie_symbol symbols[]);
    if(symbols[0].symbol_event == svt_pcie_symbol::SKP_ENDED) begin
      skp_count++;
      `svt_xvm_debug("TEST: post_symbol_in_get_callback",$sformatf("\n Received SKP OS..
RCVD SKP OS count is %0d.\n",skp_count))
      $display("symbols.size() %0d", symbols.size());
end
```

11.26.2 Framing Token Callback

The callback used to get the received framing token contents at 128/130b is `framing_token_in_get` (`svt_pcie_pl` pl, `svt_pcie_symbol` symbol). When this callback gets executed the `symbol_event` also gets populated with appropriate token type STP/EDS/EDB/SDP/IDL.

Therefore, at the execution of callback `framing_token_in_get` you can get framing token type in `symbol_event`, framing token contents in `rx_framing_token_data` and last byte of the received framing token in data variable. The event associated with this callback are as follows :

- `STP_FRAMING_TOKEN_ENDED` - STP framing token ended with this symbol.
- `EDS_FRAMING_TOKEN_ENDED` - EDS framing token ended with this symbol.
- `EDB_FRAMING_TOKEN_ENDED` - EDB framing token ended with this symbol.
- `SDP_FRAMING_TOKEN_ENDED` - SDP framing token ended with this symbol.
- `IDL_FRAMING_TOKEN_ENDED` - IDL framing token ended with this symbol.

Note the framing tokens will be populated in the following manner in the register `rx_framing_token_data`:

```
* STP : 32'h{TLPLength[3:0],F,FP,TLPLength[10:4],FCRC,TLPSequenceNumber}
* EDS : 32'h1f809000
* EDB : 32'hc0c0c0c0
* SDP : 32'hf0ac0000
* IDL : 32'h00000000
```

Example 11-11 Usage Example

The following example receives the framing tokens through callback in the test and compares them with the specification defined framing token encodings.

```
  virtual function void framing_token_in_get(svt_pcie_pl pl, svt_pcie_symbol symbol);
    if (symbol.symbol_event == svt_pcie_symbol::EDS_FRAMING_TOKEN_ENDED)
      begin
        if (symbol.rx_framing_token_data != 32'h1f809000)
          `svt_xvm_error("rc_framing_token_in_callback", "RC received EDS with incorrect
EDS token data.");
      end
    else if (symbol.symbol_event == svt_pcie_symbol::SDP_FRAMING_TOKEN_ENDED)
      begin
        if (symbol.rx_framing_token_data != 32'hf0ac0000)
          `svt_xvm_error("rc_framing_token_in_callback", "RC received SDP with incorrect
SDP token data.");
      end
    else if (symbol.symbol_event == svt_pcie_symbol::EDB_FRAMING_TOKEN_ENDED)
      begin
        if (symbol.rx_framing_token_data != 32'hc0c0c0c0)
          `svt_xvm_error("rc_framing_token_in_callback", "RC received EDB with incorrect
EDB token data.");
      end
    else if (symbol.symbol_event == svt_pcie_symbol::IDL_FRAMING_TOKEN_ENDED)
      begin
        if (symbol.rx_framing_token_data != 32'h00000000)
          `svt_xvm_error("rc_framing_token_in_callback", "RC received IDL with incorrect
IDL token data.");
      end
  endfunction
```

12 PCIe Device Agent

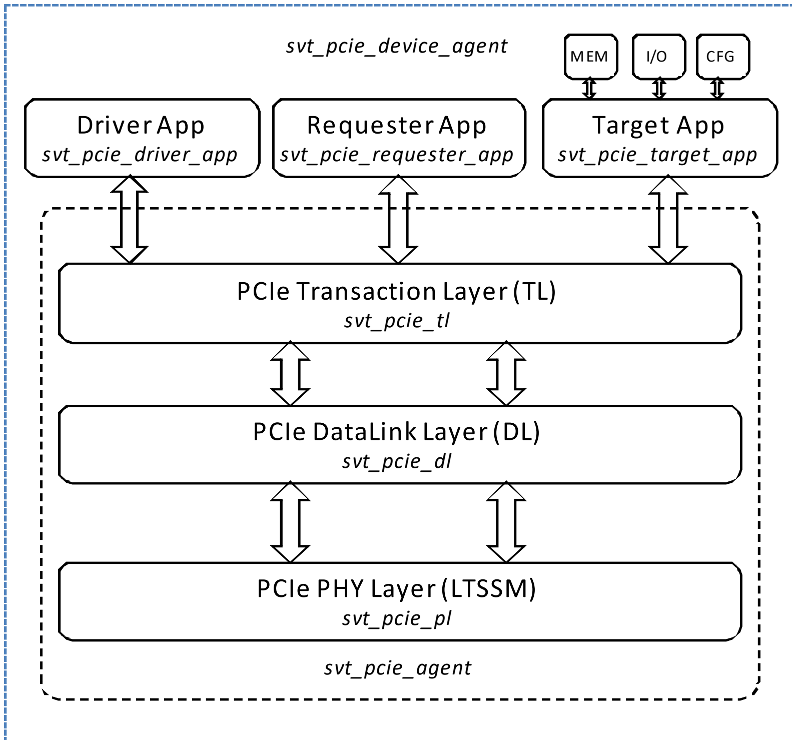
This chapter discusses the following topics:

- Overview
- Configuration
- Status
- Sequencers

12.1 Overview

The PCIe UVM VIP, at its highest level, is composed of the `svt_pcie_device_agent` class, which encapsulates the PCIe Agent (Class type=`svt_pcie_agent`)—an application layer that comprises of Driver Application, Requester Application, and Target Application (Memory, I/O, Configuration database).

Figure 12-1 Block Diagram – PCIe UVM VIP



- The Application Layer: The PCIe VIP has a layer on top of the PCIe stack representing the software layer in a real application of the PCIe bus. The application layer is responsible for generating and handling transactions. The application layer of the PCIe VIP is the layer that is typically programmed by the test to generate stimulus or respond to the incoming requests. The application layer of the PCIe VIP has the following blocks that perform specific functions:
  - Driver Application (Type=`svt_pcie_driver_app`, Instance=`driver[0]`): The Driver application provides a simple interface that can be used to quickly create PCIe transaction requests (memory read/write request, I/O read/write requests etc.,). The application deals with driver application transaction objects (`svt_pcie_driver_app_transaction`) which is an abstract description of the transaction layer packet.
  - Requester Application (Type=`svt_pcie_requester_app`, Instance=`requester`): The requester application can be used to generate PCIe memory/read transaction to a remote target. The application can be configured to choose addresses at random (constrained by minimum/maximum configuration parameters) and have varying lengths (again, constrained by minimum/maximum configuration parameters) and generate traffic at a requested bandwidth (again, constrained by minimum/maximum configuration parameters). This application will be useful where a background exerciser of write/read request is required.
  - Target Application (Type=`svt_pcie_target_app`, Instance=`target[0]`): The target application is the block that automatically responds to various inbound requests. Read transaction requests can be optionally broken up into multiple completions, potentially interleaved with other read completions by configuration. The target application has auxiliary blocks that represent the completion memory used while generating completions. These blocks include Memory Target, IO Target and Configuration Database. The blocks comprise of a sparse memory allowing a wide variety memory/IO addresses/registers to be accessed by a DUT.

    - Memory Target (Type=`svt_pcie_mem_target`, Instance=`mem_target`): The memory target is the PCIe VIP's sparse memory model used to store write data and return read data to the incoming memory requests. This sparse memory model responds to a wide variety of addresses (32-bit and 64-bit) when accessed by a requester. The memory target has APIs to write into its sparse memory via backdoor or read from its sparse memory via backdoor to meet different kinds of testing requirements.
    - I/O Target (Type=`svt_pcie_io_target`, Instance=`io_target`): The I/O target is the PCIe VIP's sparse memory model used to store write data and return read data to the incoming I/O requests. This sparse memory model responds to a wide variety of addresses (32-bit and 64-bit) when accessed by a requester. The I/O target has APIs to write into its sparse memory via backdoor or read from its sparse memory via backdoor to meet different kinds of testing requirements.
    - Configuration Database (Type=`svt_pcie_cfg_database`, Instance= `svt_pcie_target_app::cfg_database`): The configuration database is the PCIe VIP's sparse memory model used to store write data and return read data to the incoming configuration requests. This sparse memory model responds to a wide variety of addresses (type 0, type 1, extended capability registers, and so on) when accessed by a requester. The configuration database has APIs to write into its sparse memory via backdoor or read from its sparse memory via backdoor to meet different kinds of testing requirements.
- PCIe Agent: The PCIe Agent encapsulates the UVM drivers that represents the Transaction Layer, the Data-link Layer and the Physical Layer of the PCIe protocol stack. For more details, see "PCIe Agent" on page 229.

12.2 Configuration

The PCIe Device Agent is configured using an object of class `svt_pcie_device_configuration`. This class has other class objects defined within it to form a hierarchy that corresponds to the hierarchy inside the PCIe Device Agent.

```
svt_pcie_device_configuration
              |
              |--------> driver_cfg[]  (type=svt_pcie_driver_app_configuration)
              |
              |--------> requester_cfg (type=svt_pcie_requester_app_configuration)
              |
              |--------> target_cfg[]  (type=svt_pcie_target_app_configuration)
              |
              |--------> pcie_cfg      (type=svt_pcie_configuration)
                                |
                                |--------> tl_cfg  (type=svt_pcie_tl_configuration)
                                |
                                |--------> dl_cfg  (type=svt_pcie_dl_configuration)
                                |
                                |--------> pl_cfg  (type=svt_pcie_pl_configuration)
```

The PCIe Device Agent is configured using an object of `svt_pcie_device_configuration` class. This class is comprised of direct variables and class objects that are used to configure other agents/drivers that are part of the device agent. The following table contains the attributes of this class.

Table 12-1 Device Configuration Members

Type	Member	Description
svt_pcie_driver_app_configuration	driver_cfg [int]	Holds the configuration attributes for the Driver application.
bit	is_active = 1;	Specifies if the agent is an active or passive component. Allowed values are: 1, Configures component in active mode. Enables sequencer and driver. 0:=, Configures component in passive mode. Enables only the monitor in the agent. Configuration type: Static
string	model_instance_scope	This is the full hierarchical path name to the instance of the Verilog instantiation model. The programming of this variable binds the PCIe UVM Agent with its Verilog counterpart.
pipe_spec_ver_enum	pipe_spec_ver	PCIE PIPE specification version number. Allowed values are: • svt_pcie_device_configuration::PIPE_SPEC_VER_2, configures the PCIE PIPE specificaiton version number to 2.0 • svt_pcie_device_configuration::PIPE_SPEC_VER_4, configures the PCIE PIPE specificaiton version number to 4.0 • svt_pcie_device_configuration::PIPE_SPEC_VER_4_2, configures the PCIE PIPE specificaiton version number to 4.2 • svt_pcie_device_configuration::PIPE_SPEC_VER_4_3, configures the PCIE PIPE specificaiton version number to 4.3
pcie_spec_ver_enum	pcie_spec_ver	Specifies the PCIE specification version number. Allowed values are: • svt_pcie_device_configuration::PCIE_SPEC_VER_1_1, configures the PCIE specification to version 1.1 Note: Active component does not support PCIe specification version 1.1. • svt_pcie_device_configuration::PCIE_SPEC_VER_2_0, configures the PCIe specification to version 2.0 • svt_pcie_device_configuration::PCIE_SPEC_VER_2_1, configures the PCIe specification to version 2.1 • svt_pcie_device_configuration::PCIE_SPEC_VER_3_0, configures the PCIe specification to version 3.0 • svt_pcie_device_configuration::PCIE_SPEC_VER_3_1, configures the PCIe specification to version 3.1 • svt_pcie_device_configuration::PCIE_SPEC_VER_4_0, configures the PCIe specification to version 4.0
svt_pcie_configuration	pcie_cfg	Holds the configuration attributes for the PCIE agent.

Table 12-1 Device Configuration Members

Type	Member	Description
svt_pcie_requester_app_configuration	requester_cfg	Holds the configuration attributes for the Requester application.
svt_pcie_target_app_configuration	target_cfg [int]	Holds the configuration attributes for the Target application.

Note
The variable `model_instance_scope` is a string variable that must be set to a value that is the Verilog Cross-module reference to the Verilog instantiation model that is instanced and connected to the DUT. This configuration parameter binds the UVM agent to its Verilog counterpart. If this variable is not set correctly, then it results in an error at runtime.

12.2.1 Initial Configuration

The initial configuration of the PCIe Device Agent (and all of its sub-components) is established using the configuration database (`uvm_config_db`) class defined in UVM. The PCIe Device Agent—the recipient of the configuration object has a `uvm_config_db::get()` defined within the `build_phase()` and so the parent test/environment has to specify a `uvm_config_db::set()` in its `build_phase()`. Before calling the `uvm_config_db:set()`, the desired configuration attributes must be set to user-defined values. The example below illustrates the initial configuration step.

The `uvm_config_db::set()` in the illustration above will program the agent and all of its sub-components. In the Example 12-1, only a few parameters from the different layers are shown as being modified and for the rest, a default value is assumed. For the complete list of the configuration attributes and their default values check the HTML class description of `svt_pcie_device_configuration` at the following location: *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_device_configuration.html*

Example 12-1

```
class pcie_device_basic_env extends uvm_env;
   …
   svt_pcie_device_agent root;
   svt_pcie_device_configuration root_cfg
   …
   function void build_phase( uvm_phase phase );
      super.build_phase( phase );
      …
      root_cfg = new ("root_cfg");
      // Functions that programs values to different configuration parameter.
      setup_system_defaults(root_cfg);
      // Setting up the UVM agent with its Verilog counterpart.
      root_cfg.model_instance_scope = "test_top.root";
      // Setting the type of the device. In this example a root complex.
      root_cfg.device_is_root = 1;

      // Call the uvm_config_db::set() to set the configuration of the UVM agent.
      // Arg1 & Arg2: context & instance name. Hierarchical path to the object data
      // Arg3: Field name, "cfg" is the field name used by the agent class
      // Arg3: Object/Value being set
      uvm_config_db#(svt_pcie_device_configuration)::set(this,"root", "cfg",  root_cfg);
      root = svt_pcie_device_agent::type_id::create( "root", this );
```

```
      …
   endfunction
…
   function void setup_system_defaults (svt_pcie_device_configuration cfg);
      cfg.pcie_spec_ver = svt_pcie_device_configuration::PCIE_SPEC_VER_2_1;

      // Programming configuration attributes of the Applications
      cfg.driver_cfg[0].requester_id = 'h300;
      cfg.driver_cfg[0].percentage_use_tlp_digest = 50;

      cfg.target_cfg[0].completer_id = 'h300;
      cfg.target_cfg[0].percentage_use_tlp_digest = 50;
      cfg.target_cfg[0].max_read_cpl_data_size_in_bytes = 512;

      // Programming the configuration attributes of the PCIe protocol layers
      cfg.pcie_cfg.tl_cfg.credit_starvation_timeout = 8000;
      cfg.pcie_cfg.tl_cfg.completion_timeout = 400000;

      cfg.pcie_cfg.dl_cfg. updatefc_timeout_ns = 40000;

      cfg.pcie_cfg.pl_cfg.set_link_width_values(4);
      cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_2_5G);
      cfg.pcie_cfg.pl_cfg.skip_polling_active = 1;
   endfunction
endclass
```

12.2.2 Dynamic Configuration

The configuration set during the build of the UVM agent can be dynamically (during the `run_phase`) modified if the test requires a change in the VIP's configuration. The agent provides the following three ways to reconfigure the agent or its sub-components:

1. Using `svt_pcie_device_agent::reconfigure_via_task()`: This task can be called and a new configuration object with the desired programming can be specified as an input to this task.
2. Using `svt_pcie_device_agent::refresh_cfg()`: This task can be called to refresh the configuration of the agent or its sub-components based on a `uvm_config_db::set()` that is issued before calling `refresh_cfg()`.
3. Using the `REFRESH_CFG` service request: The PCIe Device Agent supports a service request interface via the service sequencer which is discussed under the sequencers sub-section. Using the service sequencer in the agent class a service request can be placed to refresh the configuration of the agent or sub-components based on a `uvm_config_db::set()` that is issued before requesting a `REFRESH_CFG` service.

It is important to note that before reconfiguring the driver application, you must make sure that the driver is idle. This will ensure the driver is not in the middle of a transaction request or waiting for the completion of a transaction request during the reconfiguration process. Also, the reconfiguration process should not be started when the model is in the reset state. The driver application (also the VIP or any of its sup-components) should not be reconfigured while the mode is in reset as defined in section Resetting the PCIe VIP.

The targeted modification can be on a specific layer of the VIP or across multiple layers depending on what the test is trying to achieve. In the Example 12-2, a typical use of dynamic reconfiguration is shown by using `reconfigure_vias_task()` task. In this example, the intent of reconfiguration is to cause the model to operate at a new PIPE width, PIPE clock rate and link speed after exiting `HOT_RESET`.

Example 12-2

```
class pcie_pipe_speed_width extends uvm_test;
   `uvm_component_utils(pcie_pipe_speed)
   …
   task run_phase (uvm_phase phase);
      svt_configuration temp_cfg = null;
      svt_pcie_device_configuration new_cfg = null;

      super.run_phase(phase);
      …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'
      // VIP's DL is enabled.
      // LTSSM gets operational at 2.5G and uses a PIPE model.
      // Link operates at default PCLK rate/PIPE width defined in the PL.
      // PCLK at 2.5G = 250MHz; PIPE width = 8-bits
      // PCLK at 5.0G = 500MHz; PIPE width = 8-bits
      // PCLK at 8.0G = 1000MHz; PIPE width = 8-bits
      // The test initiates a transition to take the LTSSM to HOT_RESET

      wait(env.root.status.pcie_status.pl_status.ltssm_state ==
svt_pcie_types::HOT_RESET);
       // After a timeout the LTSSM is expected to be in DETECT
          wait(env.root.status.pcie_status.pl_status.ltssm_state ==
svt_pcie_types::DETECT_QUIET);

       // Wait for VIP to be idle before changing configuration
      wait(env.root.status.pcie_status.pl_status.is_pipe_idle == 1 );

      // Fetch the current configuration of the PCIe device agent
      env.root.get_cfg_via_task(temp_cfg);
      $cast(new_cfg, temp_cfg.clone();

      // Program the new values for PCLK rate and PIPE width.
      // Note: the configuration variables are defined in the PL
      // Indices 0, 1 and 2 PCLK rate/PIPE width at Gen1, Gen2 and Gen3
      new_cfg.pcie_cfg.pl_cfg.pclk_rate[0] = svt_pcie_pl_configuration::PCLK_1000_MHZ;
      new_cfg.pcie_cfg.pl_cfg.pclk_rate[1] = svt_pcie_pl_configuration::PCLK_1000_MHZ;
      new_cfg.pcie_cfg.pl_cfg.pclk_rate[2] = svt_pcie_pl_configuration::PCLK_1000_MHZ;
      new_cfg.pcie_cfg.pl_cfg.pipe_width[0] = svt_pcie_pl_configuration::PIPE_8_BITS;
      new_cfg.pcie_cfg.pl_cfg.pipe_width[1] = svt_pcie_pl_configuration::PIPE_8_BITS;
      new_cfg.pcie_cfg.pl_cfg.pipe_width[2] = svt_pcie_pl_configuration::PIPE_8_BITS;

      // Reconfigure the VIP with the newly defined configuration
      new_cfg.pcie_cfg.pl_cfg.set_link_speed_values(`SVT_PCIE_SPEED_8_0G |
`SVT_PCIE_SPEED_5_0G | `SVT_PCIE_SPEED_2_5G);
      env.root.reconfigure_via_task(new_cfg);
   endtask
endclass
```

Similarly, a reconfiguration of the PCIe Device Agent or its sub-components can be performed using the `refresh_cfg()` function or by requesting a `REFRESH_CFG` service on the agent as illustrated in the Example 12-3.

Example 12-3

```
class pcie_pipe_speed_width extends uvm_test;
   `uvm_component_utils(pcie_pipe_speed)
   …
   task run_phase (uvm_phase phase);
      svt_configuration temp_cfg = null;
      svt_pcie_device_configuration new_cfg = null;
      svt_pcie_device_agent_service_sequence dev_agent_serv_seq;

      super.run_phase(phase);
      …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'

      // Did a number of things…

// About to change the configuration of the PCIe VIP

      // Assumptions:
      //   VIP is in DETECT_QUIET state
      //   pl_status.is_pipe_idle == 1

      // Fetch the current configuration of the PCIe device agent
      env.root.get_cfg_via_task(temp_cfg);
      $cast(new_cfg, temp_cfg.clone();

      // Modify members inside new_cfg
      // Call the uvm_config_db::set() to set the new configuration.
      uvm_config_db#(svt_pcie_device_configuration)::set(this,"env.root", "cfg",
new_cfg);

      // Refresh the agent with the new configuration (option 1)
      env.root.refresh_cfg();

         … OR …

      // Refresh the agent with the new configuration (option 2)
      dev_agent_serv_seq = new();
      dev_agent_serv_seq.service_type = svt_pcie_device_agent_service::REFRESH_CFG;
      dev_agent_serv_seq.start(env.root.device_agent_service_seqr);

      // Do other things
   endtask
endclass
```

12.3 Status

The PCIe Device Agent provides a set of state values representing the status of its sub-components at anytime in the test simulation. Table 12-2 contains status class members. And these state values are encapsulated within the `svt_pcie_device_status` class. The device agent class has an object of type `svt_pcie_device_status` instanced as status. The members of class `svt_pcie_device_status` are listed in the following table.

Table 12-2 Status Class Members

Class Objects	Member	Description
svt_pcie_driver_app_status	driver_status [$]	Status of Driver application
svt_pcie_io_target_status	io_target_status	Status of IO Target application
svt_pcie_mem_target_status	mem_target_status	Status of Mem Target application
svt_pcie_status	pcie_status	Status of MAC, that is 3 layer stack Transaction layer, Data link layer and Physical layer.
svt_pcie_requester_app_status	requester_status	Status of Requester application
svt_pcie_target_app_status	target_status [$]	Status of Target application

A test can access the status object of the device agent in the following two ways:

1. Directly accessing object `svt_pcie_device_agent::status` as illustrated in Example 12-2 where the test check for the LTSSM state as a control variable.
2. The test can use the `svt_pcie_device_agent::get_device_status()` function to retrieve the status of the agent and use a local copy of the status object as shown in Example 12-4.

Note
The `get_device_status` function takes an input which is passed by reference.

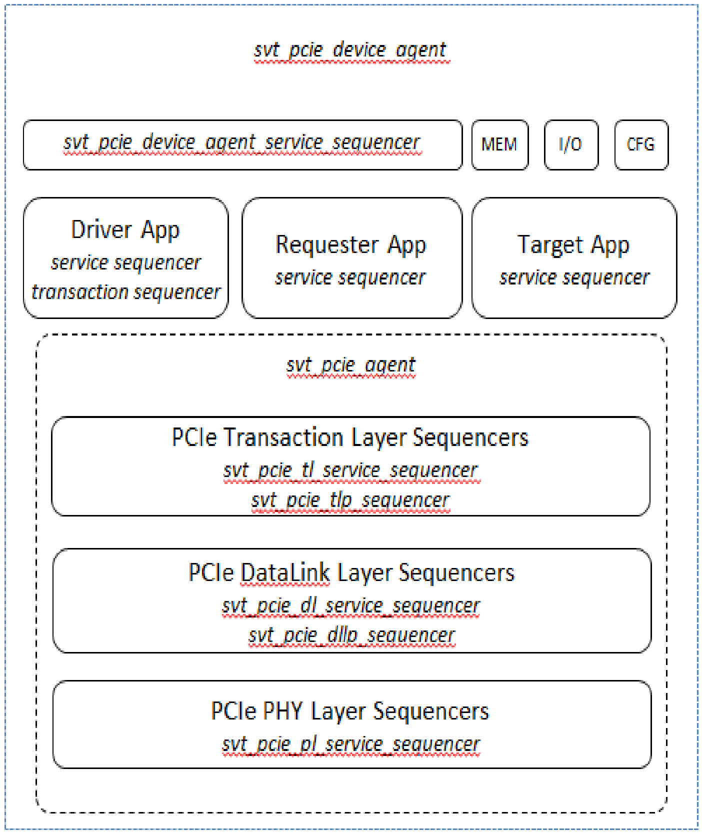
Example 12-4

```
class pcie_pipe_speed_width extends uvm_test;
   `uvm_component_utils(pcie_pipe_speed)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_device_status root_dev_status;
      super.run_phase(phase);
      …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'

      env.root.get_device_status(root_dev_status);

      // Use root_device_status to check the current status of any layer.
      // An example, LTSSM state value.

      // Do other things
   endtask
endclass
```

12.4 Sequencers

The PCIe Device Agent class (`svt_pcie_device_agent`) has eight UVM sequencer objects to schedule transaction requests or service requests on any of its subcomponent Drivers. A UVM sequencer is an arbiter that controls the transaction flow from multiple stimulus generators. The sequencers communicate with drivers using the TLM interfaces.

Figure 12-2 Block Diagram – Sequencers



Note

Blocks named Mem, I/O and CFG are service sequencers corresponding to memory target, I/O target and configuration database UVM drivers.

Sequencers in the PCIe VIP are broadly classified as service sequencers and transaction sequencers.

12.4.1 Service Sequencers

A service sequencer is used to schedule sequences that are referred to as services on any UVM driver in the device agent class. An example of a service request can be a request on the memory target to write/read data in an attempt to load/store the completion memory of the VIP. Service sequences will not generate PCIe transactions on the PCIe bus but they are used to request a change in the behavior or sample a state value of the UVM driver.The device agent class includes the following seven service sequencer objects:

1. Configuration Database Service Sequencer (Type=`svt_pcie_cfg_database_service_sequencer`, Instance=`cfg_database_seqr`): A sequencer used to schedule services such as, backdoor write/read services to configuration database. It feeds service transactions of type `svt_pcie_cfg_database_service` to Sequencer Item Pull Port (SIPP) `uvm_seq_port` of UVM driver `svt_pcie_cfg_database` class.
2. PCIe Device Agent Service Sequencer (Type=`svt_pcie_device_agent_service_sequencer`, Instance=`device_agent_service_seqr`): A sequencer used to schedule services such as, refresh configuration of the agent. It feeds service transactions of type `svt_pcie_device_agent_service` to SIPP `device_agent_service_seq_item_port` of UVM agent `svt_pcie_device_agent` class.

3. Driver Application Service Sequencer (Type=`svt_pcie_driver_app_service_sequencer`, Instance=`driver_seqr[0]`): A sequencer used to schedule services such as, applying reset to the driver application. It feeds service transactions of type `svt_pcie_driver_service` to the SIPP `service_seq_item_port` of UVM driver `svt_pcie_driver_app` class.
4. IO Target Service Sequencer (Type=`svt_pcie_io_target_service_sequencer`, Instance=`io_target_seqr`): A sequencer used to schedule services such as, backdoor write/read services to the I/O completion space of the VIP. It feeds service transactions of type `svt_pcie_io_target_service` to the SIPP `seq_item_port` of UVM driver `svt_pcie_io_target` class.
5. Memory Target Service Sequencer (Type=`svt_pcie_mem_target_service_sequencer`, Instance=`mem_target_seqr`): A sequencer used to schedule services such as, backdoor write/read services to the memory completion space of the VIP. It feeds service transactions of type `svt_pcie_mem_target_service` to the SIPP `seq_item_port` of UVM driver `svt_pcie_mem_target` class.
6. Requester Application Service Sequencer (Type=`svt_pcie_requester_app_service_sequencer`, Instance=`requester_seqr`): A sequencer used to schedule services such as starting the requester application. It feeds service transactions of type `svt_pcie_requester_app_service` to the SIPP `seq_item_port` of UVM driver `svt_pcie_requester_app` class.
7. Target Application Service Sequencer (Type=`svt_pcie_target_app_service_sequencer`, Instance=`target_seqr[0]`): A sequencer used to schedule services such as, starting the requester application. It feeds service transactions of type `svt_pcie_requester_app_service` to the SIPP `seq_item_port` of UVM driver `svt_pcie_requester_app` class.

Example 12-5 shows how you can request a service on the driver application using the driver application service sequencer. The requested service is to wait for the idle state of the driver application.

Example 12-5

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_driver_app_service_wait_until_idle_sequence drv_app_serv_seq;
         …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'
      // VIP's DL is enabled.
      // LTSSM is in L0
      // the test generated a number of transaction requests using the driver
application
      drv_app_serv_seq = new();
      drv_app_serv_seq.start(env.root.driver_seqr[0]);
      // End of test. The driver app is idle and so all transaction requests are done.
   endtask
endclass
```

12.4.2 Transaction Sequencers

A transaction sequencer is used to schedule sequences that cause PCIe transactions (TLPs and DLLPs) on the PCIe bus. There is only a single transaction sequencer object in the device agent class:

- Driver Application Transaction Sequencer
  (Type=`svt_pcie_driver_app_transaction_sequencer`, `Instance=driver_transaction_seqr[0]`): A sequencer used to schedule TLP transactions on the driver application of the VIP. It feeds transactions of type `svt_pcie_driver_app_transaction` to the SIPP `seq_item_port` of UVM driver class `svt_pcie_driver_app`.

Example 12-6 shows how you can send a transaction request to the driver using the driver application transaction sequencer.

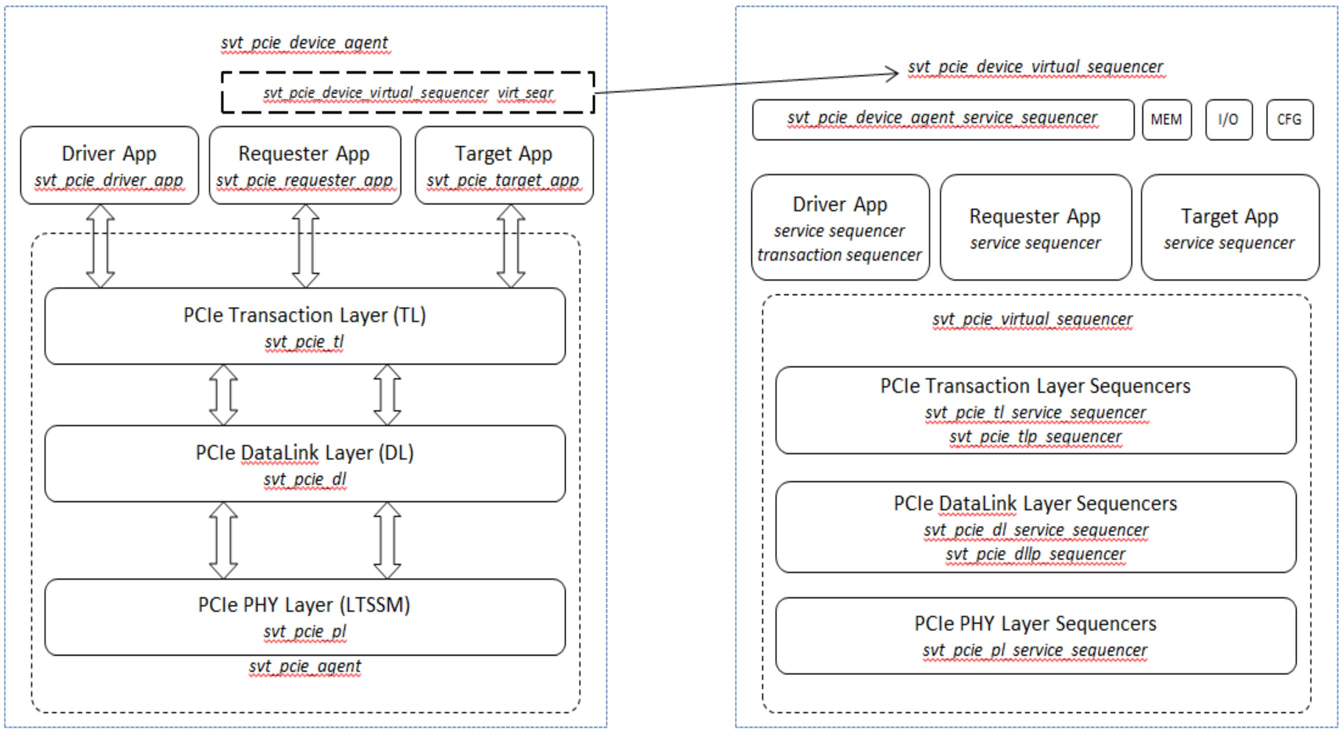
Example 12-6

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_driver_app_transaction_mem_write_sequence mem_wr_seq;
         …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'
      // VIP's DL is enabled.
      // LTSSM is in L0
      mem_wr_seq = new();
      mem_wr_seq.randomize with {
         address == 'h8000;
         length == 2;
         first_dw_be == 0;
         last_dw_be == 0;
         traffic_class == 0;
         address_translation == 2'b00;
         foreach(payload[i])
           payload[i] == 'hc0de_0000 + i;
      };
      mem_wr_seq.start(env.root.driver_transaction_seqr[0]);
      // Add end of test criteria.
      // End of test.
   endtask
endclass
```

12.4.3 Virtual Sequencer

The device agent class has a virtual sequencer object of type `svt_pcie_device_virtual_sequencer` instanced as `virt_seqr` that connects to all sequencers that are defined under its hierarchy. The sequencer class object has a hierarchical structure that mirrors `svt_pcie_device_agent` class. Instead of having UVM drivers/agents as sub-components, it has the UVM sequencer of the corresponding UVM driver/agent.

Figure 12-3 Block Diagram – Virtual Sequencer



Note

Blocks named Mem, I/O and CFG are service sequencers corresponding to memory target, I/O target and configuration database UVM drivers.

The example that illustrates the use of the driver application service sequencer and driver application transaction sequencer can alternatively access these sequencers via the virtual sequencer instance as illustrated in Example 12-7 and Example 12-8.

Example 12-7

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_driver_app_service_wait_until_idle_sequence drv_app_serv_seq;
         …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'
      // VIP's DL is enabled.
      // LTSSM is in L0
      // the test generated a number of transaction requests using the driver
application
      drv_app_serv_seq = new();
      drv_app_serv_seq.start(env.root.virt_seqr.driver_seqr[0]);
      // End of test. The driver app is idle and so all transaction requests are done.
   endtask
endclass
```

Example 12-8

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_driver_app_transaction_mem_write_sequence mem_wr_seq;
         …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'
      // VIP's DL is enabled.
      // LTSSM is in L0
      mem_wr_seq = new();
      mem_wr_seq.randomize with {
         address == 'h8000;
         length == 2;
         first_dw_be == 0;
         last_dw_be == 0;
         traffic_class == 0;
         address_translation == 2'b00;
         foreach(payload[i])
           payload[i] == 'hc0de_0000 + i;
      };
      mem_wr_seq.start(env.root.virt_seqr.driver_transaction_seqr[0]);
      // Add end of test criteria.
      // End of test.
   endtask
endclass
```

The examples above shows the virtual sequencer as an alternative mechanism for generating services and transactions on the driver application without showcasing its real value. The real value of the virtual sequencer is seen when the PCIe device agent is part of a test environment that has other agents that drive stimulus to other possible interfaces on the DUT. As an example, consider a system-level environment that has both PCIe and USB VIP's being used to drive stimulus to the PCIe and USB interfaces of the SoC. To simplify test writing, a system wide virtual sequencer class can be defined to access both the PCIe VIP and USB VIP sequencers. And this system wide sequencer can be defined in the system environment and connected to the PCIe and USB VIP agents as shown in the Example 12-9.

Example 12-9

```
class my_system_virtual_sequencer extends uvm_sequencer;
   …
   svt_pcie_device_virtual_sequencer pcie_dev_virt_seqr;
   usb_device_virt_sequencer usb_virt_seqr;

   function new(…);
      …
   endfunction

endclass
class my_system_env extends uvm_env;
   svt_pcie_device_agent root;
   usb_device_agent usb_dev;
   my_system_virtual_sequencer sys_virt_seqr;

   …
```

```
    function void build_phase(uvm_phase phase);
       super.build_phase(phase);

       …
       this.sys_virt_seqr = my_system_virtual_sequencer::create("sys_virt_seqr", this);

    endfunction

    function void connect_phase(uvm_phase phase);
       super.connect_phase(phase);
       this.sys_virt_seqr.pcie_dev_virt_seqr = root.virt_seqr;
       this.sys_virt_seqr.usb_virt_seqr = usb_dev.virt_seqr;

       …
    endfunction
 endclass
```

With this system-level sequencer defined, the UVM test will have a single reference to all sequencers that are part of the system. The test can drive stimulus to both the PCIe and USB interface by being oblivious to the agents or the hierarchies under them.

Example 12-10

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_driver_app_transaction_mem_write_seqeunce pcie_wr_seq;
      usb_device_transaction_sequence usb_tr_seq;
         …

      pcie_wr_seq = new();
      pcie_wr_seq.randomize with {
         address == 'h8000;
         length == 2;
         first_dw_be == 4'b1111;
         last_dw_be == 4'b1111;
         traffic_class == 0;
         address_translation == 2'b00;
         foreach(payload[i])
           payload[i] == 'hc0de_0000 + i;
      };

      usb_tr_seq = new();
      usb_tr.randomize with {
         …
      };

      fork
pcie_wr_seq.start(env.sys_virt_seqr.pcie_dev_virt_seqr.driver_transaction_seqr[0]);
      usb_tr.start(env.sys_virt_seqr.usb_virt_seqr.ss_pkt_seqr);
      join
      // Add end of test criteria.
      // End of test.
   endtask
endclass
```

Note
The USB VIP agent used in the example above is purely hypothetical. It does not represent the USB VIP product from Synopsys or any other vendor.

13 PCIe Agent

This chapter discusses the following topics:

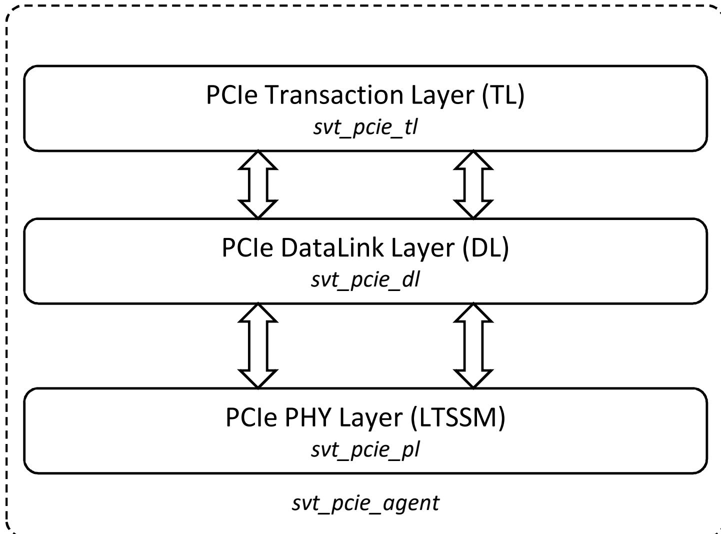
- Overview
- Configuration
- Status
- Sequencers

13.1 Overview

The PCIe Agent encapsulates the UVM drivers that represent the layered stack specified in the PCIe specification. Class `svt_pcie_agent` represents this encapsulation in the PCIe VIP. The PCIe agent class is instanced as `pcie_agent` within `svt_pcie_device_agent`. The agent class consists of the following layers:

1. The Transaction Layer (Type=`svt_pcie_tl`, Instance=`pcie_tl`): Class `svt_pcie_tl` defines the functions of the transaction layer (TL) in the PCIe VIP. The applications transfer transaction requests to the TL for transmission. The TL composes the transaction layer packet (TLP) and hands it down to the data-link layer located below it. The transaction layer also receives TLPs from the data-link layer which gets routed up to the correct application. It is also possible for the test to interface directly with the TL to generate TLPs. But it is recommended to use the application layers.
2. The Data-link Layer (Type=`svt_pcie_dl`, Instance=`pcie_dl`): Class `svt_pcie_dl` defines the functions of the data-link (DL) layer in the PCIe VIP. The TL transfers TLPs to the DL to be framed with a sequence number and a CRC and ensure the remote receiver receives the packet without any errors. The DL also performs the other standard functions of link management using data-link layer packets (DLLPs).
3. The Physical Layer (Type=`svt_pcie_pl`, Instance=`pcie_pl`): Class `svt_pcie_pl` defines the functions of the physical layer (PL) in the PCIe VIP. The PL breaks down packets it receives (from the DL) into symbols and encodes them as per the specification before transmission. It also composes packets from data received on the receive data lanes and sends them back to the DL. It also performs other functions to maintain the link such as the LTSSM and functions for low power and so on.

Figure 13-1 Block Diagram – PCIe Agent



13.2 Configuration

The PCIe Agent is configured using an object of class type `svt_pcie_configuration`. An object of this type with an instance name of `pcie_cfg` is defined in `svt_pcie_device_configuration` class. This class is comprised of data members and class object members. The object members represent the configuration of sub-components of the PCIe Agent class.

```
svt_pcie_device_configuration
                |
                |--------> driver_cfg[]  (type=svt_pcie_driver_app_configuration)
                |
                |--------> requester_app (type=svt_pcie_requester_app_configuration)
                |
                |--------> target_cfg[]  (type=svt_pcie_target_app_configuration)
                |
                |--------> pcie_cfg      (type=svt_pcie_configuration)
                                   |
                                   |--------> tl_cfg (type=svt_pcie_tl_configuration)
                                   |
                                   |--------> dl_cfg (type=svt_pcie_dl_configuration)
                                   |
                                   |--------> pl_cfg (type=svt_pcie_pl_configuration)
```

The `svt_pcie_configuration` class also has data members that define the behavior of the agent. For example, `svt_pcie_configuration::enable_cov` is a configuration variable that enables functional coverage.

Note

`svt_pcie_configuration::enable_cov` is a 6-bit variable and each bit enables a specific kind of coverage. For details about this variable and other variables, see HTML class description of the `svt_pcie_configuration` class available at the following location: *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_configuration.html*

13.2.1 Initial Configuration

The initial configuration of the `svt_pcie_agent` or its sub-components is set as illustrated in "Initial Configuration" of the PCIe Device Agent section. The configuration attributes defined within the `svt_pcie_device_configuration::pcie_agent` and its sub-configuration objects can be programmed before the `uvm_config_db::set()` call illustrated in Example 12-2.

13.2.2 Dynamic Configuration (reconfiguration)

Dynamic configuration changes to the configuration of the `svt_pcie_agent` or its sub-components can be made as defined in the "Dynamic Configuration" of the PCIe Device Agent section. The members of `svt_pcie_device_agent::pcie_cfg` can be modified before the calls to reconfigure the configuration of the device agent as illustrated in Example 12-2 and Example 12-3.

13.3 Status

The PCIe Agent has a set of state values representing the status of its constituent blocks at any given time in the test simulation. And these state values are encapsulated within the `svt_pcie_status` class. The `svt_pcie_device_status` class has an object of type `svt_pcie_status` instanced as `pcie_status`. The members of class `svt_pcie_status` are listed in the table below.

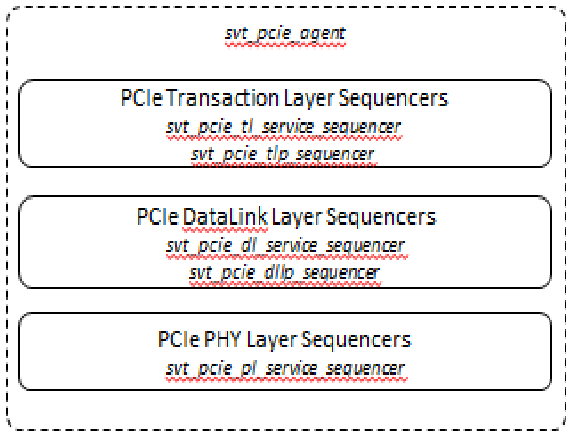
Class Objects	Member	Description
svt_pcie_config_space_status	config_space_status	Status class for tracking observed PCIe configuration space transactions
svt_pcie_dl_status	dl_status	Status of Link layer
svt_pcie_pl_status	pl_status	Status of Physical layer
svt_pcie_tl_status	tl_status	Status of Transaction layer

The test can access these state variables as described by the status section of the PCIe device agent class. Example 12-2 referenced by that section uses `svt_pcie_device_status::status.pl_status.ltssm_state` as a control variable.

13.4 Sequencers

The PCIe Agent class (`svt_pcie_agent`) has six UVM sequencer objects to schedule transaction requests or service requests on any of its subcomponent drivers. A UVM sequencer is an arbiter that controls the transaction flow from multiples stimulus generators. The sequencers communicate with drivers using TLM interfaces.

Figure 13-2 Block Diagram – Sequencers



Sequencers in the PCIe Agent are broadly classified as service sequencers and transaction sequencers.

13.4.1 Service Sequencers

A Service Sequencer is used to schedule sequences that are referred to as services on any UVM driver in the PCIe Agent class. An example of a service request can be a request on the Physical layer to initiate a link width change. Service sequences will not generate PCIe transactions on the PCIe bus, but they are used to request a change in the behavior or sample a state value of the UVM driver. The PCIe Agent class includes the following three service sequencer objects:

1. Data-link Layer Service Sequencer (Type=`svt_pcie_dl_service_sequencer`, Instance=`dl_seqr`): A sequencer used to schedule services such as enabling of the Data-link layer driver. It feeds service transactions of type `svt_pcie_dl_service` to SIPP (Sequencer Item Pull port) `seq_item_port` of UVM driver `svt_pcie_dl` class.
2. Physical Layer Service Sequencer (Type=`svt_pcie_pl_service_sequencer`, Instance=`pl_seqr`): A sequencer used to schedule services such as speed change on the Physical layer driver. It feeds service transactions of type `svt_pcie_pl_service` to SIPP (Sequence Item Pull Port) `seq_item_port` of UVM driver `svt_pcie_pl` class.
3. Transaction Layer Service Sequencer (Type=`svt_pcie_tl_service_sequencer`, instance=`tl_seqr`): A sequencer used to schedule services such as setting up a traffic class map. It feeds service transactions of type `svt_pcie_tl_service` to the SIPP (Sequence Item Pull Port) `service_seq_item_port` of UVM driver `svt_pcie_tl` class.

Example 13-1 shows how you can request a service on the TL via the TL service sequencer. The requested service is to map a given traffic class to a VC.

Example 13-1

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_tl_service_set_tc_map_sequence tl_serv_seq;
      …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
```

```
      // The UVM environment has an instance of the PCIe device agent named 'root'
      tl_serv_seq = new();

      // Enable TC value 1. By default only TC=0 is enabled and mapped to VC0
      // Map TC=1 to VC1
      tl_serv_seq.tc_enable = 1;
      tl_serv_seq.tc_num    = 1;
      tl_serv_seq.vc_num    = 1;
      tl_serv_seq.start(env.root.pcie_agent.tl_seqr);

      // Map TC=2 to VC2
      tl_serv_seq.tc_enable = 1;
      tl_serv_seq.tc_num    = 2;
      tl_serv_seq.vc_num    = 2;
      tl_serv_seq.start(env.root.pcie_agent.tl_seqr);

      // Map TC=2 to VC2
      tl_serv_seq.tc_enable = 1;
      tl_serv_seq.tc_num    = 3;
      tl_serv_seq.vc_num    = 3;
      tl_serv_seq.start(env.root.pcie_agent.tl_seqr);

      Etc.,

      // Generate transaction requests.

      // End of test. Check for an idle state of testbench blocks before ending test.
   endtask
endclass
```

13.4.2 Transaction Sequencers

A transaction sequencer is used to schedule sequences that cause PCIe transactions (TLPs, DLLPs) on the PCIe bus. The two transaction sequencers in the PCIe Agent class are as follows:

1. TLP Transaction Sequencer (Type= `svt_pcie_tlp_sequencer`, Instance=`tlp_seqr`): A sequencer used to schedule TLP transactions on the transaction layer of the VIP. It feeds transactions of type `svt_pcie_tlp` to the SIPP (Sequence Item Pull Port) `seq_item_port` of UVM driver class `svt_pcie_tl`.
2. DLLP Transaction Sequencer (Type=`svt_pcie_dllp_sequencer`, Instance=`dllp_seqr`): A sequencer used to schedule DLLP transactions on the data-link layer of the VIP. It feeds transactions of type `svt_pcie_dllp` to the SIPP (Sequence Item Pull Port) `seq_item_port` of UVM driver class `svt_pcie_dl`.

Example 13-2 shows how you can schedule a TLP transaction request to the TL using the TLP sequencer.

Example 13-2

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_tlp_mem_request_sequence mem_tlp_seq;
         …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'
```

```
        // VIP's DL is enabled.
        // LTSSM is in L0
        mem_tlp_seq = new();
        mem_tlp_seq.randomize with {
           address == 'h8000;
           length == 2;
           requester_id == 'h100;
           tlp_type == svt_pcie_tlp::MEM_REQ;
           fmt == svt_pcie_tlp::WITH_DATA_3_DWORD;
           // Program other header fields.
            …
           foreach(payload[i])
             payload[i] == 'hc0de_0000 + i;
        };
        mem_tlp_seq.start(env.root.pcie_agent.tlp_seqr);
        // Add end of test criteria.
        // End of test.
     endtask
  endclass
```

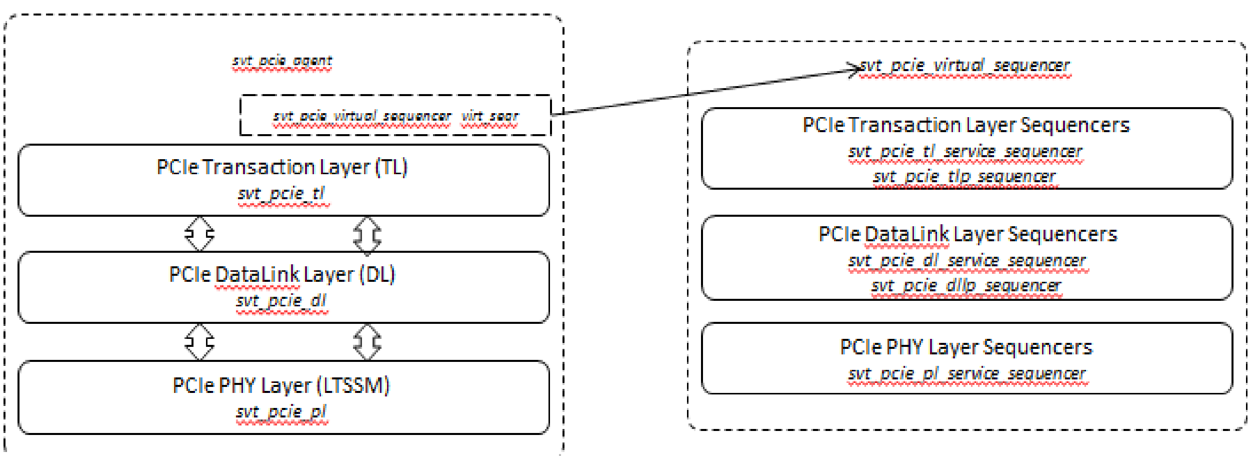
Note

Testbench should constrain the `tlp_type` as per device configuration. If a sequencer requests for TLP from Root Complex, then testbench must generate only those transactions which are valid from Root Complex based on the value of `device_is_root` variable in `svt_pcie_device_configuration` object. And same rule should also apply for Endpoint.

13.4.3 Virtual Sequencer

The PCIe Agent class has a virtual sequencer object of type `svt_pcie_virtual_sequencer` instanced as `virt_seqr` that connects to all sequencers that are defined under its hierarchy. The sequencer class object has a hierarchical structure similar to the `svt_pcie_agent` class. Instead of having UVM drivers/agents as subcomponents, it has the UVM sequencer of the corresponding UVM drivers/agents.

Figure 13-3 Block Diagram – Virtual Sequencer



The example that illustrates the use of the TL service sequence and TLP transaction sequencer can alternatively access these sequencers via the virtual sequencer instance as illustrated in Example 13-3 and Example 13-4.

Example 13-3

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_tl_service_set_tc_map_sequence tl_serv_seq;
         …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'
      tl_serv_seq = new();

      // Enable TC value 1. By default only TC=0 is enabled and mapped to VC0
      // Map TC=1 to VC1
      tl_serv_seq.tc_enable = 1;
      tl_serv_seq.tc_num    = 1;
      tl_serv_seq.vc_num    = 1;
      tl_serv_seq.start(env.root.pcie_agent.virt_seqr.tl_seqr);

      // Map TC=2 to VC2
      tl_serv_seq.tc_enable = 1;
      tl_serv_seq.tc_num    = 1;
      tl_serv_seq.vc_num    = 1;
      tl_serv_seq.start(env.root.pcie_agent.virt_seqr.tl_seqr);

      // Map TC=2 to VC2
      tl_serv_seq.tc_enable = 1;
      tl_serv_seq.tc_num    = 1;
      tl_serv_seq.vc_num    = 1;
      tl_serv_seq.start(env.root.pcie_agent.virt_seqr.tl_seqr);

      Etc.,

      // Generate transaction requests.

      // End of test. Check for an idle state of testbench blocks before ending test.
   endtask
endclass
```

Example 13-4

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_tlp_mem_request_sequence mem_tlp_seq;
         …
      // Assumptions:
      // The UVM test has an instance of UVM environment named 'env'
      // The UVM environment has an instance of the PCIe device agent named 'root'
      // VIP's DL is enabled.
      // LTSSM is in L0
      mem_tlp_seq = new();
      mem_tlp_seq.randomize with {
```

```
            address == 'h8000;
            length == 2;
            requester_id == 'h100;
            tlp_type == svt_pcie_tlp::MEM_REQ;
            fmt == svt_pcie_tlp::WITH_DATA_3_DWORD;
            // Program other header fields.
             …
            foreach(payload[i])
              payload[i] == 'hc0de_0000 + i;
         };
         mem_tlp_seq.start(env.root.pcie_agent.virt_seqr.tlp_seqr);
         // Add end of test criteria.
         // End of test.
      endtask
   endclass
```

Example 13-3 and Example 13-4 illustrate the use of the `svt_pcie_agent::virt_seqr` to access sequencers that are part of `svt_pcie_agent` class. But it finds practical usage in cases where the PCIe VIP agent is part of a test environment that has other agents that drive stimulus to other possible interfaces on the DUT. With such a system-level testbench, a system wide virtual sequencer class can be defined to access both PCIe VIP and sequencers that are part of other VIP agents. And this system wide sequencer can be defined in the system environment and connected to the sequencers of the PCIe VIP and sequencers of other VIP agents. Example 13-5 shows the typical usage.

Example 13-5

```
class my_system_virtual_sequencer extends uvm_sequencer;
   …
   svt_pcie_device_virtual_sequencer pcie_dev_virt_seqr;
   usb_device_virt_sequencer usb_virt_seqr;

   function new(…);
      …
   endfunction
endclass
class my_system_env extends uvm_env;
   svt_pcie_device_agent root;
   usb_device_agent usb_dev;
   my_system_virtual_sequencer sys_virt_seqr;

   …

   function void build_phase(uvm_phase phase);
      super.build_phase(phase);

      …
      this.sys_virt_seqr = my_system_virtual_sequencer::create("sys_virt_seqr", this);

   endfunction

   function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
      this.sys_virt_seqr.pcie_dev_virt_seqr = root.virt_seqr;
      this.sys_virt_seqr.usb_virt_seqr = usb_dev.virt_seqr;

      …
   endfunction
endclass
```

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_tlp_mem_request_sequence mem_tlp_seq;
      usb_device_transaction_sequence usb_tr_seq;
         …

      mem_tlp_seq = new();
      mem_tlp_seq.randomize with {
         address == 'h8000;
         length == 2;
         requester_id == 'h100;
         tlp_type == svt_pcie_tlp::MEM_REQ;
         fmt == svt_pcie_tlp::WITH_DATA_3_DWORD;
         // Program other header fields.
          …
         foreach(payload[i])
           payload[i] == 'hc0de_0000 + i;
      };

      usb_tr_seq = new();
      usb_tr.randomize with {
         …
      };

      fork
      mem_tlp_seq.start(env.sys_virt_seqr.pcie_dev_virt_seqr.pcie_virt_seqr.tlp_seqr);
      usb_tr.start(env.sys_virt_seqr.usb_virt_seqr.ss_pkt_seqr);
      join
      // Add end of test criteria.
      // End of test.
   endtask
endclass
```

Note

The USB VIP agent used in the example above is purely hypothetical. It does not represent the USB VIP product from Synopsys or any other vendor.

Note

Example 13-5 accesses the TLP sequencer using the device agent virtual sequencer PCIe Device agent (`svt_pcie_device_agent`) class `svt_pcie_device_agent::virt_seqr.pcie_virt_seqr.tlp_seqr`. And this is same as accessing the TLP sequencer via the virtual sequencer defined inside `svt_pcie_agent` class `svt_pcie_agent::virt_seqr.tlp_seqr`. The latter is illustrated in Example 13-4.

14 Using the Transaction Layer

14.1 Transaction Layer

The Transaction Layer, TL, is implemented as a uvm_driver, svt_pcie_tl. The TL contains a configuration object, svt_pcie_tl_configuration (see "Transaction Layer Configuration"), a service sequencer (see "Transaction Layer Sequencer and Sequences"), which work together to setup and control the behavior of the TL. Additionally, the TL offers callbacks (see "Transaction Layer Callbacks and Exceptions"), with exception capability (see "Transaction Layer Callbacks and Exceptions"), as well as a status object "Transaction Layer Status".

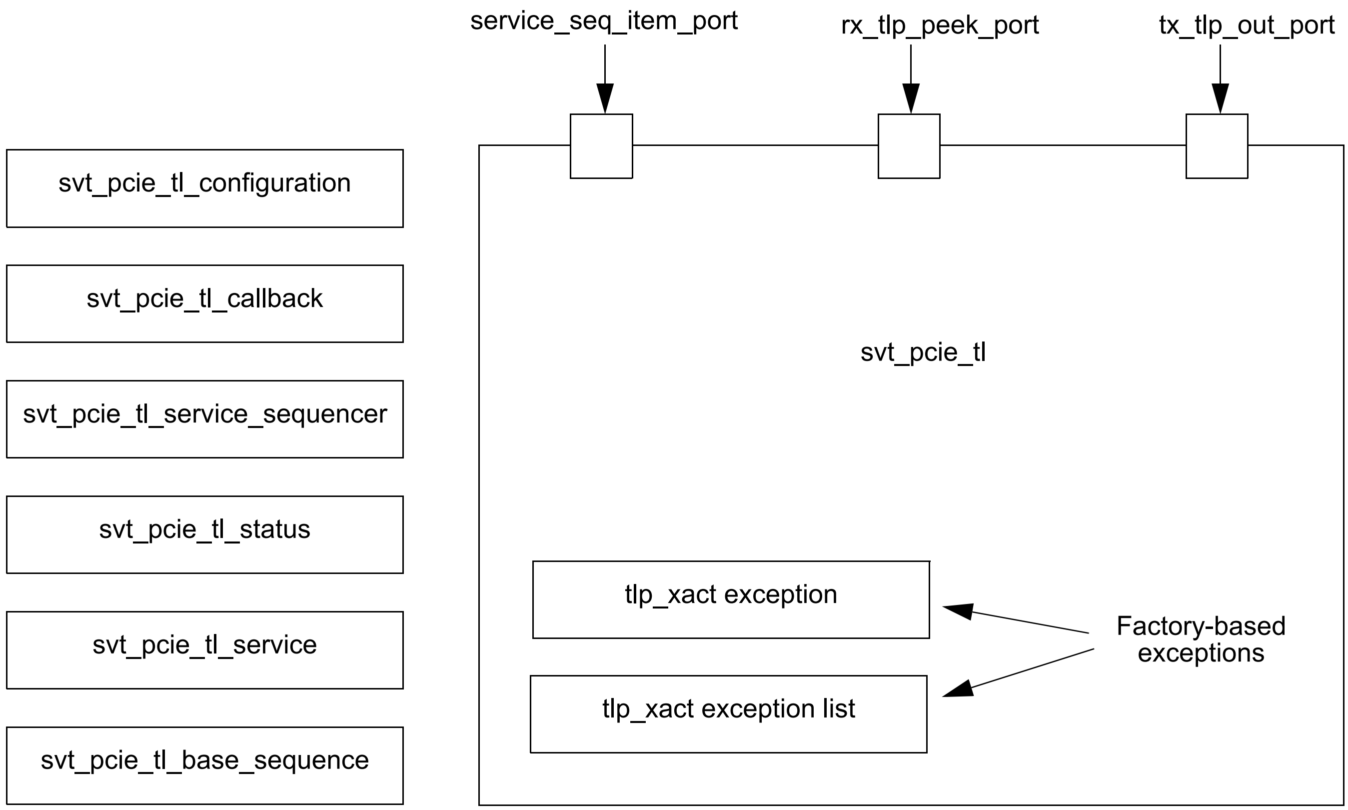
Attention Note, the descriptions for the classes and applications show the most important and often used members/features. Consult the PCIe UVM HTML Class Reference for a complete listing of the members and their data types.

Consult the following to get information on TLP related programming tasks.

- "SolvNet PCIe VIP Articles" on page 533.
- PCIe SVT FAQ

A block diagram of the Transaction Layer elements is shown in Figure 14-1.

Figure 14-1 Transaction Layer block diagram



The VIP TL layer is analogous to that of the transaction Layer of the PCIe specification. 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.

Note

If you use Virtual Channels other than VC0, those Virtual Channels must be initialized. To initialize VC1-VC7, credits must be initialized. Use svt_pcie_tl_configuration::init*_tx_credits followed by a call to svt_pcie_tl_service_tl_set_vc_en_sequence to set up the VCs.

14.2 Transaction Layer Configuration

The transaction layer configuration class is svt_pcie_tl_configuration. The members within the class control the settings of the Transaction Layer, TL.

The class is accessed via the following instance hierarchy:

instance name of svt_pcie_device_configuration.pcie_cfg.tl_cfg

Members of the svt_pcie_tl_configuration class are listed in Table 14-1.

Table 14-1 Transaction Layer configuration members

svt_pcie_tl_configuration::auto_enable_vc0_at_startup Type: rand bit Range: 0-1 Default: 1 Description: VC0 automatically initializes when enabled.
svt_pcie_tl_configuration::credit_starvation_timeout_ns Type: rand int unsigned Range: 0+ ns Default: 10000ns Description: 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.
svt_pcie_tl_configuration::default_route_at_appl_id Type: rand int unsigned Default: 0 Description: Route address translation (AT).
svt_pcie_tl_configuration::default_route_cfg_type0_appl_id Type: rand int unsigned Default: 0 Description: Default application ID to use.
svt_pcie_tl_configuration::default_route_cfg_type1_appl_id Type: rand int unsigned Default: 0 Description: Default application ID to use.
svt_pcie_tl_configuration::default_route_io_appl_id Type: rand int Default: 0 Description: Default application ID to use.
svt_pcie_tl_configuration::default_route_mem_appl_id Type: rand int unsigned Default:0 Description: Default application ID to use.

Table 14-1 Transaction Layer configuration members (Continued)

svt_pcie_tl_configuration::default_route_msg_appl_id Type: rand int unsigned Default: 0 Description: Application ID.
svt_pcie_tl_configuration::enable_route_at_to_function Type: rand bit Default: 0 Description: Routing to function address translation (AT).
svt_pcie_tl_configuration::enable_route_cfg_type0_to_function Type: rand bit Default: 0 Description: Enable routing type 0 configuration requests.
svt_pcie_tl_configuration::enable_route_cfg_type1_to_function Type: rand bit Default: 0 Description: Enable routing type1 configuration requests.
svt_pcie_tl_configuration::enable_route_io_to_function Type: rand bit Default: 0 Description: Enable routing I/O requests.
svt_pcie_tl_configuration::enable_route_mem_to_function Type: rand bit Default: 0 Description: Enable routing memory requests.
svt_pcie_tl_configuration::enable_route_msg_to_function Type: rand bit Default: 0 Description: Enable routing message requests.
svt_pcie_tl_configuration::init_[cpl|np|p]_data_tx_credits Type: rand int unsigned [8] Range: 0-4096 Default: 1025 Description: Set initial data credits.

Table 14-1 Transaction Layer configuration members (Continued)

svt_pcie_tl_configuration::init_[cpl|np|p]_hdr_tx_credits Type: rand int unsigned [8] Range: 0-255 Default: 101 Description: Set initial header credits.
svt_pcie_tl_configuration::max_vc[0-7]_[p|np|cpl]_updatefc_delay Type: rand int unsigned Range: 0+ ns Default: 1 ns Description: Maximum flow control (FC) delay.
svt_pcie_tl_configuration::min_vc[0-7]_[p|np|cpl]_updatefc_delay Type: rand int unsigned Range: 0+ ns Default: 100 ns Description: Minimum flow control (FC) delay
svt_pcie_tl_configuration::remote_extended_tag_field_enabled Type: rand int unsigned Range: 0-1 Default: 0 Description: Enable remote extended tag.
svt_pcie_tl_configuration::remote_max_read_request_size Type: rand int unsigned Range: 0-4096 Default: 512 Description: Remote max read request size.

14.2.1 Verilog Configuration Parameters

Not all configuration items are currently controlled from within the UVM interface. At this time the items in “Compile-time Verilog Parameters” and “Runtime-changeable Verilog Parameters” are controlled only via Verilog.

14.2.1.1 Compile-time Verilog Parameters

Parameters that are only changeable when instantiating the TL as part of the instantiation model are listed in Table 14-2.

Table 14-2 Transaction Layer runtime Verilog parameters

Parameter Name	Type	Range	Default Value	Description
DEFAULT_ROUTE_AT_APPL_ID				
	Integer	0 - large value	0	Default Application ID to route Address Translation requests to.
NUM_APPL_ID				
	Integer	8-128	8	Max number of unique Application IDs. IDs assigned to applications must be less than this value.
RID_APPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique RID to Appl_id map entries.
RID_MSGCODE_APPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique {RID,msgcode} to Appl_id map entries.
MEM_ADDR_ADDPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique Mem Address to Appl_id map entries.
IO_ADDR_ADDPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique I/O Address to Appl_id map entries.
AT_ADDR_ADDPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique AT Address to Appl_id map entries.

14.2.1.2 Runtime-changeable Verilog Parameters

Transaction Layer parameters that are changeable at runtime are listed in Table 14-3.

Table 14-3 Transaction Layer runtime Verilog parameters

Parameter Name	Type	Range	Default Value	Description
MAX_NUM_END_TO_END_PREFIXES				
	Integer		4	Max number of prefixes allowed. Version 3 only.

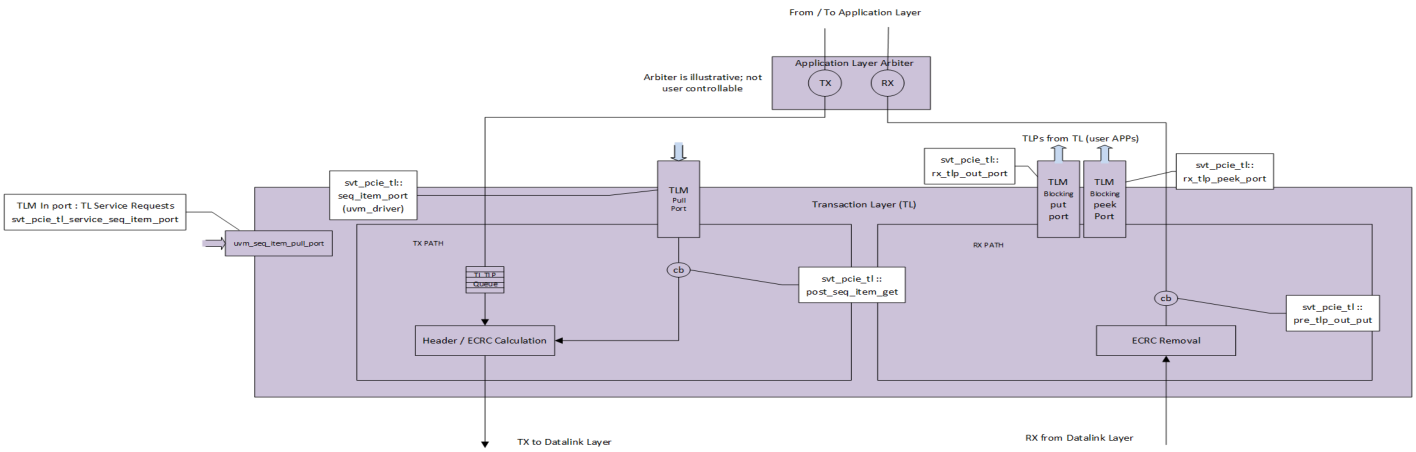
14.3 Transaction Layer Data Flow

The transaction layer is responsible for accepting TLP data from the application layer or user's test. It generates a full TLP from the data along with the header and ECRC calculation, if enabled.

14.3.1 Flow Diagram

Figure 14-2 highlights the key components of data flow through the VIP's Transaction Layer. Available TLMs and callbacks (cb) are shown in the diagram.

Figure 14-2 Flow Diagram – TL Layer



14.3.2 Transaction Layer Sequencer and Sequences

The transaction layer supports both service and transaction sequences. All TL sequences run on the svt_pcie_tl_service_sequencer. The TL sequencer is accessed through one of the following paths:

Virtual Sequencer:

This is the path through the virtual sequencer, virt_seqr. The virtual sequencer contains references to all of the nonvirtual sequencers. This is made available to enable the development of a high level virtual sequence that coordinates between all the nonvirtual sequencers. The path is

instance name of svt_pcie_agent.virt_seqr.tl_seqr

Sequencer:

This is the instantiated path to the nonvirtual sequencer:

instance name of svt_pcie_agent.tl_seqr

Note

The nonvirtual and virtual sequencers both are valid access points to the TL sequencer. Either may be used, though the virtual version is recommended when coordinating between multiple nonvirtual sequencers.

Services are commands to give the model that are related to behavior or configuration. They are not transaction items that are to be sent across the bus. For the TL there is a base service, svt_pcie_tl_service, and there is a base service sequence, svt_pcie_tl_service_base_sequence.

The service, svt_pcie_tl_service, is a sequence_item that supports all the transaction types supported by the TL. A service is selected by constraining the service_type_enum of the class to one of the enumerated service types.

Alternatively, the model provides several sequences that contain the base svt_pcie_tl_service that can be used for test development.

Example 14-1

```
svt_pcie_tl_check_final_credits_sequence fc_seq;
…
`uvm_do_on (fc_seq, p_sequencer.root_virt_seqr.pcie_virt_seqr.tl_seqr);
…
```

Transaction Layer service sequences are listed in Table 14-4.

Table 14-4 Transaction Layer service sequences

svt_pcie_dl_serivce_set_link_en_sequence Attribute: rand bit enable Range: 0-1 Description: Service enables the Data Link Layer. The Data Link Layer will automatically initiate VC0 InitFC process when enabled.
svt_pcie_tl_service_base_sequence Description: Base sequence for the TL; all other sequences extend from this sequence.
svt_pcie_tl_service_check_final_credits_sequence Description: 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. This task should be called at the end of every test. Any lost credits will be flagged.
svt_pcie_tl_service_clr_stats_sequence Description: Clears all stats in Transaction Layer.
svt_pcie_tl_service_disp_stats_sequence Description: Displays all stats in Transaction Layer.
svt_pcie_tl_service_null_sequence Description: Null sequence used to turn off the default sequence; typically used with UVM 1.0
svt_pcie_tl_service_set_vc_en_sequence Attribute: rand bit vc_enable Attribute: rand bit [2:0] vc_num Description: Enable and disable virtual channel. Call for each VC to enable. VC0 is enabled by default. vc_enable = 1 to enable, 0 to disable. vc_num is between 0 and 7.

In additon to the sequence library shown inTable 14-4, the Transaction Layer provides the svt_pcie_tl_service, which is a sequence item that can be used to build custom sequences. Enumerated values of the service type and their associated attributes are listed in Table 14-5.

Table 14-5 Service type enumerated parameters

Parameter	Attributes	I/O	Attribute Description
ADD_MEM_ADDR_APPL_ID_MAP_ENTRY Used to map memory addresses to an application.			
	memory_addr [63:0]	I	Base address of the memory range.
	memory_window [63:0]	I	Window of addresses that will cause a match of entry.
	appl_id [31:0]	I	Application ID to map TLP to.
	error [1]	O	Indication that addition of new entry failed.
ADD_IO_ADDR_APPL_ID_MAP_ENTRY Used to map I/O addresses to an application.			
	memory_addr [63:0]	I	Base address of the memory range .
	memory_window [63:0]	I	Window of addresses that will cause a match of entry.
	appl_id [31:0]	I	Application ID to map TLP to.
	error [1]	O	Indication that addition of new entry failed.
ADD_IO_ADDR_APPL_ID_MAP_ENTRY Used to map memory addresses that need address translation to an application.			
	memory_addr [63:0]	I	Base address of the memory range.
	memory_window [63:0]	I	Window of addresses that will cause a match of entry.
	appl_id [31:0]	I	Application ID to map TLP to.
	error [1]	O	Indication that addition of new entry failed.
ADD_AT_ADDR_APPL_ID_MAP_ENTRY Used to map memory addresses that need address translation to an application.			
	memory_addr [63:0]	I	Base address of the memory range.
	memory_window [63:0]	I	Window of addresses that will cause a match of entry.
	appl_id [31:0]	I	Application ID to map TLP to.
	error [1]	O	Indication that addition of new entry failed.
ADD_CFG_BDF_APPL_ID_MAP_ENTRY Used to map Config Request {Bus, Device, Function} to applications. Used when the VIP is the 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, the onfiguration is a Type 1 request. If false, the configuration is a Type 0 request..
	bdf [15:0]	I	{bus, device, function} of the request.
	appl_id [31:0]	I	Application ID to map TLP to.
	error [1]	O	Indication that addition of new entry failed.

Table 14-5 Service type enumerated parameters (Continued)

ADD_RID_APPL_ID_MAP_ENTRY 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 [15:0]	I	Requester ID to map.
	appl_id [31:0]	I	Application ID to map TLP to.
	error [1]	O	Indication that addition of new entry failed.
ADD_RID_MSG_CODE_APPL_ID_MAP_ENTRY Used to map {Requester Ids, msgcode} of MSG TLPs to applications.			
	requester_id [15:0]	I	Requester ID to map.
	msgcode [7:0]	I	Msgcode of TLP. Same value as msgcode field in TLP header. See Include/pciesvc_parms.vp file for defines.
	appl_id [31:0]	I	Application ID to map TLP to.
	error [1]	O	Indication that addition of new entry failed.
DISPLAY_MEM_ADDR_APPL_ID_MAP Displays all memory addressses.			
DISPLAY_IO_ADDR_APPL_ID_MAP Displays all I/O address to application ID map entries.			
DISPLAY_AT_ADDR_APPL_ID_MAP Displays all memory address to application ID map entries that require address translation.			
DISPLAY_CFG_BDF_APPL_ID_MAP Displays all configuration Type 0 and Type 1 {Bus, Device, Function} to application ID map entries. Use when the VIP is the upstream port of a link.			
DISPLAY_RID_APPL_ID_MAP Displays all Requester ID to application ID map entries. Use when the VIP is the upstream port of a link.			
DISPLAY_RID_MSG_CODE_APPL_ID_MAP Displays all {Requester ID, MsgCode} to application ID map entries.			
DISPLAY_STATS Displays all stats in the Transaction Layer.			
CLEAR_STATS Clears all stats in the Transaction Layer.			
CHECK_FINAL_CREDITS 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.			
SET_VC_ENABLE Enable or disable a virtual channel. Called for each VC to enable. VC0 is enabled by default.			

Table 14-5 Service type enumerated parameters (Continued)

	rand bit vc_enable	I	Enable or disable the VC.
	rand bit[31:0] vc_num	I	Virtual channel number.
IS_TL_IDLE Indicates whether the Transaction Layer is currently idle. **Note** IS_TL_IDLE will be deprecated in a future release. The preferred way to check whether the TL is idle is to use the status object at shown in "Determining if the Transaction Layer is Idle".			
	rand bit tl_idle	O	Returns the TL status: tl_idle == 0 means not idle, tl_idle == 1 means idle.

14.3.3 Transaction Layer Callbacks and Exceptions

The Transaction Layer provides a callback class, svt_pcie_tl_callback, for applying exceptions to TLP transactions. Refer to the Flow Diagram – TL Layer where the TX and RX callbacks are applied.For more information on the TL callbacks and exceptions, refer to Chapter 19.6.

14.3.4 Transaction Layer TLMs

The svt_pcie_tl provides TLMs for access to the received TLPs. There is a uvm_blocking_put_port and a uvm_blocking_peek_imp, which are rx_tlp_out_put and rx_ltp_peek_port, respectively. You can connect to those ports for access to all received TLPs.

Additonally, there is a uvm_seq_item_pull_port, service_seq_item_port that is used for service requests and user applications.

14.4 Transaction Layer Status

The transaction layer provides a status class that provides statistics regarding the TL layer. The class is accessed using the following instance hierarchy:

instance name of svt_pcie_agent.status.tl_status

Refer to the HTML Reference for a full listing of status members.

14.4.0.1 Determining if the Transaction Layer is Idle

The TL provides the svt_pcie_tl_status::is_idle member for determining if the Transaction Layer is idle (that is, all queues are empty). This is useful for determining end-of-test.

Example 14-2

```
svt_pcie_device_status root_status = p_sequencer.get_root_shared_status(this);
wait(root_status.pcie_status.tl_status.is_idle);
```

Additionally, you can use the svt_pcie_tl_service to run a service to check on the status. See IS_TL_IDLE in Table 14-5.

Note
In a future release the IS_TL_IDLE parameter will be deprecated. The preferred way to check whether the Transaction Layer is idle is to use the status object as shown in Example 14-2.

14.5 Transaction Layer Verilog Interface

The Verilog component of the TL is instantiated within the MAC. It can be found at:

path to instantiation model.port0.tl0

The signals at this level are useful for debugging. A few of the signals are highlighted in the following sections.

14.5.0.1 Transaction Layer Module IOs

The Transaction Layer module I/O signals listed in Table 14-6 are the Verilog module port connections to the TL layer. These are useful for browsing the VIP reset and checking if a particular VC is initialized.

Table 14-6 Transaction Layer Module IOs

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 [31:0]	Bits (use predefined parameters 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

15 Data Link Layer Features and Classes

15.1 Classes and Applications for Using the VIP's Data Link Layer

The following classes have members and tasks to implement Data Link Layer features and operation.

- **"Component Class svt_pcie_dl".** Describes power management at DL Layer.
- **Component Class svt_pcie_dl**. A uvm_component which implements the PCIe Data Link Layer. This class is included in the svt_pcie_agent. When you instantiate the PCIe UVM agent, you will also instantiate this component.
- **Configuration class svt_pcie_dl_configuration**. This class contains class members to configure the behavior of the Data Link Layer. For example, the class has the member svt_pcie_dl_configuration::replay_timeout which you would use to configure the length of the replay timer in symbols.
- **Status class svt_pcie_dl_status** Used for returning status and statistics back to your testbench.
- **Service Class svt_pcie_dl_service.** Service transactions for Link layer module. For example, if you want to initiate a transition to the L0 state from the PM low power state, then you would use the member INITIATE_PM_EXIT(10).

The following section lists and describes the members and tasks of the major classes to implement various features for your testbench.

Attention Note, the descriptions for the classes and applications show the most important and often used members/features. Consult the PCIe UVM HTML Class Reference for a complete listing of the members and their data types.

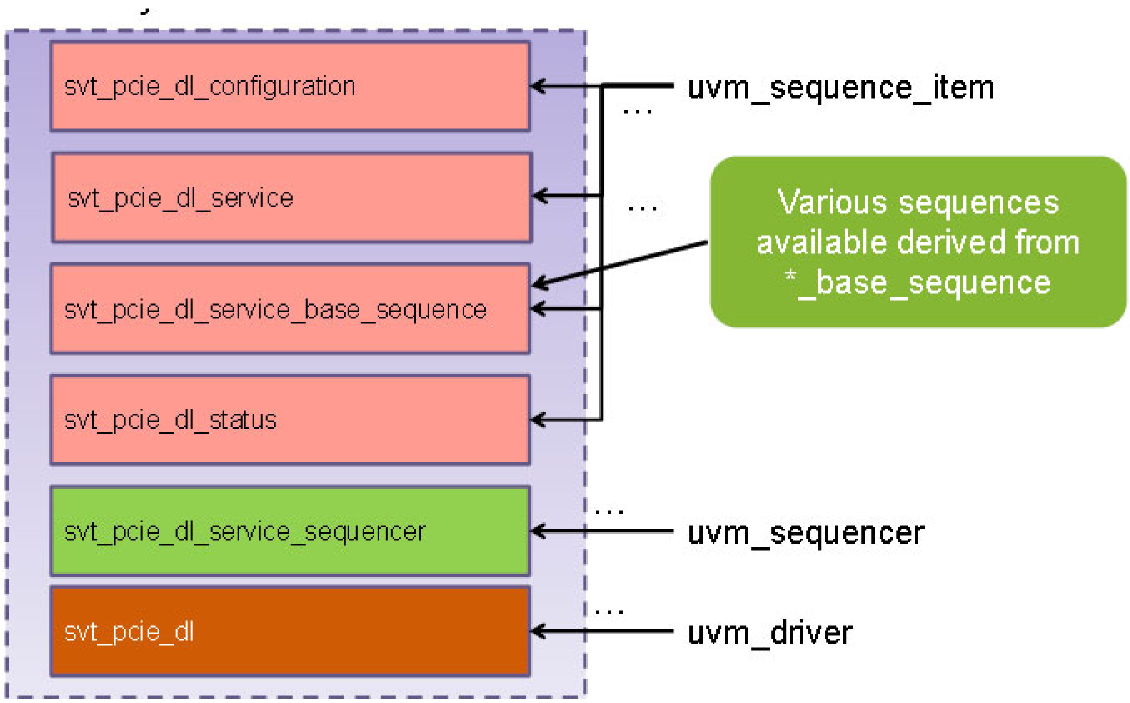
15.2 Additional Documentation on DL Programming Tasks

Consult the following to get information on DL related programming tasks.

- "SolvNet PCIe VIP Articles" on page 533.
- PCIe SVT FAQ

15.3 Class Elements of the Link Layer

The following illustration shows the classes making up the Link Layer. They will be discussed in various sections of the chapter.



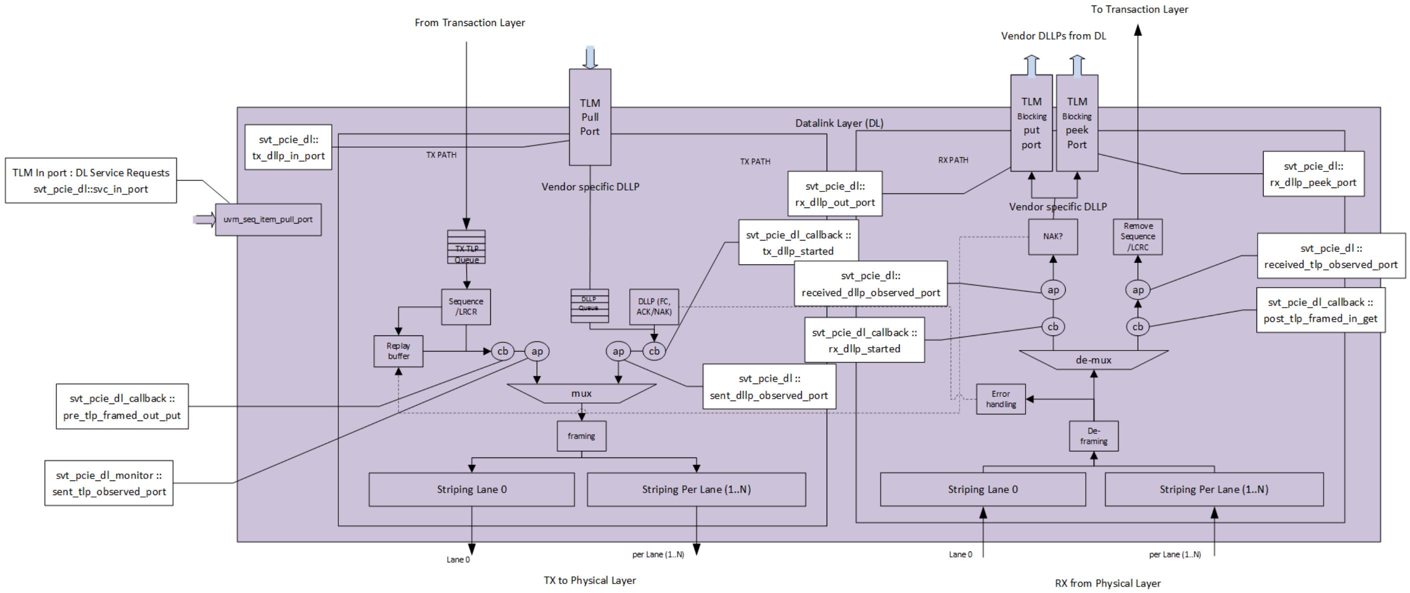
15.4 Datalink Layer Data Flow

The Datalink layer is responsible for TLP and DLP traffic. Regarding the TLPs, the layer accepts formed TLPs from the Transaction Layer, appends sequence and LCRC information and then frames and stripes into lane data before passing to the Physical Layer. For DLPs, a generator creates flow control and ack/nak DLPs which are then framed and striped per lane. Additionally, user-defined DLPs may be mixed in. On the RX side, lanes are de-striped and unframed for both DLPs and TLPs. Protocol checks are validated and eventually the DLP is passed out a TLM and TLPs removed of sequence and LRCR before being passed to the Transaction layer.

15.4.1 Flow Diagram

Figure 15-1 highlights the key components of data flow through the VIP's Datalink Layer. Available TLMs, analysis ports (ap), and callbacks (cb) are shown in the diagram.

Figure 15-1 Flow Diagram – Datalink Layer



15.4.2 Service Class svt_pcie_dl_service

The Data Link Layer service class is responsible for the implementing the following major tasks and features.

- Initiating transitions for the VIP to enter low power states:
  - ASPM Tx L0s
  - ASPM L1 low power state.
  - PM L1
  - PM L2/L3
  - Back to L0 from ASPM
  - Back to L0 from PM
- Setting ACKFactor value
- Displaying DLL statistics
- Allowing DLCMSM to transition out of Disabled state.
- Enabling and disabling gating
- Gating FC type of INITFC frames
- Setting the Virtual channel of the INITFC frames to be gated
- Getting Status information about the current processing state

15.4.2.1 Members and Features

The following table lists major members of the svt_pcie_dl_service and the features they support.

Table 15-1 Frequently Used Members and Features of Service Class svt_pcie_dl_service

Member	Feature/Use
ack_factor	Updates the AckFactor table entries listed in specification.
enable	Set to allow DLCMSM to transition out of Disabled state.
fc_type	FC type of INITFC frames to be gated.
gate_enable	Gating enable/disable
status	Status information about the current processing state
rate	Store the link speed used during the updating of the AckFactor table entries listed in specifications.
The following define the elements of the service_type_enum	
SET_LINK_ENABLE(0)	Enables Data Link layer. When enabled, data link layer automatically initiates the VC0 InitFC process
DISPLAY_ATTACHED_REPLAY_TIMER_TOLERANCES(1)	Displays replay timer checker tolerances for link widths of 1, 2,4,8,12,16, and 32.
DISPLAY_ATTACHED_ACK_NAK_LATENCY_TOLERANCES(2)	Displays AckNak Latency checker tolerances for link widths of 1, 2,4,8,12,16, and 32.
DISPLAY_STATS(3)	Displays link layer statistics.
CLEAR_STATS(4)	Resets statistics counters.
INITIATE_ASPM_L0S_ENTRY(5)	Initiates VIP to enter ASPM Tx L0s low power state.
INITIATE_ASPM_L1_ENTRY(6)	Initiates VIP to enter ASPM L1 low power state.
INITIATE_PM_L1_ENTRY(7)	Initiates VIP to enter PM L1 low power state.
INITIATE_PM_L23_ENTRY(8)	Initiates VIP to enter PM L2/L3 low power states.
INITIATE_ASPM_EXIT(9)	Initiates VIP to transition back to L0 from ASPM low power state.
INITIATE_PM_EXIT(10)	Initiates VIP to transition back to L0 from PM low power state.
GATE_TX_INITFC1(11)	Gates the Transmitted INITFC1 frames
GATE_TX_INITFC2(12)	Gates the transmitted INITFC2 frames
GATE_TX_ACKNAK(13)	Gates the Transmitted ACK and NAK frames
SET_ACK_FACTOR(14)	Sets ACK Factor value

15.4.3 Datalink Layer TLMs

The svt_pcie_dl provides TLMs for access to the received DLPs. There is a uvm_blocking_put_port and a uvm_blocking_peek_imp, which are rx_dllp_out_put and rx_dllp_peek_port, respectively. You can connect to those ports for access to all received DLLPs. Additionally, there is a uvm_seq_item_pull_port, tx_dllp_in_port for user-defined DLLPs.

15.4.4 Datalink Layer Callbacks and Exceptions

The Datalink Layer provides a callback class, svt_pcie_dl_callback, for observation and application of exceptions to both incoming and outbound transactions. Refer to the Flow Diagram – Datalink Layer where the TX and RX callbacks are applied. For more information on the DL callbacks and exceptions, refer to Chapter 19.6.

15.4.5 Datalink Layer Analysis Ports

The Datalink Layer provides analysis ports for the all outbound and inbound TLPs and DLLPs. For more details see pcie_svt_dl_monitor in the HTML class reference and Flow Diagram – Datalink Layer.

15.5 Component Class svt_pcie_dl

The UVM component class svt_pcie_dl is responsible for the following features:

- Implements Link layer module
- Implements static and dynamic configuration. Dynamic configuration is the ability of the model to configure and re-configure at run time. Static configuration is done before at time zero (simulation time). The following table shows the members supporting dynamic and static configuration and general status monitoring.
- Responsible for reconfigure PCIE
- Provides status of the application.
- Provides a SIPP [Sequence Item Pull Port] to cater to services of type svt_pcie_dl_service.
- Provides classes and members for error injection

Following table lists significant functions and data members.

Table 15-2 Members and Features for svt_pcie_dl

Member	Feature
Functions	
post_tlp_framed_in_get(...)	Called by the component after recognizing a TLP Transaction received on the link.
pre_dllp_out_putt(...)	Callback issued by the component just prior to putting a Vendor Specific DLLP transaction received on the link on the put port.
pre_dllp_transmission_svc_callback t(...)	Method used to apply TX DLLP exceptions.
pre_phy_pkt_w_framing_transmission_svc_callback t(...)	Method used to apply PHY packet framing exceptions.
pre_tlp_framed_out_put t(...)	Callback issued by the component after scheduling a TLP transaction for transmission on the link, just prior to framing.
pre_tlp_transmission_svc_callback t(...)	Method used to apply TX TLP exceptions.

Table 15-2 Members and Features for svt_pcie_dl (Continued)

Member	Feature
rx_dllp_post_deframed_callback t(...)	Called from the SVC Link Layer to report an inbound DLLP for callback.
rx_dllp_startedt(...)	Callback issued by the component immediately after receiving a User DLLP transaction on the set_item_port prior to its further processing.
rx_tlp_post_deframed_callback t(...)	Called from the SVC Link Layer to report an inbound TLP for callback.
tx_dllp_pre_framed_callback t(...)	Called from the SVC Link Layer to report an outbound DLLP for callback.
tx_dllp_startedt(...)	Called by the component after building a DLLP Transaction just prior to its further processing.
tx_tlp_pre_framed_callbackt(...)	Called from the SVC Link Layer to report an outbound TLP for callback.
get_cfg t(...)	Overrides the base method to generate an error.
reconfiguret(...)	Overrides the base method to generate an error.
set_err_checkt(...)	Used to set the err_check object and to fill in all of the local checks.
Following are derived from various base classes for port tracking and error injection	
svt_pcie_dl_tlp_exception dl_tlp_xact_rx_exception	Randomization factory to create RX TLP exception (error etc.) to be inserted in transaction
svt_pcie_dl_tlp_exception_list dl_tlp_xact_rx_exception_list	Randomization factory to create RX TLP exception list for a TLP transaction
svt_pcie_dl_tlp_exception dl_tlp_xact_tx_exception = null;	Randomization factory to create TX TLP exception list for a TLP transaction
svt_pcie_dl_tlp_exception_list dl_tlp_xact_tx_exception_list	Randomization factory to create TX TLP exception list for a TLP transaction
svt_pcie_dllp_exception dllp_xact_exception	Randomization factory to create TX DLLP exception list for a TLP transaction
svt_pcie_dllp_exception_list dllp_xact_exception_list	Randomization factory to create TX DLLP exception list for a TLP transaction
svt_pcie_phy_transaction_exception phy_xact_exception	Randomization factory to create TX PHY transaction framing exception (error etc.) to be inserted in packet
phy_xact_exception_list	Randomization factory to create TX PHY transaction framing exception list for a packet

Table 15-2 Members and Features for svt_pcie_dl (Continued)

Member	Feature
svt_debug_opts_analysis_port received_dllp_observed_port	Analysis port for received DLLPs. This port is generally used for scoreboarding. The DLLPs observed via this analysis port are controlled by svt_pcie_dl_configuration :: received_dllp_interface_mode
svt_debug_opts_analysis_port received_tlp_observed_port	Analysis port for received TLPs. This port is generally used for scoreboarding. The TLPs observed via this analysis port are controlled by svt_pcie_dl_configuration :: received_tlp_interface_mode.
svt_debug_opts_blocking_put_port rx_dllp_out_port	RX DLLP Put Port Provides a mechanism for external components to receive vendor specific DLLPs from the DAta Link Layer. The handle to this DLLP put port can be set or obtained through the driver's public member rx_dllp_out_port
svt_debug_opts_blocking_peek_imp_port rx_dllp_peek_port RX DLLP Peek port.	Provides a mechanism for external components to retrieve Vendor Specific DLLPs from the Data Link Layer. The handle to this DLLP peek port can be set or obtained through the driver's public member rx_dllp_peek_port .
svt_debug_opts_analysis_port sent_dllp_observed_port	Analysis port for sent DLLPs. This port is generally used for scoreboarding. The DLLPs observed via this analysis port are controlled by svt_pcie_dl_configuration :: sent_dllp_interface_mode.
svt_debug_opts_analysis_port sent_tlp_observed_port	Analysis port for sent TLPs. This port is generally used for scoreboarding. The TLPs observed via this analysis port are controlled by svt_pcie_dl_configuration :: sent_tlp_interface_mode.

15.6 Configuration class svt_pcie_dl_configuration

Use the svt_pcie_dl_configuration to define the overall behavior of the Data Link Layer. You can define the following behaviors for over 88 different features. In addition, note that most configurable attributes are defined as SystemVerilog RAND types. This allows your testbench to randomize them following predefined constraints provided by Synopsys. Consult the PCIe HTML Class Reference on declared data types.

15.6.1 Members and Features

The following table lists configuration members under these major functional areas:

- **Flow/Credit Management**
- **Replay Ack/Nak**
- **DLL Packet Control**
- **Power Management**

15.6.1.1 Flow/Credit Management

Following are some tasks using Flow/Credit Management configuration:

- Set initial credits
- Control response to credit requests

Table 15-3 Flow and Credit Configuration Features

Member	Feature
initfc_timeout_ns	Maximum time between receiving all 3 types (P, NP, CPL) of InitFC DLLPs. SVC will issue NOTICE if timeout value is exceeded.
max_initfc_delay	Maximum interval between InitFC DLLPs. Used to model delays to due media access. Value should be relatively small.
max_updatefc_delay	Maximum interval between UpdateFC DLLPs. Used to model delays to due media access. Value should be relatively small.
min_initfc_delay	Minimum interval between InitFC DLLPs. Used to model delays to due media access. Value should 0 be relatively small.
min_num_tx_initfc1_cpl	Minimum number of InitFC1-CPL DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC1-CPL DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays.
min_num_tx_initfc1_np	Minimum number of InitFC1-NP DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC1-NP DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays.

Table 15-3 Flow and Credit Configuration Features (Continued)

Member	Feature
min_num_tx_initfc1_	Minimum number of InitFC1-P DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC1-P DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays.
min_num_tx_initfc2_cpl	Minimum number of InitFC2-CPL DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC2-CPL DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays. Setting all to 0 will allow the VIP to send only 1 iniFC2-P/NP/CPL in FC_INIT2 state.
min_num_tx_initfc2_np	Minimum number of InitFC2-NP DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC2-NP DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays. Setting all to 0 will allow the VIP to send only 1 iniFC2-P/NP/CPL in FC_INIT2 state.
min_num_tx_initfc2_p	Minimum number of InitFC2-P DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC2-P DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays. Setting all to 0 will allow the VIP to send only 1 iniFC2-P/NP/CPL in FC_INIT2 state.
updatefc_timeout_ns	Maximum time between receiving UpdateFC DLLPs. Timeout is tracked per VC/Type. SVC will issue NOTICE if timeout value is exceeded.
min_updatefc_delay	Minimum interval between UpdateFC DLLPs. Used to model delays to due media access. Value should be relatively small.
vc0_updatefc_interval_ns	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.
vc1_updatefc_interval_ns	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.

Table 15-3 Flow and Credit Configuration Features (Continued)

Member	Feature
vc2_updatefc_interval_ns	
vc3_updatefc_interval_ns	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.
vc4_updatefc_interval_ns	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.
vc5_updatefc_interval_ns	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.
vc6_updatefc_interval_ns	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.
vc7_updatefc_interval_ns	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.
tx_fc_init_completed_by_tlp	Enables sending a TLP or an UpdateFC DLLP to complete flow control initialization instead of sending and INITFC2 DLLP. A TLP must be queue in order for initialization to complete. The probablility of one of these is controlled by the percentage. The weights control the relative likelihood of one over the other.
tx_fc_init_completed_by_updatefc	Enables sending a TLP or an UpdateFC DLLP to complete flow control initialization instead of sending and INITFC2 DLLP. A TLP must be queue in order for initialization to complete. The probablility of one of these is controlled by the percentage. The weights control the relative likelihood of one over the other. ;
tx_fc_init_completed_percentage	Enables sending a TLP or an UpdateFC DLLP to complete flow control initialization instead of sending and INITFC2 DLLP. A TLP must be queue in order for initialization to complete. The probablility of one of these is controlled by the percentage. The weights control the relative likelihood of one over the other.

Table 15-3 Flow and Credit Configuration Features (Continued)

Member	Feature
initfc_timeout_ns	Maximum time between receiving all 3 types (P, NP, CPL) of InitFC DLLPs. SVC will issue NOTICE if timeout value is exceeded.
max_initfc_delay	Maximum interval between InitFC DLLPs. Used to model delays to due media access. Value should be relatively small.
max_updatefc_delay	Maximum interval between UpdateFC DLLPs. Used to model delays to due media access. Value should be relatively small.
min_initfc_delay	Minimum interval between InitFC DLLPs. Used to model delays to due media access. Value should 0 be relatively small.
min_num_tx_initfc1_cpl	Minimum number of InitFC1-CPL DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC1-CPL DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays.
min_num_tx_initfc1_np	Minimum number of InitFC1-NP DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC1-NP DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays.
min_num_tx_initfc1_p	Minimum number of InitFC1-P DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC1-P DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays.
min_num_tx_initfc2_cpl	Minimum number of InitFC2-CPL DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC2-CPL DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays. Setting all to 0 will allow the VIP to send only 1 iniFC2-P/NP/CPL in FC_INIT2 state.
min_num_tx_initfc2_np	Minimum number of InitFC2-NP DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC2-NP DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays. Setting all to 0 will allow the VIP to send only 1 iniFC2-P/NP/CPL in FC_INIT2 state.

Table 15-3 Flow and Credit Configuration Features (Continued)

Member	Feature
min_num_tx_initfc2_p	Minimum number of InitFC2-P DLLPs that must be transmitted in FC_INIT1 state before transitioning to FC_INIT2 state. The VIP might send more InitFC2-P DLLPs in FC_INIT1 state depending on reception of InitFC-P/NP/CPL DLLPs per spec. Commonly set to a value greater than the default to mimic real life processing delays. Setting all to 0 will allow the VIP to send only 1 iniFC2-P/NP/CPL in FC_INIT2 state.
updatefc_timeout_ns	Maximum time between receiving UpdateFC DLLPs. Timeout is tracked per VC/Type. SVC will issue NOTICE if timeout value is exceeded.
min_updatefc_delay	Minimum interval between UpdateFC DLLPs. Used to model delays to due media access. Value should be relatively small.
vc0_updatefc_interval_ns	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.
vc1_updatefc_interval_ns	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.
vc2_updatefc_interval_ns	
vc3_updatefc_interval_ns	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.
vc4_updatefc_interval_ns	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.
vc5_updatefc_interval_ns	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.

Table 15-3 Flow and Credit Configuration Features (Continued)

Member	Feature
vc6_updatefc_interval_ns	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.
vc7_updatefc_interval_ns	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.
tx_fc_init_completed_by_tlp	Enables sending a TLP or an UpdateFC DLLP to complete flow control initialization instead of sending and INITFC2 DLLP. A TLP must be queue in order for initialization to complete. The probablility of one of these is controlled by the percentage. The weights control the relative likelihood of one over the other.
tx_fc_init_completed_by_updatefc	Enables sending a TLP or an UpdateFC DLLP to complete flow control initialization instead of sending and INITFC2 DLLP. A TLP must be queue in order for initialization to complete. The probablility of one of these is controlled by the percentage. The weights control the relative likelihood of one over the other. ;
tx_fc_init_completed_percentage	Enables sending a TLP or an UpdateFC DLLP to complete flow control initialization instead of sending and INITFC2 DLLP. A TLP must be queue in order for initialization to complete. The probablility of one of these is controlled by the percentage. The weights control the relative likelihood of one over the other.
rx_credit_latency_in_symbols	The variable represents maximum latency until a received credit is passed to the TL for processing. The latency is in symbol times.

15.6.1.2 Replay Ack/Nak

Table 15-4 Replay Ack/Nak Configuration Members and Features

Member	Feature
max_payload_size	
attached_ack_nak_latency_tolerance_x1	Variance in number of symbols in which ACK may be received. This should account for clock domain crossings, TLP processing variances, etc. Each link width can have a different value.
attached_ack_nak_latency_tolerance_x12	Variance in number of symbols in which ACK may be received. This should account for clock domain crossings, TLP processing variances, etc. Each link width can have a different value.

Table 15-4 Replay Ack/Nak Configuration Members and Features (Continued)

Member	Feature
attached_ack_nak_latency_tolerance_x16	Variance in number of symbols in which ACK may be received. This should account for clock domain crossings, TLP processing variances, etc. Each link width can have a different value.
attached_ack_nak_latency_tolerance_x2	Variance in number of symbols in which ACK may be received. This should account for clock domain crossings, TLP processing variances, etc. Each link width can have a different value.
attached_ack_nak_latency_tolerance_x32	Variance in number of symbols in which ACK may be received. This should account for clock domain crossings, TLP processing variances, etc. Each link width can have a different value.
attached_ack_nak_latency_tolerance_x4	Variance in number of symbols in which ACK may be received. This should account for clock domain crossings, TLP processing variances, etc. Each link width can have a different value.
attached_ack_nak_latency_tolerance_x8	Variance in number of symbols in which ACK may be received. This should account for clock domain crossings, TLP processing variances, etc. Each link width can have a different value.
attached_internal_delay_2_5g	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.
attached_internal_delay_5g	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.
attached_internal_delay_8g	Internal delay used in “Ack Transmit Latency Limit” equation to check attached replay timer and AckNak latency timer at 8G. Accounts for packet processing delay and transmission latency.
attached_replay_timeout	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.
attached_replay_timeout_tolerance_x1	Number of symbols in which retry may be received after a timeout. This should account for clock domain crossings, ACK processing variances, etc. Each link width can have a different value.
attached_replay_timeout_tolerance_x12	Number of symbols in which retry may be received after a timeout. This should account for clock domain crossings, ACK processing variances, etc. Each link width can have a different value.
attached_replay_timeout_tolerance_x16	Number of symbols in which retry may be received after a timeout. This should account for clock domain crossings, ACK processing variances, etc. Each link width can have a different value.

Table 15-4 Replay Ack/Nak Configuration Members and Features (Continued)

Member	Feature
attached_replay_timeout_tolerance_x2	Number of symbols in which retry may be received after a timeout. This should account for clock domain crossings, ACK processing variances, etc. Each link width can have a different value.
attached_replay_timeout_tolerance_x32	Number of symbols in which retry may be received after a timeout. This should account for clock domain crossings, ACK processing variances, etc. Each link width can have a different value.
attached_replay_timeout_tolerance_x4	Number of symbols in which retry may be received after a timeout. This should account for clock domain crossings, ACK processing variances, etc. Each link width can have a different value.
attached_replay_timeout_tolerance_x8	Number of symbols in which retry may be received after a timeout. This should account for clock domain crossings, ACK processing variances, etc. Each link width can have a different value.
max_nak_latency	The maximum latency before a NAK is issued
max_ack_nak_latency	Max length of the ACK/NAK latency timer in symbols. If called, value is sticky. Setting to value = 0 will enable automatic updates.
max_attached_ack_nak_latency	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.
max_attached_nak_latency	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.
max_num_replays ;	Maximum number of replays before the link gives up on transmitting a TLP
internal_delay_2_5g	Internal delay used in "Ack Transmit Latency Limit" equation for replay timer and AckNak latency timer at 2.5G.
internal_delay_5g	Internal delay used in "Ack Transmit Latency Limit" equation for replay timer and AckNak latency timer at 5G.
internal_delay_8g	Internal delay used in "Ack Transmit Latency Limit" equation for replay timer and AckNak latency timer at 8G.
min_ack_nak_latency	Min Length of the ACK/NAK latency timer in symbols.
min_attached_ack_nak_latency	Min length of the attached ACK/NAK latency timer in symbols. Default value = 0.
min_nak_latency	Minimum latency until a NAK is issued

Table 15-4 Replay Ack/Nak Configuration Members and Features (Continued)

Member	Feature
min_pm_ack_latency	The variable represents minimum latency until PM_REQUEST_ACK is issued.
replay_timeout	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.

15.6.1.2.1 Calculating Ack/Nak Latency Values

Ack/Nak latency values are calculated as shown in the following example:

- attached_acknak_latency_timer_limit

```
attached_acknak_latency_timer_limit
                                   = attached_max_nak_latency +
                                     attached_internal_delay +
                                     internal_delay +
                                     internal_phy_delay +
                                     attached_internal_phy_delay
```

- acknak_latency_timer_limit

  `acknak_latency_timer_limit` is a random value between `min_acknak_latency` and `max_acknak_latency`.

- acknak_latency

```
acknak_latency
           = (((MAX_PAYLOAD_SIZE_VAR + TLP_OVERHEAD) * ack_factor) / smallest_width) +
             internal_delay
```

- attached_acknak_latency

```
attached_acknak_latency
        = (((MAX_PAYLOAD_SIZE_VAR + TLP_OVERHEAD) * ack_factor) / smallest_width) +
          attached_internal_delay
```

15.6.1.3 DLL Packet Control

Table 15-5 DLL Packet Control

Member	Feature
enable_ei_tx_tlp_on_retry	The variable enalbes error injection on retry frames.
enable_tx_tlp_reporting	If enabled, transmitted TLPs will be reported to the global shadow memory.
initial_receive_sequence_value	The first TLP sent must have this sequence number
initial_transmit_sequence_value	The first TLP will be transmitted with this value. Used to check rollover at 4096
ltr_l1_2_threshold_scale	The variable represents LTR Threshold scale for L1.2 substate.
ltr_l1_2_threshold_value	The variable represents LTR Threshold value for L1.2 substate.
max_size_packet	Maximum possible header size plus maximum possible payload size.
max_header_size	Maximum possible header size plus maximum possible prefixes.
max_num_prefixes	Maximum possible prefixes.
max_tx_ipg	Max gap between packets (TLPs and/or DLLPs) as measured in symbol times.
max_tx_nullified_tlp_len	Maximum nullified TLP length in dwords.
min_tx_ipg	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.
min_tx_nullified_tlp_len	Minimum nullified TLP length in dwords.
model_instance_scope	This is the full hierarchical path name to the instance of the SVC model which the driver model is intantiated in. The path name is concatenated with the name of this component passed to the constructor to generate the lookup string used to find the SV API instance.
received_dllp_interface_mode	Select DLLPs available at Receive DLLP analysis port. DLLPs are filtered based on th bits enabled in received_dllp_interface_mode bit vector.
received_tlp_interface_mode	Select TLPs available at Receive TLP analysis port. TLPs are filtered based on the bits enabled in received_tlp_interface_mode bit vector
sent_dllp_interface_mode	Select DLLPs available at Sent DLLP analysis port. DLLPs are filtered based on the bits enabled in sent_dllp_interface_mode bit vector

Table 15-5 DLL Packet Control

Member	Feature
sent_tlp_interface_mode	Select TLPs available at Sent TLP analysis port. TLPs are filtered based on the bits enabled in sent_tlp_interface_mode bit vector
tx_nullified_tlp_hdr0_value	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	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_hdr2_value	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_hdr3_value	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.

15.6.1.4 Power Management

Table 15-6 Power Management and Features

Members	Features
aspm_timeout_cnt_limit	If DUT fails to respond to ASPM request handshake for this # symbols, NOTICE will be issued and ASPM entry will be aborted.
aspm_timeout_cnt_limit	If DUT fails to respond to ASPM request handshake for this # symbols, NOTICE will be issued and ASPM entry will be aborted.
enable_aspm_l0s_entry	Enable ASPM L0S entry.
enable_aspm_l1_1_entry	The variable enables ASPM L1.1 entry.
enable_aspm_l1_2_entry	The variable enables ASPM L1.2 entry.
enable_aspm_l1_entry	Enable ASPM L1S entry.
enable_pm_l1_1_entry	The variable enables PM L1.1 entry.
enable_pm_l1_2_entry	The variable enables PM L1.2 entry.
max_tx_pm_ipg	Max number of symbol times between PM DLLPs.
min_tx_pm_ipg	Min number of symbol times between PM DLLPs.
max_pm_ack_latency	The variable represents maximum latency until PM_REQUEST_ACK is issued.

15.7 Status class svt_pcie_dl_status

This class makes available the specific status information as it relates to the Data Link Layer. For example you can get the number of:

- NACK DLLPs received and sent
- TLPs and DDLPs received and sent with errors
- Tx alignment errors injected with two STPs per symbol.
- Tx DLLP code violation errors injected.
- Tx DLLPs with non-zero reserved bits injected.
- Number of packets that had to be retransmitted.
- Number of credits on every virtual channel.

Consult the following table for all the status and stasticis you can gather from the class svt_pcie_dl_status.

Table 15-7 Status Members and Features

Status Member	Feature
dl_link_up	Indicates DL Up status.
is_idle	Indicates whether the data link layer is idle
num_ack_received num_ack_sent	Number of ACK DLLPs received and sent.
num_vendor_dllp_received num_vendor_dllp_sent	Number of Vendor DLLPs sent and received.
num_bad_retries_sent	Number of retry TLPs sent with error.
num_bad_tlp_received num_bad_tlp_sent	Number of TLPs received with error.
num_dllp_received num_dllp_sent	Number of DLLPs received and sent.
num_duplicate_tlp_received	Number of duplicate TLPs received.
num_ei_rx_replay_count_failure	Number of replay count failure errors injected.
num_ei_rx_tlp_nak_good_tlp num_ei_rx_tlp_withhold_ack_nak	Number of NAK good TLP errors injected. Number of ACK withheld errors injected.
num_ei_tx_alignment_2_sdp_per_symbol num_ei_tx_alignment_2_stp_per_symbol num_ei_tx_alignment_sdp_packet_gap num_ei_tx_alignment_sdp_wrong_lane num_ei_tx_alignment_stp_packet_gap num_ei_tx_alignment_stp_wrong_lane	Not supported.

Table 15-7 Status Members and Features (Continued)

Status Member	Feature
num_ei_tx_dllp_8g_corrupt_header num_ei_tx_dllp_bit_flip num_ei_tx_dllp_code_violation num_ei_tx_dllp_corrupt_crc num_ei_tx_dllp_corrupt_disparity num_ei_tx_dllp_duplicate_ack num_ei_tx_dllp_missing_end num_ei_tx_dllp_missing_start num_ei_tx_dllp_rsvd_non_zero num_ei_tx_dllp_scrambler_error num_ei_tx_dllp_unknown_type	Number of Tx DLLP 8G corrupt header errors injected. Number of Tx DLLP bit flip errors injected. Number of Tx DLLP code violation errors injected. Number of Tx DLLPs corrupt CRC errors injected. Number of Tx DLLPs with corrupt disparity errors injected. Number of Tx DLLP duplicate ACK errors injected. Number of Tx DLLP missing end errors injected. Number of Tx DLLP missing start errors injected. Number of Tx DLLPs with non-zero reserved bits injected. Number of Tx DLLP scrambler errors injected. Number of Tx DLLPs with unknown type codes injected.
num_ei_tx_tlp_8g_corrupt_header_crc num_ei_tx_tlp_bit_flip num_ei_tx_tlp_code_violation num_ei_tx_tlp_corrupt_disparity num_ei_tx_tlp_corrupt_lcrc num_ei_tx_tlp_duplicate_seq_num num_ei_tx_tlp_illegal_seq_num num_ei_tx_tlp_missing_end num_ei_tx_tlp_missing_start num_ei_tx_tlp_nullified num_ei_tx_tlp_nullified_corrupt_lcrc num_ei_tx_tlp_nullified_good_lcrc num_ei_tx_tlp_scrambler_error	Number of Tx TLP 8G corrupted header CRC errors injected. Number of Tx TLPs with bit flip errors injected. Number of Tx TLPs with code violations injected. Number of Tx TLPs with disparity errors injected. Number of Tx TLP corrupt LCRC errors injected. Number of Tx TLP duplicate sequence number errors injected. Number of Tx TLP illegal sequence number errors injected. Number of Tx TLP with missing end errors injected. Number of Tx TLP with missing start errors injected. Number of nullified Tx TLPs injected. Number of nullified Tx TLPs with corrupted LCRC injected. Number of nullified Tx TLPs with good LCRC injected. Number of Tx TLP scrambler errors injected.
num_good_dllp_received num_good_dllp_sent	Number of DLLPs received and sent without error.
num_good_retries_sent	Number of retry TLPs sent without error.
num_good_tlp_received num_good_tlp_sent	Number of TLPs received and sent without error.
num_nak_received num_nak_sent	Number of ACK DLLPs received and sent.
num_nullified_tlp_received	Number of nullified TLPs received.
um_out_of_order_tlp_received	Number of out-of-order TLPs received.
num_phy_link_down_events	Number of Phy Link Down events
num_phy_retrain	Number of times the link was retrained to due the replay attempt counter rolling over.

Table 15-7 Status Members and Features (Continued)

Status Member	Feature
num_retries_due_to_nak num_retries_due_to_replay_timeout num_retries_sent	Number of retried packets due to a NAK. Number of retried packets due to replay timeout. Number of packets that had to be retransmitted.
num_rx_dllp_missing_end num_rx_tlp_missing_end	Number of DLLPs received with missing ENDs. Number of TLPs received with missing ENDs.
num_tlp_received num_tlp_sent	Number of TLPs received and sent.
num_vc<0-7>_cpl_initfc1_received num_vc<0-7>_cpl_initfc1_sent num_vc<0-7>_cpl_initfc2_received num_vc<0-7>_cpl_initfc2_sent num_vc<0-7>_cpl_updatefc_received num_vc<0-7>_cpl_updatefc_sent num_vc<0-7>_initfc1_received num_vc<0-7>_initfc1_sent num_vc<0-7>_initfc2_received num_vc<0-7>_initfc2_sent num_vc<0-7>_np_initfc1_received num_vc<0-7>_np_initfc1_sent num_vc<0-7>_np_initfc2_received num_vc<0-7>_np_initfc2_sent num_vc<0-7>_np_updatefc_received num_vc<0-7>_np_updatefc_sent num_vc<0-7>_p_initfc1_received num_vc<0-7>_p_initfc1_sent num_vc<0-7>_p_initfc2_received num_vc<0-7>_p_initfc2_sent num_vc<0-7>_p_updatefc_received num_vc<0-7>_p_updatefc_sent num_vc<0-7>_updatefc_received num_vc<0-7>_updatefc_sent	The number of sent and received credits on Virtual Channels zero-to-seven: • CPL UpdateFC DLLPs • InitFC1/2 DLLPs • NP UpdateFC DLLPs • P, NP, or CPL InitFC1 DLLPs

16 PHY Layer Features and Classes

16.1 Classes and Applications for Using the VIP's PHY Layer

The following classes have members and tasks to implement Data Link Layer features and operation.

- **"Service Class svt_pcie_pl_service"** on page 274.
- **"UVM Component Class svt_pcie_pl" on page 286**
- **"PHY Layer Configuration Class" on page 287**

The following section lists and describes the members and tasks of the major classes to implement various features for your testbench.

Attention Note, the descriptions for the classes and applications show the most important and often used members/features. Consult the PCIe UVM HTML Class Reference for a complete listing of the members and their data types.

16.2 Additional Documentation on PHY Programming Tasks

Consult the following to get information on PHY related programming tasks.

- "SolvNet PCIe VIP Articles" on page 533.

- PCIe SVT FAQ

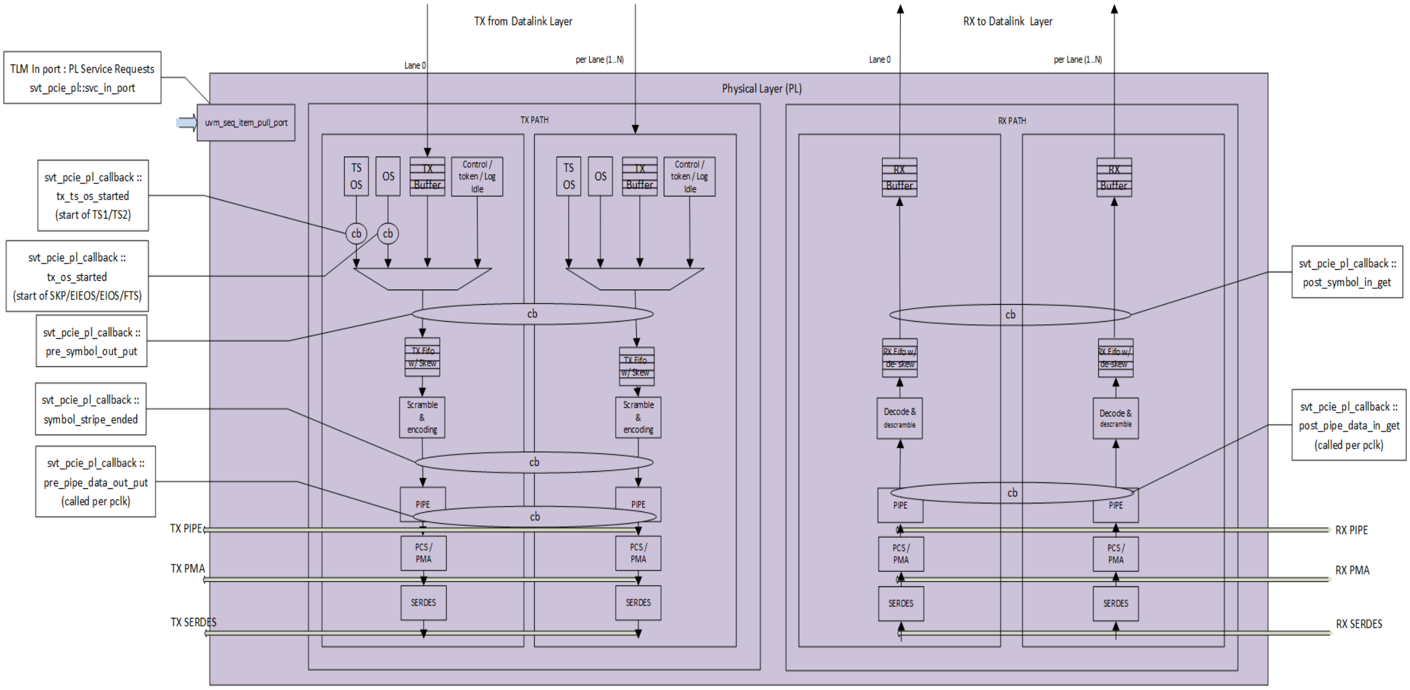
16.3 PHY Layer Data Flow

The Physical layer of the VIP performs scrambling and encoding of TLP, DLLP, and control/idle commands into OS/TS/OS/symbols to be driven on the PIPE, PCS/PMA, or SERDES. Similarly, it can receive data on one of the interfaces, decode and de-scramble and pass to the Datalink Layer.

16.3.1 Flow Diagram

Figure 16-1 highlights the key components of data flow through the VIP's Physical Layer. Available TLMs and callbacks (cb) are shown in the diagram.

Figure 16-1 Flow Diagram – PHY Layer



16.3.2 Service Class svt_pcie_pl_service

This transaction class supports all of the Service requests which can be processed by the PHY Layer.

Table 16-1 Class svt_pcie_pl_service

Member	Description
assert_clkreq_n	Controls whether the VIP will assert bidirectional signal clkreq_n or not. Note there is a soft pullup if clkreq is not asserted When '1' clkreq_n will be asserted (ie driven to 'b0). When '0' clkreq_n will not be asserted by the VIP.
corrupt_disparity_byte_wt[16]	Automatic OS corruption weight, Tx OS Bytes 0-15.
corrupt_lane_mask	Automatic OS corruption and FORCE_LANE_TX_ELEC_IDLE lane mask.
corrupt_tx_idle_data_enable	Automatic EIOS corruption percentage.
corrupt_tx_os_percentage	Automatic OS corruption percentage. Following is the list of service types that gets affected by this parameter: • `SET_TX_EIOS_CORRUPTION` • `SET_TX_EIEOS_CORRUPTION` • `SET_TX_FTS_CORRUPTION` • `SET_TX_SKP_CORRUPTION` • `SET_TX_SDS_CORRUPTION` • `SET_TX_EDS_CORRUPTION`

Table 16-1 Class svt_pcie_pl_service (Continued)

Member	Description
cursor_coeff	Cursor value for an equalization request.
direction_change_response	Direction change response.
ei_bytenum	User Task Tx TS EI Byte Number.
ei_code	User Task Tx TS EI Code.
ei_tx_phy_data_bit_flip_weight	Tx Data Pattern Bit Flip weighting - weighting to replace outgoing data with an invalid codeword when an error is injected. 2.5G/5G only.
ei_tx_phy_data_corrupt_data_weight	Tx Data Pattern Corrupt Data weighting - weighting to replace outgoing data with random data when an error is injected.
ei_tx_phy_data_corrupt_disparity_weight	Tx Data Pattern Corrupt Disparity weighting - weighting to replace outgoing data with an invalid codeword when an error is injected. 2.5G/5G only.
ei_tx_phy_data_invalid_codeword_weight	Tx Data Pattern Invalid Codeword weighting - weighting to replace outgoing data with an invalid codeword when an error is injected. 2.5G/5G only.
ei_tx_phy_data_lane_mask	Tx Data Pattern Lane mask- Each bit corresponds to a lane. Lanes with a 'b0 will not have errors injected.
ei_tx_phy_data_pattern_enable	Tx Data Pattern Enable - enable the phy data error pattern.
eq_lane_num	Programs the lane number for equalization information.
expect_reject	Reject response from link partner.
figure_of_merit	Figure Of Merit response.
hot_plug_mode	This attribute controls Hot plug mode. Table 16-2 on page 279 shows the various hot plug modes.
hot_reset_mode	This attribute controls Hot reset mode. Refer to Table 16-3 on page 279 for a listing of hot reset modes.
internal_condition	Type of internal condition. Refer to Table 16-4 on page 279 for a listing of internal conditions.
invalid_codeword_byte_wt[16]	Automatic OS corruption weight, Tx OS Bytes 0-15.
invalid_data_byte_wt[16]	Automatic OS corruption weight, Tx OS Bytes = 0-15.
lane_enabled	Set to enable the association.
lane_num	Lane Number.
loopback_enable	Initiate Loopback Enable.
max_ei_tx_phy_data_pattern_burst	Max Tx Data Pattern Burst - maximum number of symbols in an error burst.

Table 16-1 Class svt_pcie_pl_service (Continued)

Member	Description
max_ei_tx_phy_data_pattern_spacing	Max Tx Data Pattern Spacing - max number of symbols between error bursts.
max_user_tx_ts_burst	User Task Max Tx TS Burst.
max_user_tx_ts_spacing	User Task Max Tx TS Spacing.
min_ei_tx_phy_data_pattern_burst	Min Tx Data Pattern Burst - minimum number of symbols in an error burst.
min_ei_tx_phy_data_pattern_spacing	Min Tx Data Pattern Spacing - min number of symbols between error bursts.
min_user_tx_ts_burst	User Task Min Tx TS Burst.
min_user_tx_ts_spacing	User Task Min Tx TS Spacing.
phy_enable	When clear the LTSSM will attempted to enter the DISABLED state. This is not the same thing as turning off the LTSSM!!! To completely disable the LTSSM the hot_plug_mode should be set to HOT_PLUG_UNPLUG.
phy_id	Phy number to configure.
phy_response_code	Reject response from link partner.
postcursor_coeff	Post Cursor value for an equalization request.
precursor_coeff	Pre Cursor value for an equalization request.
preset_valid	Preset valid for an equalization request.
preset_value	Preset value for an equalization request.
reject_coefficient_preset_requests	For the side that is receiving requests during equalization, setting this will force rejection of incoming requests. clearing this bit will allow requests to once again be accepted.
service_type	Transaction command. Consult Table 16-5 on page 279 for a complete listing of all the service requests supported by the VIP.
status	Status information about the current processing state.
symbol<0-15>	User Task Tx TS Symbol <0-15>
tx_deemph	Tx Deemphasis Value
user_tx_ts_enable	User Task Tx TS Enable.
pipe_get_local_preset_coefficients	Enables MPIPE VIP to assert `GetLocalPresetCoefficients` signal for one PCLK.
go_to_l1ss	Directs VIP's LTSSM to go to the specified L1 sub-state (L1.0/L1.1/ L1.2). This service request is effective only when LTSSM is in L1.Entry state.

Table 16-1 Class svt_pcie_pl_service (Continued)

Member	Description
pipe_change_l1ss_powerstate	Drives additional user-specific PowerDown value(s) in L1.1/L1.2.Idle LTSSM sub-states.
pipe_eject_reset	De-asserts PIPE `Reset#` signal prior to PCLK getting ON from PHY (toggling of PCLK) and after VIP reset is complete.
training_control_lane_vector	Indicates on which lanes VIP will set the Training Control bit to 1 while transmitting TS1 OS in `Loopback, Disabled`, and `HotReset` LTSSM states when it entered these states from `Recovery.Idle`. Each bit of this configuration corresponds to a lane.Bit 0 corresponds to lane 0, bit 1 corresponds to lane 1 and so on. • If a bit is set to 1 on a lane, then VIP will set the training control bit 0, 1 or 2 to 1 in TS1 OS on that particular lane. • If a bit is set to 0 on a lane, then VIP will set the training control bit 0, 1 or 2 to 0 in TS1 OS on that particular lane. The default value of this configuration is 32'hffff_ffff. This configuration will be reset to its default value of 32'hffff_ffff each time the LTSSM state is changed from `Loopback, Disabled`, and `HotReset` to the next LTSSM state. **Note** This configuration should only be used when VIP is going to `Loopback`, `Disabled`, and `HotReset` from `Recovery.Idle` and should not be used when going to these states from `ConfigLwStart`.
directed_ltssm_state	Defines the directed LTSSM state transition combinations. This can be used along with `INITIATE_DIRECTED_LTSSM_TRANSITION` service type to direct the VIP to next spec defined LTSSM state. Currently, it only supports `Recovery.Idle` to `Config.Lw.Start` directed transition. Note: `INITIATE_DIRECTED_LTSSM_TRANSITION` service type along with `RECOVERY_TO_CONFIGURATION` must not be used to do link width up/down sizing. For link width up/down sizing, use the `INITIATE_LINK_WIDTH_CHANGE` service type.
`enable_loopback_w_equalization`	This bit directs the VIP which is acting as loopback Master to enter Equalization from `Loopback.Entry`. Note: This is applicable for Gen5 devices only.
`loopback_lane_under_test`	This control defines the physical lane under test on which the 32 GT/s equalization procedure will be performed before entering `loopback.active`. Note: This is applicable for Gen5 devices only.
`set_ts_transmit_modified_compliance_pattern_in_loopback`	This bit sets the value of the `transmit_modified_compliance_pattern_in_loopback` bit in the training control field of transmitted training sets. This is for use by the loopback Master only in the case where loopback equalization will be performed. Note: This is applicable for Gen5 devices only.

Table 16-1 Class svt_pcie_pl_service (Continued)

Member	Description
upstream_port_loopback_tx_preset_in_eq_ts1	Specifies the transmitter preset(symbol 6 bit[6:3]) value of EQ TS1 OS that gets transmitted by VIP(Upstream Port) as Loopback Master in Loopback Entry at 2.5G and 5G. **Note** This is applicable only for Upstream Port active VIP, that is – when VIP acts as a Loopback Master and the value of `enable_eq_ts1_in_loopback_state` configuration variable is set to 1.
upstream_port_loopback_rx_preset_hint_in_eq_ts1	Specifies the receiver preset hint(symbol 6 bit[2:0]) value of EQ TS1 OS that gets transmitted by VIP(Upstream Port) as Loopback Master in Loopback Entry at 2.5G and 5G. **Note** This is applicable only for Upstream Port active VIP, that is – when VIP acts as a Loopback Master and the value of `enable_eq_ts1_in_loopback_state` configuration variable is set to 1.
upstream_port_tx_preset_in_loopback_state	Specifies the transmitter preset(symbol 6 bit[6:3]) value of TS1 OS that gets transmitted by Upstream Port(EP) as Loopback Master in Loopback Entry at 8G. **Note** This is applicable only for active VIP, that is – when VIP acts as a Loopback Master and it is configured as an Endpoint.
upstream_port_tx_preset_in_loopback_state_16g	Specifies the transmitter preset(symbol 6 bit[6:3]) value of TS1 OS that gets transmitted by Upstream Port(EP) as Loopback Master in Loopback Entry at 16G. **Note** This is applicable only for active VIP, that is – when VIP acts as a Loopback Master and it is configured as an Endpoint.
upstream_port_tx_preset_in_loopback_state_32g	Specifies the transmitter preset(symbol 6 bit[6:3]) value of TS1 OS that gets transmitted by Upstream Port(EP) as Loopback Master in Loopback Entry at 32G. **Note** This is applicable only for active VIP, that is – when VIP acts as a Loopback Master and it is configured as an Endpoint.

The following table shows the various types of hot plug modes. The member "hot_plug_mode" sets the mode type.

Table 16-2 Hot Plug Modes

Mode	Description
HOT_PLUG_UNPLUG	Hot plug mode is unplug. Applicable to VIP in active and passive modes.
HOT_PLUG_MONITOR	Hot plug mode is monitor. Only applicable to VIP in active mode.
HOT_PLUG_WAIT	Hot plug mode is wait. Only applicable to VIP in active mode.
HOT_PLUG_DETECT	Hot plug mode is detect. Only applicable to VIP in active mode.

The following table list the various hot reset modes. The member "hot_reset_mode" sets the reset mode.

Table 16-3 Hot Reset Modes

Mode	Description
HOT_RESET_INACTIVE	LTSSM is not instructed to enter reset. If LTSSM initiated the hot reset and is in the hot reset state, setting to inactive will direct the LTSSM out of hot reset.
HOT_RESET_FORCE	Instruct the LTSSM to enter hot reset and remain in this state.
HOT_RESET_WAIT	Instruct the LTSSM to enter hot reset, then wait for the attached link to enter hot reset. Once the attached link has shut down its transmitters LTSSM will automatically exit to detect and reset the hot reset mode to INACTIVE.

The following table lists various internal conditions. The member "internal_condition" sets these conditions.

Table 16-4 Internal Condition States

Internal Condition	Description
INT_COND_RX_PATH_BLOCK_ALIGNMENT_ACHIEVED	Indicates Block alignment status on RX path.
INT_COND_TX_PATH_BLOCK_ALIGNMENT_ACHIEVED	Indicates Block alignment status on TX path.
INT_COND_RX_PATH_8G_SPEED_FIRST_PKT_RECEIVED	Received first packet on RX path at 8G speed.
INT_COND_TX_PATH_8G_SPEED_FIRST_PKT_RECEIVED	Received first packet on TX path at 8G speed

The following table shows the service type calls available through the "service_type"member.

Table 16-5 Service Requests

Requested Service	Description
SET_PHY_ENABLE(0)	When clear, LTSSM will attempt to enter the DISABLED state through the exchange of TS.

Table 16-5 Service Requests (Continued)

Requested Service	Description
INITIATE_LINK_WIDTH_CHANGE(1)	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.
INITIATE_RETRAIN_LINK(2)	Direct the LTSSM to Recovery. This service task has same functionality as setting retrain_link bit in PCIE Cfg's Link Control Register.
INITIATE_SPEED_CHANGE(3)	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.
INITIATE_LOOPBACK(4)	Instructs the VIP to enter loopback as a loopback master.
CONFIGURE_LANE(5)	This service 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- CONFIGURE_LANE(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
SET_USER_TX_TS1(6)	Defines a user training set and optionally an error injection for a given lane
SET_USER_TX_TS1_PATTERN(7)	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.
SET_EI_TX_TS1_PATTERN(8)	Defines a constrained random burst of TS1 ordered sets with errors.
SET_USER_TX_TS2(9)	Defines a user training set and optionally an error injection for a given lane.
SET_USER_TX_TS2_PATTERN(10)	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.

Table 16-5 Service Requests (Continued)

Requested Service	Description
SET_EI_TX_TS2_PATTERN(11)	Defines a constrained random burst of TS2 ordered sets with errors.
REQUEST_HOT_PLUG_MODE(12)	Allows users to simulate unplugging of VIP from the link. During a hot plug event all the transmitters are shut off and the LTSSM will go back to Detect. The VIP will not respond until it is plugged back in. In a passive VIP, LTSSM goes to 'Searching...' state and waits until it detects electrical idle on its RX and TX paths. While in 'Searching...' state monitor does not evaluate any protocol check. As soon as electrical idle is entered, LTSSM goes to Detect state and resumes normal operation.
PERFORM_EQUALIZATION(13)	Sets perform_equalization variable. This is used by LTSSM to determine if equalization should be performed during 8 GT/s speed negotiation or otherwise
REQUEST_EQUALIZATION(14)	Sets request_equalization variable. This is used by LTSSM to determine if equalization should be requested while in 8GT/s speeds.
QUEUE_EQ_FIGURE_MERIT_RESPONSE(15)	Sets a Figure Of Merit response from equalization evaluation. Response ranges from 0 to 255. If no responses are queued then the default value of 255 is returned as a figure of merit value. This is applicable only when VIP operates in PHY PIPE mode.
QUEUE_EQ_DIRECTION_CHANGE_RESPONSE(16)	Sets the direction change response from equalization evaluation. If no responses are queued, the default response used is no change on any of the coefficients.
QUEUE_EQ_TX_REQUEST_PRESET_COEFF(17)	Queue either a Preset or a coefficient request. VIP sends this request in TS1 at 8GT/s in Recovery.Equalization. As long as a request is queued, VIP 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.
SET_HOT_RESET_MODE(18)	Allows users to request the LTSSM to enter/exit hot reset.
SET_REJECT_COEFFICIENT_PRESET_REQUESTS(19)	Controls whether incoming requests during equalization are automatically rejected or accepted
SET_EI_TX_PHY_DATA_PATTERN(20)	When enabled, the phy will periodically inject errors into the data stream.
ASSERT_CLKREQ_N(21)	Controls whether clkreq_n is asserted or not. If not asserted the soft pullup will drive it to 'b1

Table 16-5 Service Requests (Continued)

Requested Service	Description
MON_INITIATE_L1_2_EXIT(22)	Directs monitor LTSSM to exit from L1.2.Exit and transition to L1.Idle. It is applicable to a passive monitor instance monitoring either Endpoint or Root Complex. If LTSSM is already in L1.2.Exit state, then LTSSM transitions from this state to L1.Idle as soon as it is directed. If LTSSM is not in L1.2.Exit, then the service request is not effective immediately, but when LTSSM enters L1.2.Exit next time, LTSSM transitions from this state to L1.Idle immediately as it has been already directed to do so.
MON_INITIATE_L2_EXIT(23)	Directs monitor LTSSM to exit from L2.Idle state and transition to Detect.Quiet. It is applicable only to a passive monitor instance monitoring Root Complex. If LTSSM is already in L2.Idle state then LTSSM transitions from this state to Detect.Quiet as soon as it is directed. If LTSSM is not already in L2.Idle state then the service request is not effective immediately, but when LTSSM enters L2.Idle next time, LTSSM transitions from L2.Idle state to Detect.Quiet immediately as it has been already directed to do so.
MON_GOTO_L2_TRANSMIT_WAKE(24)	Directs monitor LTSSM to transition to L2.Transmit.Wake from L2.Idle. It is applicable only to a passive monitor instance monitoring Endpoint. If LTSSM is already in L2.Idle state, then LTSSM transitions from this state to L2.Transmit.Wake as soon as it is directed. If LTSSM is not in L2.Idle, then the service request is not effective immediately, but when LTSSM enters L2.Idle next time, LTSSM transitions from this state to L2.Transmit.Wake immediately as it has been already directed to do so.
SET_TX_IDLE_DATA_CORRUPTION(25)	Set automatic corruption of TX Idle Data. Corruption of Idle Data is controlled by percentage. * If Idle Data is selected, actual byte to corrupt is based on weightings. * Byte corruption weightings available: * invalid_data_byte[0] (all speeds) * corrupt_disparity_byte[0] (2.5-5G only) * invalid_codeword_byte[0] (2.5-5G only) * Idle Data corruption is enabled via lane mask.
SET_TX_EIOS_CORRUPTION(26)	Set automatic corruption of TX EIOS. Corruption of EIOS is controlled by percentage. * If EIOS is selected, actual byte to corrupt is based on weightings. * Byte corruption weightings available: * invalid_data_byte[0-15] (all speeds) * corrupt_disparity_byte[0-3] (2.5-5G only) * invalid_codeword_byte[0-3] (2.5-5G only) * EIOS corruption is enabled via lane mask.

Table 16-5 Service Requests (Continued)

Requested Service	Description
SET_TX_EIEOS_CORRUPTION(27)	Set automatic corruption of TX EIEOS. Corruption of EIEOS is controlled by percentage. * If EIEOS is selected, actual byte to corrupt is based on weightings. * Corruption mechanism selected by weightings: * invalid_data_byte[0-15] (all speeds) * corrupt_disparity_byte[0-15] (2.5-5G only) * invalid_codeword_byte[0-15] (2.5-5G only) * EIEOS corruption is enabled via lane mask.
SET_TX_FTS_CORRUPTION(28)	Set automatic corruption of TX FTS. Corruption of FTS is controlled by percentage. * If FTS is selected, actual byte to corrupt is based on weightings. * Corruption mechanism selected by weightings: * invalid_data_byte[0-15] (all speeds) * corrupt_disparity_byte[0-3] (2.5-5G only) * invalid_codeword_byte[0-3] (2.5-5G only) * FTS corruption is enabled via lane mask.
SET_TX_SKP_CORRUPTION(29)	Set automatic corruption of TX SKP. Corruption of SKP is controlled by percentage. * If SKP is selected, actual byte to corrupt is based on weightings. * Corruption mechanism selected by weightings: * invalid_data_byte[0-15] (all speeds) * corrupt_disparity_byte[0-3] (2.5-5G only) * invalid_codeword_byte[0-3] (2.5-5G only) * SKP corruption is enabled via lane mask.
SET_TX_SDS_CORRUPTION(30)	Set automatic corruption of TX SDS. Corruption of SDS is controlled by percentage. * If SDS is selected, actual byte to corrupt is based on weightings. * Corruption mechanism selected by weightings: * invalid_data_byte[0-15] (8G only) * SDS corruption is enabled via lane mask.
SET_TX_EDS_CORRUPTION(31)	Set automatic corruption of TX EDS. CCorruption of EDS is controlled by percentage. * If EDS is selected, actual byte to corrupt is based on weightings. * Corruption mechanism selected by weightings: * invalid_data_byte[0-3] (8G only).
MON_INITIATE_DISABLED_EXIT(32)	Directs monitor LTSSM to exit from Disabled state and transition to Detect.Quiet. It is applicable only to a passive monitor instance monitoring Root Complex. If LTSSM is already in Disabled state then LTSSM transitions from this state to Disabled as soon as it is directed. If LTSSM is not already in Disabled state then the service request is not effective immediately, but when LTSSM enters Disabled next time, LTSSM transitions from Disabled state to Detect.Quiet immediately as it has been already directed to do so.

Table 16-5 Service Requests (Continued)

Requested Service	Description
MON_SET_INTERNAL_CONDITION(33)	Sets internal conditions for monitor. The condition represents monitor's internal status * which monitor uses for tracking link activities. This service request should not be used * during normal operations. It is intended to be used in situations where the monitor cannot * track the LTSSM correctly due to observation delays. This service request can be used to * provide hints to the monitor that will allow tracking to continue.
MON_GET_INTERNAL_CONDITION(34)	Returns monitor's internal condition's current value. This is intended to be used * as a debug feature.
MON_INITIATE_LINK_WIDTH_SIZING(35)	Request the monitor to enter CONFIGURATION_LINKWIDTH_START as soon as it goes * to RECOVERY_IDLE.
MON_INITIATE_HOT_RESET_EXIT(36)	Directs monitor LTSSM to exit from Hot Reset state and transition to Detect.Quiet. It is applicable only to a passive monitor instance monitoring Root Complex. If LTSSM is already in Hot Reset state then LTSSM transitions from this state to Hot Reset as soon as it is directed. If LTSSM is not already in Hot Reset state then the service request is not effective immediately, but when LTSSM enters Hot Reset next time, LTSSM transitions from Hot Reset state to Detect.Quiet immediately as it has been already directed to do so.
MON_INITIATE_LINK_SPEED_NEGOTIATION(37)	Directs monitor LTSSM to enter Recovery.Rcvrlock from L0 state. It also sets directed_speed_change_variable to 1'b1. If LTSSM is already in L0 state then LTSSM transitions from this state to Recovery as soon as it is directed. If LTSSM is not already in L0 state then the service request is not effective immediately, but when LTSSM enters L0 next time, LTSSM transitions from L0 state to Recovery.Rcvrlock immediately as it has been already directed to do so.
MON_GOTO_POLLING_COMPLIANCE(38)	Directs monitor LTSSM to transition to Polling.Compliance from Polling.Active. It is applicable only to a passive monitor instance. If LTSSM is already in Polling.Active state, then LTSSM transitions from this state to Polling.Compliance as soon as it is directed. If LTSSM is not in Polling.Active, then the service request is not effective immediately, but when LTSSM enters Polling.Active next time, LTSSM transitions from this state to Polling.Compliance immediately as it has been already directed to do so.

Table 16-5 Service Requests (Continued)

Requested Service	Description
MON_INITIATE_POLLING_COMPLIANCE_EXIT(39)	Directs monitor LTSSM to exit from Polling.Compliance state and transition to Detect.Quiet. It is applicable only to a passive monitor instance. If LTSSM is already in Polling.Compliance state then LTSSM transitions from this state to Detect.Quiet as soon as it is directed. If LTSSM is not already in Polling.Compliance state then the service request is not effective immediately, but when LTSSM enters Polling.Compliance next time, LTSSM transitions from Polling.Compliance state to Detect.Quiet immediately as it has been already directed to do so.
SET_LANE_TX_DEEMPH(40)	Drives the tx_deemph pins on the pipe bus to the specified value at 2.5G and 5G.
QUEUE_PIE8_EQ_TX_REQUEST_PRESET_COEFF(41)	Queue either a Preset or a coefficient request in PIE8 mode. VIP sends this request in TS1 at 8GT/s in Recovery.Equalization. As long as a request is queued, VIP 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.
FORCE_LANE_TX_ELEC_IDLE(42)	Forces the specified lanes to shutdown and transmit electrical idle. corrupt_lane_mask[31:0] is a bit-mapped vector where corrupt_lane_mask[0] control physical lane 0 and corrupt_lane_mask[31] control physical lane 31. When corrupt_lane_mask[n] is 1'b1, the lane is turned OFF by forcing electrical idle on the lane. When corrupt_lane_mask[n] is 1'b0, the lane is turned back ON if it was forced to enter electrical idle earlier. When corrupt_lane_mask[n] is 1'b0 and if the lane was not forced to enter electrical idle earlier, then this exception doesn't have any effect on lane n. Lanes are not turned OFF immediately at the same symbol time, but they are shut down or forced to electrical idle after a programmable latency. The latency is programmed using an attribute namely svt_pcie_pl_configuration :: tx_shutdown_latency.
INITIATE_DIRECTED_LTSSM_TRANSITION	Directs the VIP to next LTSSM state in a directed manner. Current LTSSM state to next LTSSM state is specified by `directed_ltssm_state` enum.

16.3.3 Physical Layer Callbacks and Exceptions

The Datalink Layer provides a callback class, svt_pcie_pl_callback, for observation and application of exceptions to both incoming and outbound transactions. Refer to the Flow Diagram – PHY Layer where the Tx and Rx callbacks are applied. For more information on the PL callbacks and exceptions, see Chapter 19.6.

16.4 UVM Component Class svt_pcie_pl

This class is UVM Driver that implements Physical layer module. The class is responsible to reconfigure PCIE SVC Physical layer module. It is also responsible to provide status of the application. It provides a SIPP [Sequence Item Pull Port] to cater to services of type svt_pcie_pl_service. Note, the class supports all UVM phases.

Table 16-6 Class svt_pcie_pl

Member	Description
pre_symbol_out_put	Called by the component after gathering all the symbols to be transmitted on the PCIe link. This is the last chance to the user to corrupt any symbol before it goes on the link.
reconfigure_via_task	Depreciated.
tx_os_started	Called by the component after building an OS Transaction. The callback gives user a chance to attach exception list to the OS transaction prior to it transmission on the link.
tx_ts_os_started	Called by the component after building a TS OS Transaction. The callback gives user a chance to attach exception list to the TS OS transaction prior to it transmission on the link.
os_xact_exception	Randomization factory to create TX OS exception (error etc.) to be inserted in transaction.
os_xact_exception_list	Randomization factory to create TX OS exception list for an OS transaction.
svc_in_port	PL Service TLM Sequence Item Pull Port. Provides a mechanism for submitting PL Service transactions recognized by the PL Layer. The handle to this TLM sequence item pull port can be set or obtained through the driver's public member svc_in_port.
symbol_exception	Randomization factory to create TX symbols exception (error etc.) to be inserted in symbols.
symbol_exception_list	Randomization factory to create TX symbols exception list for symbols to be transmitted.
ts_os_xact_exception	Randomization factory to create TX TS OS exception (error etc.) to be inserted in transaction.
ts_os_xact_exception_list	Randomization factory to create TX TS OS exception list for a TS OS transaction.

16.5 PHY Layer Configuration Class

Table 16-7 PHY Layer Functions and Configuration Members

Member	Description
Functions	
enable_autonomous_16g_equalization	When set to 1, VIP maintains default behavior of carrying out autonomous 16G equalization procedure. When set to 0, the VIP does not carry out autonomous 16G equalization procedure. NOTE. When set to 0, user must call PERFORM_EQUALIZATION service sequence from the VIP (Downstream Port) to execute software based 16G equalization procedure.
get_expected_link_speed_value	This function returns the expected link speed value.
get_expected_link_width_value	This function returns the current expected link width. NOTE: This function does not return the deferred expected link width value. Please see the NOTE for set_link_width_values function which describes when the link width settings are deferred.
get_link_width_value	This function returns the current maximum link width setting supported by VIP. NOTE: This function does not return the deferred maximum link width value. Please see the NOTE for set_link_width_values function which describes when the link width settings are deferred.
get_supported_link_speeds_value	This function returns the supported link speeds.
get_supported_link_width_vector_value	This function returns the current supported link width vector. NOTE: This function does not return the deferred supported link width vector value. Please see the NOTE for set_link_width_values function which describes when the link width settings are deferred
get_target_link_speed_value	This function returns the target link speed value.
is_link_speed_settings_valid	Checks to see that link speed settings follow PCIe protocol requirements.
get_supported_link_width_vector_value	This function returns the current supported link width vector. NOTE: This function does not return the deferred supported link width vector value. Please see the NOTE for set_link_width_values function which describes when the link width settings are deferred.
get_target_link_speed_value	This function returns the target link speed value.
is_link_speed_settings_valid	Checks to see that link speed settings follow PCIE protocol requirements.
is_link_width_settings_valid	Checks to see that link width settings follow PCIE protocol requirements.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
set_link_speed_values	Sets the values of the link speed settings of VIP. NOTE: Link speed settings should not be reconfigured while VIP is in LTSSM Configuration.Complete state. VIP issues a warning when attempted to reconfigure while LTSSM is in Configuration.Complete state and reconfigured values are ignored. • supported_link_speeds_value - It represents all the possible link speeds which are supported during link retraining. If link training is bypassed, the highest supported speed is always used. • target_link_speed_value - Sets the desired speed of operation for a downstream port. For an upstream port the target link speed is not applicable. When unset, the target link speed is automatically updated to the highest supported link speed for a downstream port. • expected_link_speed_value - The expected negotiated link speed value. The expected link speed check is performed only after LTSSM is in L0 state and a downstream port has transmitted a SDP packet. When unset, it is automatically updated to be same as the target link speed.
set_link_width_values	Sets the values of the link width settings of VIP. NOTE: Link width settings are applicable instantly when VIP LTSSM is not in L0 state. If link width settings are reconfigured while VIP LTSSM is in L0 state, then link width settings are deferred until LTSSM transitions out of L0 state. The deferred values can be reconfigured again to override initial deferred values of link width settings until LTSSM continues to stay in L0 state. • link_width_value - The maximum link width the LTSSM is allowed to negotiate. When link training is bypassed, VIP uses maximum link width as negotiated link width. • supported_link_width_vector_value - All the possible link widths a VIP can support from 1 upto link_width value. When unset in the function call, it prompts VIP to set supported_link_width_vector to support all the possible link widths from 1 upto link_width value. • expected_link_width_value - The expected negotiated link width value. This value cannot be greater than the maximum link width value as represented by link_width variable. When unset in the function call, it prompts VIP to set expected_link_width value same as the value of link_width argument.
set_full_ltssm_state_timeout_values	Sets all of the LTSSM state timeouts to the values as mentioned in the specification.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
set_fast_sim_ltssm_state_timeout_values	Sets all of the LTSSM state timeouts for the fast simulation mode according to the scaling factor provided for our in-house IIP. • 0 – (4 ^ 0) / 1000 • 1 – (4 ^ 1) / 1000 • 2 – (4 ^ 2) / 1000 • 3 – (4 ^ 3) / 1000
program_default_skip_interval_for_sris_mode	Sets the default SKIP interval for SRIS mode.
Attributes	
allow_l0_exit_truncate_tx_packet	When set the LTSSM will exit L0 only when the transmitter is idle and not sending a TLP/DLLP during when receiving a compliance pattern
assert_turn_off_signaling_continuously	When the mpipe decides to use signaling to turn off lanes that aren't part of the link, setting this to a 1 makes the mpipe drive the turn off signaling continuously. A value of zero will cause the turn off signaling to occur for one pclk cycle.
attached_fs[32]	Programs the attached_fs value driven by VIP in PIPE mode for 8g when the VIP is the MAC.
attached_fs_16g[32]	Programs the attached_fs value driven by VIP in PIPE mode for 16g when the VIP is the MAC.
attached_lf[32]	Programs the attached_lf value driven by VIP in PIPE mode for 8g when the VIP is the MAC.
attached_lf_16g[32]	Programs the attached_lf value driven by VIP in PIPE mode for 16g when the VIP is the MAC.
configuration_complete_timeout_ns	Timeout value in the Configuraton.Complete state
configuration_idle_timeout_ns	Timeout value in the Configuration.Idle state.
configuration_lanenum_wait_timeout_ns	Timeout value in the Configuration.:Lanenum.Wait state.
configuration_linkwidth_accept_timeout_ns	Timeout value in the Configuration.Linkwidth state.
configuration_linkwidth_start_timeout_ns	Timeout value in the Configuration.Linkwidth state.
detect_active_timeout_ns	Timeout interval for the Detect.Active state.
detect_active_txdetectrx_spacing_ns	Spacing between receiver detects in detect.active when the first receiver detect does not return a receiver present on all configured lanes.
detect_quiet_speed_change_ns	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.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
detect_quiet_timeout_ns	Timeout interval for Detect.Quiet state.
disable_scrambling	When set to 1'b1 scrambling will be disabled for all speeds, and the LTSSM will set the disable scrambling bit on all transmitted TS1 and TS2 ordered sets. Note: This is applicable only for active VIP.
disable_128b_130b_scrambling	Specifies if scrambling is disabled for 8G and above speeds. • When set to 1'b0, scrambling is not disabled for 8G and above speeds. • When set to 1'b1, scrambling is disabled for 8G and above speeds. Note: This is applicable only for passive VIP.
disabled_timeout_ns	Time before a downstream component will exit the Disabled state if no EIOS is detected
downstream_lanes_recovery_eq_phase1_timeout_ns	Timeout in NS for downstream lanes in Recovery.Equalization.Phase1.
downstream_lanes_recovery_eq_phase2_timeout_ns	Timeout in NS for downstream lanes in Recovery.Equalization.Phase2.
downstream_lanes_recovery_eq_phase3_timeout_ns	Timeout in NS for downstream lanes in Recovery.Equalization.Phase3.
downstream_preset_value	Downstream Preset value. Applicable only for devices using downstream ports at 8G.
downstream_preset_value_16g[32]	Downstream Preset value. Applicable only for devices using downstream ports at 16G.
downstream_receiver_preset_hint[32]	The variable is applicable to a device with downstream ports only. Downstream ports advertise this value in the transmitted EQ TS1s in equalization phase 0 at 8G.
downstream_receiver_preset_hint_16g[32]	The variable is applicable to a device with downstream ports only. Downstream ports advertise this value in the transmitted EQ TS1s in equalization phase 0 at 16G.
dut_receiver_present	For SPIPE, serdes and PMA models, this indicates which lanes the VIP will see as having a receiver present when the receiver detect is done in detect.active. Each bit corresponds to a lane with bit 0 corresponding to lane 0, bit 1 to lane 1 and so on. Not valid for MPIPE models-- MPIPE will use the receiver detect mechanism defined in the PIPE interface
enable_auto_directed_speed_change	This attribute when set to 1'b1 enables automatic speed change when advertised supported speed is greater than 2.5G on transmitted and received TS Ordered Sets.
enable_equalization_coefficients_checks	Enables equalization coefficients check.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
enable_equalization_verification_mode	Enables equalization verification mode.
enable_extended_synch	When set, the transmitter for each lane must transmit 1024 TS1 ordered sets before transitioning from Recovery.RcvrLock to Recover.RcvrConfig.
enable_l1_clk_power_management	When set to 0, the LTSSM will keep clkreq# asserted in the L1 low power state. When set to a 1, the LTSSM will deassert clkreq# when entering the L1 low power state. Note that per the pipe spec if clkreq# is deasserted then the LTSSM should transition to P2 instead of P1. This control is valid only when L1 substate support is not enabled.
enable_ltssm_transition_when_compliance_bit_set	When set to 0 the LTSSM will behave normally and transition out of polling.active regardless of whether or not the compliance receive bit is set. When set to 1 (default) the LTSSM will remain in polling.active to test the case where when the DUT is receiving the compliance receive bit in polling.active it should wait until polling.active timeout and then go to polling.compliance.
enable_per_lane_spipe_phystatus_rand_delay	When set to a 1 the SPIPE VIP will randomize per lane phy phystatus asserted back to the DUT to signal completion of power state changes, speed changes, receiver detect, etc. This control has no effect for PIPE 2 models and MPIPE models.
enable_pipe_reset_n_assertion_in_detect_quiet	This attribute controls automatic pipe_resetn assertion in detect.quiet to test the switching to P1 method for lane turn off feature.
enable_random_data_on_turn_off_lane	When set, the VIP will send random data on turned off lanes for mpipe.
enable_relaxed_skp_checking	When set, the SVC 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.
enable_rxeqeval_default_settings_vector	This variable controls which phase or phases the device will request an evaluation on the settings that exist on entry to equalization. For RC: Setting bit 0 enables an eq eval during phase 1. Setting bit 1 enables an EQ eval during phase 3. For EP: Setting bit 0 enables an eq eval during phase 0. Setting bit 1 enables an EQ eval during phase 2. If both bits 0 and 1 are set, then an eq eval will be performed in both phase0/2 (EP) or phase 1/3(RC).
enable_upconfigure_support	Enables support for upconfiguration.
enter_compliance	This attribute represents enter_compliance register field.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
eq_rx_reset_eieos_interval	During equalization phase 2 and 3, when the reset eieos interval bit is set on incoming TS1 this value controls the interval at which EIEOS are received.
expected_preset_to_coefficients_mapping_entry_enable	A bit mapped variable where bit N corresponds to entry N in expected_preset_to_coefficients_mapping_table. When bit is set to 1'b1, it indicates that corresponding entry in expected_preset_to_coefficients_mapping_table is valid and VIP should check incoming TS1s to show the correct coefficient values when a particular preset has been requested. For 8G speed only.
expected_preset_to_coefficients_mapping_entry_enable_16g	A bit mapped variable where bit N corresponds to entry N in expected_preset_to_coefficients_mapping_table. When bit is set to 1'b1, it indicates that corresponding entry in expected_preset_to_coefficients_mapping_table is valid and VIP should check incoming TS1s to show the correct coefficient values when a particular preset has been requested. For 16G speed only.
expected_preset_to_coefficients_mapping_table[16]	Preset to coefficients mapping table used to verify DUT's preset to coefficient mappings. The table should be programmed exactly like the DUT's preset table. For a passive VIP, the table should be programmed exactly like the monitored device's preset table. The table is indexed by a preset value. Mapping table stores coefficients values packed in the following format. { postcursor_coeff, cursor_coeff, precursor_coeff } Default value = { 6'h0c, 6'h24, 6'h00} == 18'h0c900. For 8G only.
expected_preset_to_coefficients_mapping_table_16g[16]	Preset to coefficients mapping table used to verify DUT's preset to coefficient mappings. The table should be programmed exactly like the DUT's preset table. For a passive VIP, the table should be programmed exactly like the monitored device's preset table. The table is indexed by a preset value. Mapping table stores coefficients values packed in the following format. { postcursor_coeff, cursor_coeff, precursor_coeff } Default value = { 6'h0c, 6'h24, 6'h00} == 18'h0c900. For 16G only.
fs_value[32]	Represents FS value advertised by VIP in TS1s during 8g equalization phase 1.
fs_value_16g[32]	Represents FS value advertised by VIP in TS1s during 16g equalization phase 1.
get_local_preset_coefficients_timeout_ns	Represents timeout in NS at which PIPE MAC times out waiting for PIPE PHY to respond to preset to coefficeints mapping request.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
highest_enabled_equalization_phase	Specifies highest enabled equalization phase. When set to 1, enables equalization phase0 and phae 1. When set to 3, enables equalization phases 0, 1, 2, and 3. Note: DUT must also be configured to bypass equalization states when this attribute is set to 0.
hot_reset_mode	This attribute is deprecated. To control Hot Reset mode, use the SET_HOT_RESET_MODE service type in the svt_pcie_pl_service class.
hot_reset_timeout_ns	Timeout value in Hot_Reset state.
idle_to_rlock_transitioned_variable	This attribute represents idle_to_rlock_transitioned_variable LTSSM variable.
inferred_electrical_idle_recovery_rcvrcfg_8g_time_ns	The amount of time in recovery.rcvrlock at 8G before the LTSSM will infer elec idle of no TS1 are received.
inferred_electrical_idle_skp_time_ns	The amount of time in L0 before the LTSSM will infer electrical idle if no SKP ordered sets are received.
invert_tx_polarity	The variable programs polarity inversion on all lanes. It is a 32-bit vector where bit 0 control polarity inversion on lane 0. When a bit is set, the corresponding lane will invert polarity on all outgoing data.
is_pie8_mode_master	Indicates whether the component is PIE8 master or PIE8 slave. When set to 1, acts as PIE8 master. When set to 0, acts as PIE8 slave
lane_reversal_mode	This attribute controls lane reversal mode
lf_value[32]	Represents LF value advertised by VIP in TS1s during 8g equalization phase 1.
lf_value_16g[32]	Represents LF value advertised by VIP in TS1s during 16g equalization phase 1.
link_number	This attribute represents link number used during link training.
local_fs[32]	Programs the local_fs value driven by VIP in PHY PIPE mode for 8g.
local_fs_16g[32]	Programs the local_fs value driven by VIP in PHY PIPE mode for 16g.
local_lf[32]	Programs the local_lf value driven by VIP in PHY PIPE mode for 8g.
local_lf_16g[32]	Programs the local_lf value driven by VIP in PHY PIPE mode for 16g.
loopback_exit_elec_idle_timeout_ns	Amount of time the LTSSM transmit electrical idle in Loopback.exit before entering detect.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
loopback_master_enter_compliance_tx_ts1_timeout_ns	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	Time before the loopback master enters detect if no TS1s are received w/ the enter compliance bit clear.
loopback_master_speed_change_timeout_ns	Speed change delay for loopback master.
loopback_slave_speed_change_timeout_ns	Speed change delay for loopback slave.
ltssm_transition_evaluation_delay_ns	Amount of time in NS the port will delay before transitioning to next state in the presence of errors.
max_configuration_lanenum_accept_timeout_ns	Time at which config.linkwidth.accept will make a decision on the linkwidth if errors are received.
max_eq_evaluation_delay[32]	Represents maximum time in NS to evaluate attached transmitter setting changes during equalization.
max_eq_preset_coeff_validation_delay[32]	Represents maximum amount of time VIP takes to either accept or reject a received preset/coefficient request.
max_num_pclk_cycles_to_assert_invalid_request	This variable controls the maximum number of cycles that InvalidRequest will be asserted upon detecting invalid direction change feedback from an EQ evaluation request.
max_num_rx_eieos_before_fts	Maximum number of EIEOS to receive before sending FTS.
max_num_tx_eieos_before_fts	Maximum number of EIEOS ordered sets transmitted before transmitting FTS. (5GT/s only).
max_num_tx_eios_before_l1_gen1	Maximum number EIOS to send to accept L1 for Gen1.
max_num_tx_eios_before_l1_gen2	Maximum number EIOS to send to accept L1 for Gen2.
max_num_tx_eios_before_l1_gen3	Maximum number EIOS to send to accept L1 for Gen3.
max_rx_eq_eval_delay[32]	Represents maximum amount of time in NS VIP in PHY PIPE mode takes to respond to RxEqEval request.
max_rx_lane_skew_16g	Maximum allowed lane skew before the SVC flags a skew violation on the receive side at 16G.
max_rx_lane_skew_2_5g	Maximum allowed lane skew before the SVC flags a skew violation on the receive side at 2.5G.
max_rx_lane_skew_5g	Maximum allowed lane skew before the SVC flags a skew violation on the receive side at 5G.
max_rx_lane_skew_8g	Maximum allowed lane skew before the SVC flags a skew violation on the receive side at 8G.
max_rx_skp_interval_in_blocks	Maximum number of blocks before the upper phy flags an error due to lack of a SKP ordered set (8GT/s).

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
max_rx_skp_interval_in_symbol_times	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).
max_spipe_phystatus_delay	Maximum number of PIPE CLK cycles the SVC will wait before asserting phystatus indicating completion of a phy function (for example, power state change or rate change). Valid only for SPIPE models.
max_spipe_phystatus_p2_exit_change_delay	Maximum number of PIPE CLK cycles that the SVC will keep phystatus asserted upon exiting from the P2 power state. Valid for SPIPE only.
max_spipe_preset_coefficients_delay	Represents a maximum delay time by PIPE PHY to respond to preset request by returning the coefficients request.
max_tx_lane_skew_16g	Sets the maximum lane skew between transmit lanes in symbol times at 16Gb/s. Skew is randomized on every link down event, speed change, and at initial startup.
max_tx_lane_skew_2_5g	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.
max_tx_lane_skew_5g	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.
max_tx_lane_skew_8g	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_tx_skp_interval_in_blocks	Maximum number of blocks before the upper phy must schedule the insertion of a SKP ordered set (8GT/s).
max_tx_skp_interval_in_symbol_times	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).
max_tx_skp_symbols_in_ordered_set	Maximum number of SKP symbols in an ordered set at 2.5GT/s and 5GT/s.
max_tx_skp_symbols_in_ordered_set_8g	Maximum number of SKP symbols in an ordered set at 8GT/s. Acceptable values: 1 – 5.
max_t_common_mode_ns	Max time before transmitting TS2 after coming out of L1 substates.
max_t_l1_2_ns	Maximum time to wait in L1.2 when CLKREQ must remain inactive.
max_t_power_off_ns	Maximum time to wait in L1.2 after sampling CLKREQ deasserted to refclk turned off.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
max_t_power_on_ns	Maximum time to wait in L1.2 after sampling CLKREQ asserted before driving interface.
min_configuration_lanenum_accept_timeout_ns	Time at which config.linkwidth.accept will be forced to make a decision on the linkwidth if errors are received.
min_eq_evaluation_delay[32]	Represents minimum time in NS to evaluate attached transmitter setting changes during equalization.
min_eq_preset_coeff_validation_delay[32]	Represents minimum amount of time VIP takes to either accept or reject a received preset/coefficient request.
min_l1_idle_time_ns	Minimum amount of time the LTSSM will remain in L1.Idle at speeds other than 2.5G even if directed to exit.
min_num_pclk_cycles_to_assert_invalid_request	This variable controls the minimum number of cycles that InvalidRequest will be asserted upon detecting invalid direction change feedback from an EQ evaluation request.
min_num_rx_eieos_before_fts	Minimum number of EIEOS to receive before sending FTS.
min_num_tx_eieos_before_fts	SVT_PCIE_MIN_NUM_TX_EIEOS_BEFORE_FTS_DEFAULT Minimum number of EIEOS ordered sets transmitted before transmitting FTS. (5GT/s only)
min_num_tx_eios_before_l1_gen1	Minimum num EIOS to send to accept L1 for Gen1.
min_num_tx_eios_before_l1_gen2	Minimum num EIOS to send to accept L1 for Gen2.
min_num_tx_eios_before_l1_gen3	Minimum num EIOS to send to accept L1 for Gen3
min_rx_eq_eval_delay[32]	Represents minimum amount of time in NS VIP in PHY PIPE mode takes to respond to RxEqEval request.
min_rx_skp_interval_in_blocks	Minimum number of blocks before the upper phy flags an error due to the lack of a SKP ordered set (8GT/s).
min_rx_skp_interval_in_symbol_times	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)
min_spipe_phystatus_delay	Minimum number of pipe clk cycles the SVC 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_phystatus_p2_exit_change_delay	Minimum number of pipe clk cycles that the SVC will keep phystatus asserted upon exiting from the P2 power state. Valid for SPIPE only.
min_spipe_preset_coefficients_delay	Represents a minimum delay time by PIPE PHY to respond to preset request by returning the coefficients request.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
min_tx_lane_skew_16g	Sets the minimum lane skew between transmit lanes in symbol times at 16Gb/s. Skew is randomized on every link down event, speed change, and at initial startup.
min_tx_lane_skew_2_5g	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.
min_tx_lane_skew_5g	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.
min_tx_lane_skew_8g	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.
min_tx_skp_interval_in_blocks	Minimum number of blocks before the upper phy must schedule the insertion of a SKP ordered set (8GT/s).
min_tx_skp_interval_in_symbol_times	Minimum 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_symbols_in_ordered_set	Minimum number of SKP symbols in an ordered set at 2.5GT/s and 5GT/s.
min_tx_skp_symbols_in_ordered_set_8g	Minimum number of SKP symbols in an ordered set at 8GT/s. Acceptable values: 1 – 5.
min_t_common_mode_ns	Minimum time before transmitting TS2 after coming out of L1 substates.
min_t_l1_2_ns	Minimum time to wait in L1.2 when CLKREQ must remain inactive,
min_t_power_off_ns	Minimum time to wait in L1.2 after sampling CLKREQ deasserted to refclk turned off.
min_t_power_on_ns	Minimum time to wait in L1.2 after sampling CLKREQ asserted before driving interface.
model_instance_scope	This is the full hierarchical path name to the instance of the SVC model which the driver model is instantiated in. The path name is concatenated with the name of this component passed to the constructor to generate the lookup string used to find the SV API instance.
num_additional_ts_eq_before_trans_eq0	Number of additional TS1s to transmit before transitioning out of equalization phase 0.
num_additional_ts_eq_before_trans_eq1	Number of additional TS1s to transmit before transitioning out of equalization phase 1.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
num_additional_ts_eq_before_trans_eq2	Number of additional TS1s to transmit before transitioning out of equalization phase 2.
num_additional_ts_eq_before_trans_eq3	Number of additional TS1s to transmit before transitioning out of equalization phase 3.
num_rx_fts_required_2_5g	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	The minimum number of FTS ordered sets required for a receiver to infer lock when exiting from electrical idle at 5GT/s.
num_rx_fts_required_8g	The minimum number of FTS ordered sets required for a receiver to infer lock when exiting from electrical idle at 8GT/s.
num_rx_recovery_idle_data	Number of Idle data symbols received in Recovery.Idle before transitioning to L0. This attribute can be used to delay transition from Recovery.Idle to L0.
num_tx_fts_required_2_5g	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.
num_tx_fts_required_5g	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.
num_tx_fts_required_8g	When SKIP_INITIAL_LINK_TRAINING is set, this is the default N_FTS value which the LTSSM will use at 8G when exiting from P1. All subsequent exits will use a value obtained from training sets used during link training.
num_tx_recovery_idle_data	Number of Idle data symbols transmitted in Recovery.Idle before transitioning to L0. This attribute can be used to delay transition from Recovery.Idle to L0.
num_tx_ts1_in_polling_active	Sets the number of training sets the LTSSM must transmit in Polling.Active before exiting this state.
pclk_rate[4]	Specifies the PCLK rate to be used at various data rates. pclk_rate[0] represents the rate used at 2.5 GT/s data rate. pclk_rate[1] represents the rate used at 5.0 GT/s data rate. pclk_rate[2] represents the rate used at 8.0 GT/s data rate. pclk_rate[3] represents the rate used at 16.0 GT/s data rate. Also refer to the description of 'pipe_width' which summarizes various PIPE configurations and 'pipe_width' and 'pclk_rate' settings to achieve these PIPE configurations.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
pclk_turned_off_in_p2	This attribute controls whether MAC should turn off PCLK in P2 while running in PCLK as PHY input mode or not.
pie8_enable_equalization_checks	When set to a 1, it enables the PIE-8 checks in the PIE8 PHY state machine. The checks performed are enabled individually as defined by pie8_enable_equalization_checks = SVT_PCIE_PIE8_ENABLE_EQUALIZATION_CHECKS_DEFAULT.
pie8_equalization_check_vector	Enable PIE-8 checks when pie8_enable_equalization_checks is set to 1.
pie8_max_mac_wait_delay_to_phy_dataen_timeout	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.
pie8_mode_en	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 = SVT_PCIE_IS_PIE8_MODE_MASTER_DEFAULT;).
pie8_phy_delay_to_tx_cmd_out_max	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 signal and proceeding toward completion.
pie8_phy_delay_to_tx_cmd_out_minpie8_phy_delay_to_tx_cmd_out_min	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 signal and proceeding toward completion.
tx_margin[4]	This configuration sets the tx_margin field.. When set VIP(MAC) will transmit this value of the tx_margin on the PIPE interface. • tx_margin[0] - tx_margin @ Gen1 speed. • tx_margin[1] - tx_margin @ Gen2 speed. • tx_margin[2] - tx_margin @ Gen3 speed. • tx_margin[3] - tx_margin @ Gen4 speed.
tx_ui_skew_2_5g	Maximum unit interval skew for 2.5G Valid range = [0:9]. The skew is interms of a reference lane which is selected by the VIP randomly and then VIP distributes UI skew on all other active lanes randomly(within [1:tx_ui_skew_2_5g] The unit for tx_ui_skew_2_5g is bit_clk period(400ps) for 2.5G

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
tx_ui_skew_5g	Maximum unit interval skew for 5G Valid range = [0:9]. The skew is interms of a reference lane which is selected by the VIP randomly and then VIP distributes UI skew on all other active lanes randomly(within [1:tx_ui_skew_5g] The unit for tx_ui_skew_5g is bit_clk period(200ps) for 5G
tx_ui_skew_8g	Maximum unit interval skew for 8G Valid range = [0:8]. The skew is interms of a reference lane which is selected by the VIP randomly and then VIP distributes UI skew on all other active lanes randomly(within [1:tx_ui_skew_8g] The unit for tx_ui_skew_8g is bit_clk period(125ps) for 8G
tx_ui_skew_16g	Maximum unit interval skew for 16G Valid range = [0:8]. The skew is interms of a reference lane which is selected by the VIP randomly and then VIP distributes UI skew on all other active lanes randomly(within [1:tx_ui_skew_8g] The unit for tx_ui_skew_8g is bit_clk period(62.5ps) for 16G
ssc_mode [4]	Spread spectrucm clocking mode for serial bit clock on the transmit path. It is only applicable for VIP running with serial interface. 0(default) - SSC_MODE_DISABLED => no ssc profile. 1 - SSC_MODE_DOWN => down spread spectrum. 2 - SSC_MODE_CENTER => center spread spectrum. ssc_mode[0] - ssc_mode for Gen1 speed. ssc_mode[1] - ssc_mode for Gen2 speed. ssc_mode[2] - ssc_mode for Gen3 speed. ssc_mode[3] - ssc_mode for Gen4 speed.
ssc_max_spread [4]	This configuration represents the ssc spread value in ppm. When ssc_mode = SSC_MODE_DISABLED, this configuration is not used. When ssc_mode = SSC_MODE_DOWN, this configuration represents DOWN spread value in ppm. When ssc_mode = SSC_MODE_CENTER, this configuration represents CENTER spread value in ppm. ssc_max_spread[0] - ssc_max_spread @ Gen1 speed. ssc_max_spread[1] - ssc_max_spread @ Gen2 speed. ssc_max_spread[2] - ssc_max_spread @ Gen3 speed. ssc_max_spread[3] - ssc_max_spread @ Gen4 speed. default is 16'd5000 for all speed if ssc_mode = SSC_MODE_CENTER | SSC_MODE_DOWN

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
ssc_modulation_rate	This configuration represents the modulration rate for ssc clock 2'b00 - 30Khzh. 2'b01 - 31Khzh. 2'b10 - 32Khzh. 2'b11 - 33Khzh. default value is 33Khz.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
pipe_width[4]	Specifies the PIPE width to be used at various data rates. pipe_width[0] represents the width used at 2.5 GT/s data rate. pipe_width[1] represents the width used at 5.0 GT/s data rate. pipe_width[2] represents the width used at 8.0 GT/s data rate. pipe_width[3] represents the width used at 16.0 GT/s data rate. Summary of various PIPE configurations and settings of 'pipe_width' and 'pclk_rate' to achieve these configurations. 1. Fixed width[8 bits] and variable PCLK PIPE configuration [PIPE_8] pipe_width[0] = PIPE_8_BITS; pclk_rate[0] = PCLK_250_MHZ; -- Achieve 2.5 GT/s pipe_width[1] = PIPE_8_BITS; pclk_rate[0] = PCLK_500_MHZ; -- Achieves 5.0 GT/s pipe_width[2] = PIPE_8_BITS; pclk_rate[0] = PCLK_1000_MHZ; -- Achieves 8.0 GT/s pipe_width[3] = PIPE_8_BITS; pclk_rate[0] = PCLK_2000_MHZ; -- Achieves 16.0 GT/s 2. Fixed width[16 bits] and variable PCLK PIPE configuration [PIPE_16] pipe_width[0] = PIPE_16_BITS; pclk_rate[0] = PCLK_125_MHZ; -- Achieve 2.5 GT/s pipe_width[1] = PIPE_16_BITS; pclk_rate[0] = PCLK_250_MHZ; -- Achieves 5.0 GT/s pipe_width[2] = PIPE_16_BITS; pclk_rate[0] = PCLK_500_MHZ; -- Achieves 8.0 GT/s pipe_width[3] = PIPE_16_BITS; pclk_rate[0] = PCLK_1000_MHZ; -- Achieves 16.0 GT/s 3. Fixed width[32 bits] and variable PCLK PIPE configuration [PIPE_32] pipe_width[0] = PIPE_32_BITS; pclk_rate[0] = PCLK_62_5_MHZ; -- Achieve 2.5 GT/s pipe_width[1] = PIPE_32_BITS; pclk_rate[0] = PCLK_125_MHZ; -- Achieves 5.0 GT/s pipe_width[2] = PIPE_32_BITS; pclk_rate[0] = PCLK_250_MHZ; -- Achieves 8.0 GT/s pipe_width[3] = PIPE_32_BITS; pclk_rate[0] = PCLK_500_MHZ; -- Achieves 16.0 GT/s 4. Fixed PCLK[250 MHz] and variable width PIPE configuration [PIPE_250_MHZ] pipe_width[0] = PIPE_8_BITS; pclk_rate[0] = PCLK_250_MHZ; -- Achieve 2.5 GT/s pipe_width[1] = PIPE_16_BITS; pclk_rate[0] = PCLK_250_MHZ; -- Achieves 5.0 GT/s pipe_width[2] = PIPE_32_BITS; pclk_rate[0] = PCLK_250_MHZ; -- Achieves 8.0 GT/s pipe_width[3] = PIPE_32_BITS; pclk_rate[0] = PCLK_500_MHZ; -- Achieves 16.0 GT/s
polling_active_timeout_ns	If at least one but not all unconfigured lanes detect a receiver, this configures the time between receiver detect sequences.
polling_configuration_timeout_ns	Timeout limit for the Polling.Configuration state.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
preset_to_coefficients_mapping_entry_valid	A bit mapped variable where bit N corresponds to entry N in preset_to_coefficients_mapping_table. When bit is set to 1'b1, it indicates that corresponding entry in preset_to_coefficients_mapping_table is valid for 8G.
preset_to_coefficients_mapping_entry_valid_16g	A bit mapped variable where bit N corresponds to entry N in preset_to_coefficients_mapping_table. When bit is set to 1'b1, it indicates that corresponding entry in preset_to_coefficients_mapping_table is valid for 16g.
preset_to_coefficients_mapping_table[16]	Preset to coefficients mapping table is used to map received preset requests to coefficients for use in local transmitter settings. For a passive VIP, the table should be programmed exactly like preset table of the link partner of monitored device. The table is indexed by a preset value. Mapping table stores coefficients values packed in the following format. { postcursor_coeff, cursor_coeff, precursor_coeff } Default value = { 6'h0c, 6'h24, 6'h00} == 18'h0c900. For 8G speed only.
preset_to_coefficients_mapping_table_16g[16]	Preset to coefficients mapping table is used to map received preset requests to coefficients for use in local transmitter settings. For a passive VIP, the table should be programmed exactly like preset table of the link partner of monitored device. The table is indexed by a preset value. Mapping table stores coefficients values packed in the following format. { postcursor_coeff, cursor_coeff, precursor_coeff } Default value = { 6'h0c, 6'h24, 6'h00} == 18'h0c900. For 16G speed only.
quiesce_guarantee	The variable represents the quiesce_guarantee variable in the VIP and this variable is reflected in Symbol 6 bit 6 of TS2 OS transmitted at 8G in Recovery.RcvrCfg state.
recovery_idle_timeout_ns	Timeout value in Recovery.Idle state.
recovery_rcvrcfg_timeout_ns	Timeout value in Recovery.RcvrCfg.
recovery_rcvrlock_timeout_ns	Timeout in Recovery.RcvrLock state.
recovery_speed_electrical_idle_time_ns	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.
recovery_speed_timeout_ns	Timeout value in Recovery.Speed state.
reject_preset_coefficient_request	A bit mapped attribute where value of bit N applies to lane N. Setting a bit to 1'b1 forces that particular lane to reject all new requests. This bit is applicable only in equalization phase 2 for downstream ports and equalization phase 3 for upstream ports.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
rx_dc_balance_mode_8g	This attribute controls the dc_balance field in 8g Rx TS Ordered Sets. If set to DC_BALANCE mode, the dc_balance field is set per spec when link is running at 8g. If set to TS_IDENTIFIER mode, dc_balance fied is set to TS1/2 identifier similar to spec for 2.5g/5g TS1/2 Ordered Sets.
set_ts_compliance_receive	When set, outgoing training sets will have the compliance receive bit set.
sris_mode_enabled	Used to enable/disable SRIS mode. Currently, this is used to control transmission of modified compliance pattern at 128/130b in SRIS and non-SRIS modes. This attribute is used by MPIPE VIP to drive SRISEnable signal on PIPE interface for spec version greater than or equal to 4.4. If this configuration is enabled, SRISEnable signal will be asserted by MPIPE VIP on all the active lanes. This attribute is also used by SPIPE VIP and passive component to expect SRISEnable signal assertion on all the active lanes from MAC on PIPE interface for spec version greater than or equal to 4.4. The default value of this attribute is set to 0. NOTE. Currently this is only used to control transmission/ reception of modified compliance pattern at 128/130b in SRIS and non SRIS modes. If set to "1" then VIP will transmit/receive SRIS mode modified compliance pattern else non SRIS one.
skip_initial_link_training	Allows link training to be completely skipped. The SVC will enter L0 once lane alignment has been achieved.
skip_polling_active	Allows Polling.Active to be skipped during link training. This helps speed up link training for cases not specifically testing the ltssm states.
spipe_receiver_present	For SPIPE only, this sets which lanes will return receiver present value whena receiver detect is requested by a DUT MAC. Each bit corresponds to a lane with bit zero corresponding to lane 0, bit 1 to lane 1 and so on. Not valid for serial, MPIPE and PMA models.
transmit_modified_compliance	When set, the SVC will transmit the modified compliance pattern upon entry into Polling.Compliance.
turn_off_unused_lanes_on_downconfigure	When the mpipe is doing a downconfigure, this controls whether or not the special 'turn off' signaling will be sent on unused lanes. This should only be done if the link doesn't support or won't be upconfigured later on.
tx_compliance_sos	Instructs the VIP to send 2 SKP ordered sets instead of 1 when in compliance mode.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
tx_dc_balance_mode_8g	This attribute controls the dc_balance field in 8g Tx TS Ordered Sets. If set to DC_BALANCE mode, the dc_balance field is set per spec when link is running at 8g. If set to TS_IDENTIFIER mode, dc_balance fied is set to TS1/2 identifier similar to spec for 2.5g/5g TS1/2 Ordered Sets.
tx_select_deemphasis	At 5G, the value of select_deemphasis.
tx_shutdown_latency	This attribute controls the latency to shutdown a lane.
tx_ts1_reset_eieos_interval_count_bit	When transmitting TS1s at 8GT/s speeds, and this bit is set to 1'b1, it forces VIP to set reset_eieos_interval_count bit in transmitted TS1s. Each lane can be set individually, so to for example to set this bit on lanes 0-3 only, set this to 32'h0000_000f.
upstream_lanes_recovery_eq_phase0_timeout_ns	Timeout in NS for upstream lanes in Recovery.Equalization.Phase0.
upstream_lanes_recovery_eq_phase1_timeout_ns	Timeout in NS for upstream lanes in Recovery.Equalization.Phase1.
upstream_lanes_recovery_eq_phase2_timeout_ns	Timeout in NS for upstream lanes in Recovery.Equalization.Phase2.
upstream_lanes_recovery_eq_phase3_timeout_ns	Timeout in NS for upstream lanes in Recovery.Equalization.Phase3.
upstream_preset_value[32]	Upstream Preset value. Applicable only for devices using downstream ports at 8G.
upstream_preset_value_16g[32]	Upstream Preset value. Applicable only for devices using downstream ports at 16G.
upstream_receiver_preset_hint[32]	The variable is applicable to a device with downstream ports only. Downstream ports advertise this value in the transmitted EQ TS2s before equalization phase 0 at 8G.
upstream_receiver_preset_hint_16g[32]	The variable is applicable to a device with downstream ports only. Downstream ports advertise this value in the transmitted EQ TS2s before equalization phase 0 at 16G.
user_tx_ts1_lane_mask	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.
user_tx_ts2_lane_mask	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 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
rx_standby_supported	Enables requiring `rx_standby` to be asserted when `rate`, `#pclk_rate` or `#pipe_width` are changed in PIPE 4.0 and above models. Allowed values: • 0 (Disabled): If VIP is MAC (MPIPE), then it will not assert `RxStandby`. If VIP is PHY (SPIPE), then it does not require `RxStandby` assertion and will not respond with `RxStandbyStatus`. • 1 (Enabled): VIP MAC (MPIPE) asserts `RxStandby`. VIP PHY (SPIPE) responds `RxStandby` with `RxStandbyStatus` assertion. The default value is set to 1.
fixed_ppm_due_to_tx_rx_xo	This configuration represents fixed ppm value in tx serial bit clk which comes due to different tx/rx crsystal oscillator. The unit of this attribute is *ppm* (parts per million). For example, -600 *ppm* would correspond to a bit clock period of .4002401441 ns (1000000.0 * .4)/(1000000.0 -600) at Gen1 speed.
allow_rate_power_simultaneous_change	In PIPE Interface mode, setting this variable to 1 allows Rate and PowerDown to change simultaneously—that is, if set to 1 the PHY (SPIPE) will not consider this as an illegal behavior and will not issue any warning. The default value of this variable is set to 1. Note: Only utilized by the active component and in PIPE models.
enable_upstream_tx_8g_eq_ts2_with_preset_in_rcvr_cfg	This configuration controls the enabling/disabling of sending 8G EQ TS2 by upstream port while in Recovery.Cfg. It also allows you to configure VIP to send user-defined transmitter preset values in Bit [6:3] of Sym7 of TS2 OS while in Recovery.Cfg. The description of 5 bits of this parameter is as follows: • Bit[4]: Controls the enabling/disabling of this feature.If set to 1, upstream port will send 8G EQ TS2 OS. If set to 0, upstream port will send normal TS2 identifier in SYM7. • Bit[3:0]: Contains the Transmitter Preset value to be transmitted by upstream port in Bit [6:3] of Sym7 of TS2 OS while in Recovery.Cfg. The default value of this variable is set to 1.
enable_powerdown_change_before_rate	In PIPE_INTERFACE mode, setting this variable to 1 will enable the MAC (MPIPE) to change PowerDown before Rate while entering L2.Idle or Detect LTSSM state. Note: Only utilized by the active component and in PIPE models. The default value of this variable is set to 0.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
enable_additional_power_state_transition_during_l1ss_exit	Specifies whether to enable or disable additional power state transition in L1.0 as described by attribute `pipe_powerdown_state_for_l1_0` during L1ss exit. This is applicable to MPIPE mode for PIPE specification revisions 4.0 or higher. The default value of this variable is set to 0.
use_dynamic_local_fs_lf_values	Specifies the configuration to drive FS/LF fields in TS1 ordered sets by VIP for MPIPE. • When set to 1 and dynamic preset coefficient updates are enabled (`enable_get_local_preset_coefficients` set to 1), the VIP will use `LocalFS` and `LocalLF` signals driven by PHY to drive FS and LF fields in TS1 ordered sets for MPIPE. • When set to 1 and dynamic preset coefficient updates are disabled (`enable_get_local_preset_coefficients` set to 0), the VIP will use `fs_value`/`lf_value` when rate is 8 GT/s or `fs_value_16g`/`lf_value_16g` when rate is 16 GT/s to drive FS and LF fields in TS1 ordered sets for MPIPE. • When set to 0, the VIP will use `fs_value`/`lf_value` when rate is 8 GT/s or `fs_value_16g`/`lf_value_16g` when rate is 16 GT/s to drive FS and LF fields in TS1 ordered sets for MPIPE. The default value of this attribute is set to 1 and would be applicable only when dynamic preset coefficient updates are enabled.
use_dynamic_local_preset_coefficients	Specifies the configuration to drive coefficients in TS1 ordered sets by VIP for MPIPE for a preset request from the link partner. • When set to 1, the VIP will use `LocalTxPresetCoefficients` signal driven by PHY to drive coefficients in TS1 ordered sets for MPIPE to a preset request from the link partner. • When set to 0, the VIP will use preset mapping tables as indicated by `preset_to_coefficients_mapping_table` when rate is 8 GT/s or `preset_to_coefficients_mapping_table_16g` when rate is 16 GT/s to drive coefficients in TS1 ordered sets for MPIPE to a preset request from the link partner. This attribute is applicable only when dynamic preset coefficient updates are enabled (`enable_get_local_preset_coefficients` set to 1). Default value of this attribute is set to 0.
compliance_speed_change_timeout_ns	Specifies the maximum amount of time in ns the VIP will remain in electrical idle when changing speed while in Polling Compliance state prior to sending compliance pattern. If randomized, the variable will resolve to a value within the range specified by the constraint `#valid_ranges`.
rx_modified_compliance_pattern_lock_bit	Specifies the value of the pattern lock bit in modified compliance pattern on each lane. The default value is set to 0.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
re_request_setting_on_ts1_after_invalid_request	Supports re-request of last accepted preset/coefficient setting after `InvalidRequest`. Default value is set to 0.
async_power_change_ack_timeout_ns	Specifies the time in ns, SPIPE VIP after asserting Phystatus will wait for AsyncPowerChangeAck to go high required for PowerDown change process from a power state that does not provide PCLK to another power state that does not provide PCLK. Applicable when VIP is a PIPE slave. The attribute controls the timeout value for phy_async_power_change_ack_timeout protocol check. This attribute needs to be programmed to match DUT MAC behavior so that VIP can check DUT's timing for AsyncPowerChangeAck signal assertion if checking actual timing is desired. If randomized, the variable will resolve to a value within the range specified by the constraint #valid_ranges Note: Only utilized by the active component.
pclk_change_ok_timeout_ns	Specifies the time in ns, MPIPE VIP after doing PCLK Rate change will wait for PclkChangeOk to go high required for rate change process. Applicable when VIP is a PIPE master and for PCLK as PHY input mode. The attribute controls the timeout value for phy_pclk_change_ok_timeout protocol check. This attribute must be programmed to match DUT PHY behavior so that VIP can check DUT's timing for PclkChangeOk signal assertion if checking actual timing is desired. If randomized, the variable will resolve to a value within the range specified by the constraint #valid_ranges Note: Only utilized by the active component.
pclk_change_ack_timeout_ns	Specifies the time in ns, SPIPE VIP after doing PclkChangeOk assertion will wait for PclkChangeAck to go high required for rate change process. Applicable when VIP is a PIPE slave and for PCLK as PHY input mode. The attribute controls the timeout value for phy_pclk_change_ack_timeout protocol check. This attribute must be programmed to match DUT MAC behavior so that VIP can check DUT's timing for PclkChangeAck signal assertion if checking actual timing is desired. If randomized, the variable will resolve to a value within the range specified by the constraint #valid_ranges Note: Only utilized by the active component.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
phystatus_timeout_ns	Specifies the time in ns, MPIPE VIP will wait for PHYSTATUS to go high to acknowledge a receiver detect, a rate change, or PowerDown change. Applicable when VIP is PIPE master. The attribute controls the timeout value for phy_phystatus_timeout protocol check. This attribute must be programmed to match DUT PHY behavior so that VIP can check DUT's timing for PhyStatus signal assertion if actual timing check is required. If randomized, the variable will resolve to a value within the range specified by the constraint #valid_ranges. Note: Only utilized by the active component.
rxstandbystatus_timeout_ns	Specifies time in ns, MPIPE VIP after asserting RxStandby will wait for RxStandbyStatus to go high. Applicable when VIP is PIPE master. The attribute controls the timeout value for phy_rxstandbystatus_timeout protocol check. This attribute must be programmed to match DUT PHY behavior so that VIP can check DUT's timing for RxStandbyStatus signal assertion if actual timing check is required. If randomized, the variable will resolve to a value within the range specified by the constraint #valid_ranges Note: Only utilized by the active component.
enable_direct_speed_up_from_2_5g_to_16g	Enables the VIP to directly go from 2.5G to 16G when the link initially comes up. When the link directly goes from 2.5G to 16G, equalization will be performed by default at 16G. If you do not want the equalization to be performed at 16G, then you must set `highest_enabled_equalization_phase` to 0 along with this configuration. This configuration violates the specification and is used solely to speed up simulation time by decreasing the time it takes to train the link and should only be used if the DUT supports this behavior. Significance: • 1: VIP directly goes from 2.5G to 16G bypassing 8G when the link initially comes up. • 0: VIP behaves in default manner when the link initially comes up i.e link goes from 2.5G to 8G to 16G.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
rx_standby_controls	Specifies the following controls for `RxStandby` assertion in various scenarios only when configuration attribute `rx_standby_supported` is set to 1: • Bit 0: Enables `RxStandby` assertion during rate change. • Bit 1: Enables `RxStandby` assertion in `Rx.L0s.Idle` LTSSM state. • Bit 2: Enables `RxStandby` assertion in P1 or lower power states (for `PowerDown` change in `L1` and `L2` LTSSM states). • Bit 3: Enables `RxStandby` assertion for `PowerDown` change in `Detect.Quiet` LTSSM state. • Bit 4: Enables `RxStandby` assertion for `PowerDown` change in `Disabled` LTSSM state. • Bit 5: Enables `RxStandby` assertion on unused/inactive lanes or turned off lanes. • Bit 6: Enables `RxStandby` assertion when LTSSM moves from L0 state to Recovery state by Inferring Electrical Idle. • Bit 7: Enables `RxStandby` assertion in `Recovery.Speed` LTSSM state even when there is no rate change. Default value of this attribute is set to 8'b00111111.
include_delay_symbols_in_compliance_pattern_at_x1	The PCIE SVT field which controls the inclusion of delay symbols in compliance and modified compliance patterns at x1 link width. • 1: VIP will include delay symbols in compliance and modified compliance patterns at x1 link width. • 0: VIP will not include delay symbols in compliance and modified compliance patterns at x1 link width.
current_ltssm_state_transition_reason	The current state transition reason of the LTSSM. The active component updates this variable as soon as it detects LTSSM state change.
rx_transmitter_preset_compliance_pattern	Value of the received transmitter preset value in symbol7 of compliance pattern at 128/130b.
compliance_preset_deemphasis	The PCIE SVT field which defines the Compliance Preset/De-emphasis value stored in Link Control 2 - bits [15:12] field. Note: Only utilized by the active component at 128/130b for setting preset value in compliance patter. Also, used at 5G to control tx deemphasis. The default value is set to 0.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
assert_tx_commonmode_disable	Enables `TxCommonModeDisable` assertion and de-assertion during entry and exit from L1 Low Power substates as per PIPE 4.4. • 1 – Assertion of `TxCommonModeDisable` is enabled during L1SS. • 0 – Assertion of `TxCommonModeDisable` is disabled during L1SS. Note: Utilized by active component in PIPE mode. Applicable when `pipe_spec_ver` is 4.4 or higher and `enable_l1ss_pipe_handshake` is set to 0.
assert_rx_eidetect_disable	Enables `RxEIDetectDisable` assertion and de-assertion during entry and exit from L1 Low Power substates as per PIPE 4.4. • 1 – Assertion of `RxEIDetectDisable` is enabled during L1SS. • 0 – Assertion of `RxEIDetectDisable` is disabled during L1SS. Note: Utilized by active component in PIPE mode. Applicable when `pipe_spec_ver` is 4.4 or higher and `enable_l1ss_pipe_handshake` is set to 0.
beacon_signal_pulse_width	Specifies the pulse width of the beacon signal that VIP will transmit during L2.Transmit.Wake state.If randomized, the variable will resolve to a value within the range specified by the constraint #valid_ranges. The default value is set to 2. Note: Only utilized by active component.
update_status_attributes_based_on_physical_lanes	Enables updates to specific per lane PIPE status attributes and specific per lane EQ status attributes based on physical lane numbers. • 0 (disabled) – VIP updates PIPE sideband signal status variables and EQ sideband signal status variables based on logical lane indexing. • 1 (enabled) – VIP updates PIPE sideband signal status variables and EQ sideband signal status variables based on physical lane indexing. Note: Only utilized by active component. The default value of this attribute is set to 0.
spipe_assert_phystatus_as_compliance_load_board	Controls whether SPIPE VIP will assert PhyStatus or not when modeled as a compliance load board. • When set to 1, SPIPE VIP will assert PhyStatus on all lanes up to its configured link width in response to rate change when acting as a compliance load board. • When set to 0, SPIPE VIP will not assert PhyStatus in response to rate change when acting as a compliance load board. The default value of this attribute is set to 0.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
l1ss_rxelecidle_disable_assertion_delay_ns	The attribute controls the delay [in ns] of `rxelecidle_disable`/`rx_eidetect_disable` assertion with respect to `clkreq_n` de-assertion. Note: Utilized by the active component in MPIPE mode when `enable_l1ss_pipe_handshake` is set to 1 or `pipe_spec_ver` is 4.4 and higher and `enable_l1ss_pipe_handshake` is set to 0. Utilized by the passive component when `enable_l1ss_pipe_handshake` is set to 1. If randomized, the variable will resolve to a value within the range specified by the constraints `#reasonable_l1ss_rxelecidle_disable_assertion_delay_ns` and `#valid_ranges`. The default value of this attribute is set to 8.
l1ss_txcommonmode_disable_assertion_delay_ns	The attribute controls the delay [in ns] of `txcommonmode_disable`/`tx_commonmode_disable` assertion with respect to `clkreq_n` de-assertion. Note: Utilized by the active component in MPIPE mode when `enable_l1ss_pipe_handshake` is set to 1 or `pipe_spec_ver` is 4.4 and higher and `enable_l1ss_pipe_handshake` is set to 0. Utilized by the passive component when `enable_l1ss_pipe_handshake` is set to 1. If randomized, the variable will resolve to a value within the range specified by the constraints `#reasonable_l1ss_txcommonmode_disable_assertion_delay_ns` and `#valid_ranges`. The default value of this attribute is set to 8.
l1ss_rxelecidle_disable_deassertion_delay_ns	The attribute controls the delay [in ns] of `rxelecidle_disable` (when `enable_l1ss_pipe_handshake` is set to 1) de-assertion with respect to `pclkreq_n` assertion. Same attribute controls the delay [in ns] of `rx_eidetect_disable` (when `pipe_spec_ver` is 4.4 and higher and `enable_l1ss_pipe_handshake` is set to 0) de-assertion with respect to L1_1/L1_2_IDLE substate exit. Note: Utilized by the active component in MPIPE mode when `enable_l1ss_pipe_handshake` is set to 1 or `pipe_spec_ver` is 4.4 and higher and `enable_l1ss_pipe_handshake` is set to 0. Utilized by the passive component when `enable_l1ss_pipe_handshake` is set to 1. If randomized, the variable will resolve to a value within the range specified by the constraints `#reasonable_l1ss_rxelecidle_disable_deassertion_delay_ns` and `#valid_ranges`. The default value of this attribute is set to 8.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
l1ss_txcommonmode_disable_deassertion_delay_ns	The attribute controls the delay [in ns] of txcommonmode_disable (when enable_l1ss_pipe_handshake is set to 1) de-assertion with respect to pclkreq_n assertion. Same attribute controls the delay [in ns] of tx_commonmode_disable (when pipe_spec_ver is 4.4 and higher and enable_l1ss_pipe_handshake is set to 0) de-assertion with respect to L1_1/L1_2_IDLE substate exit. Note: Utilized by the active component in MPIPE mode when enable_l1ss_pipe_handshake is set to 1 or pipe_spec_ver is 4.4 and higher and enable_l1ss_pipe_handshake is set to 0. Utilized by the passive component when enable_l1ss_pipe_handshake is set to 1. If randomized, the variable will resolve to a value within the range specified by the constraints #reasonable_l1ss_txcommonmode_disable_deassertion_delay_ns and #valid_ranges. The default value of this attribute is set to 8.
pclk_turn_off_delay	Specifies the number of PCLK cycles the VIP will wait before turning off PCLK after phystatus assertion on all lanes during transition from P0/P0s/P1/Phy power state with PCLK to P2/Phy power state without PCLK in PCLK as PHY output mode. Valid only for SPIPE models. Note: Only utilized by the active component. Minimum value should be two PCLK cycles. If randomized, the variable will resolve to a value within the range specified by the constraint #valid_ranges.
pclk_toggling_after_resetn_timeout	Specifies the delay in ns after which MPIPE VIP will flag a debug message if PCLK does not toggle upon Reset# assertion. Note: The attribute is only applicable for MPIPE VIP in PCLK as PHY output mode. If randomized, the variable will resolve to a value within the range specified by the constraint #valid_ranges. The default value is set to 1000.
set_reject_coeff_bit_for_reserved_tx_preset_eq_ph0	Enables the Upstream port in Equalization.Phase0 to set Reject Coefficient bit[6] in Symbol 9 of TS1 OS in response to a reserved or unsupported Transmitter Preset it received during Recovery.Config in the EQ/8G EQ TS2 Ordered Sets. • When set to 1, the VIP will assert Reject Coefficient bit. • When set to 0, the VIP will not assert Reject Coefficient bit. The default value of this configuration is 1. Note: Only utilized by the active component.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
enable_ltssm_scrambling_at_128b_130b	Specifies if LTSSM scrambling is enabled at 128b/130b. When VIP receives TS1 OS with disable scrambling bit set in `Configuration.Complete` at 8b/10b, VIP disables its scrambling and it starts transmitting unscrambled data and expects receiving unscrambled data from the other device at 8b/10b encoding. But when the speed changes to 8G or greater and the other device starts sending scrambled data again then this configuration must be set to 1 before speed change to enable the VIP's scrambling and allow it to transmit and receive scrambled data again. Significance: • When set to 1'b0, LTSSM scrambling is disabled at 128b/130b for the above scenario. • When set to 1'b1, LTSSM scrambling will be enabled at 128b/130b for the above scenario. Default value is 0. If randomized, the variable will resolve to 0 or 1 since the variable is unconstrained. Note: This is applicable only for active VIP.
enable_eq_ts1_in_loopback_state	When 8 GT/s or higher data rates are supported this parameter enables the VIP (Loopback Master) to transmit EQ TS1 instead of standard TS1 in Loopback ltssm state. Significance: • When set to 0 VIP (loopback master) will transmit standard TS1 in loopback LTSSM state. • When set to 1 VIP (loopback master) will transmit EQ TS1 in loopback ltssm state. Default value is 0. If randomized, the variable will resolve to 0 or 1 since the variable is unconstrained. Note: This is applicable only for Active VIP when VIP is acting as Loopback Master.
enable_use_preset_in_loopback_state	Enables the VIP (Loopback Master) to transmit use preset bit as 1 in the transmitted TS1 OS at 128/130b in Loopback.Entry. Significance: • When set to 1, VIP will communicate Transmitter Preset in the 6th symbol of TS1 OS in Loopback.Entry. • When set to 0, VIP will communicate Coefficient Settings in the TS1 OS in Loopback.Entry. Default value is 0. If randomized, the variable will resolve to 0 or 1 since the variable is unconstrained. Note: This is applicable only for Active VIP when VIP is acting as Loopback Master.
enable_p1_powerstate_for_l1_substates	Specifies whether to enable P1 `PowerDown` state support for L1 substates or not. Default value of this attribute is set to 0.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
num_rx_fts_required_32g	Specifies the minimum number of FTS ordered sets required for a receiver to infer lock when exiting from electrical idle at 32GT/s.If randomized, the variable will resolve to a value within the range specified by the constraint `#valid_ranges`. The default value is set to 255.
num_tx_fts_required_32g	Specifies the default `N_FTS` value which the LTSSM will use at 32G when exiting from L0s, when `#skip_initial_link_training` is set. All subsequent exits will use a value obtained from training sets used during link training. If randomized, the variable will resolve to a value within the range specified by the constraint `#valid_ranges`. The default value is set to 255.
num_rx_fts_required_16g	Specifies the minimum number of FTS ordered sets required for a receiver to infer lock when exiting from electrical idle at 16GT/s. If randomized, the variable will resolve to a value within the range specified by the constraint `#valid_ranges`.
num_tx_fts_required_16g	Specifies the default `N_FTS` value which the LTSSM will use at 16G when exiting from L0s, when `#skip_initial_link_training` is set. All subsequent exits will use a value obtained from training sets used during link training. If randomized, the variable will resolve to a value within the range specified by the constraint `#valid_ranges`.
pclkackn_timeout_ns	• Time in ns, MPIPE VIP after de-asserting `pclkreq_n` will wait for `pclkack_n` to get deasserted. • Time in ns, MPIPE VIP after asserting `pclkreq_n` and de-asserting `rxelecidle_disable`/ `txcommonmode_disable` will wait for `pclkack_n` to get asserted. The attribute is applicable when VIP is a PIPE Master, PL configuration attribute `pipe_spec_ver` is set to a value greater than or equal to 4.0 and PL configuration attribute `enable_l1ss_pipe_handshake` is set to 1. Default value of this attribute is set to 1000.
enable_p1_powerstate_before_l1_0_powerstate_during_l1ss_entry	Specifies whether to enable or disable P1 power state transition prior to entering L1ss. Significance as follows: • When set to 0, LTSSM carries out default `PowerDown` transition as P0 -> L1.0 power state (as defined by PL configuration class attribute `pipe_powerdown_state_for_l1_0`). • When set to 1, LTSSM carries out `PowerDown` transitions as P0 -> P1 -> L1.0 power state (as defined by PL configuration class attribute `pipe_powerdown_state_for_l1_0`). This attribute is only utilized by the active component and only in PIPE models with a revision 4.0 or greater. Default value of this attribute is set to 0.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
enable_p1_powerstate_after_l1_0_powerstate_during_l1ss_exit	Specifies whether to enable or disable P1 power state transition after exiting L1ss. Significance as follows: • When set to 0 and PL configuration class attribute `enable_additional_power_state_transition_during_l1ss_exit` is clear, LTSSM carries out default `PowerDown` transition as L1.1 power state (as defined by PL configuration class attribute `pipe_powerdown_state_for_l1_1`) -> P0 OR L1.2.Exit power state (as defined by PL configuration class attribute `pipe_powerdown_state_for_l1_2_exit`) -> P0. • When set to 0 and PL configuration class attribute `enable_additional_power_state_transition_during_l1ss_exit` is set, LTSSM carries out default `PowerDown` transition as L1.0 power state (as defined by PL configuration class attribute `pipe_powerdown_state_for_l1_0`) -> P0. • When set to 1 and PL configuration class attribute `enable_additional_power_state_transition_during_l1ss_exit` is clear, LTSSM carries out `PowerDown` transitions as L1.1 power state (as defined by PL configuration class attribute `pipe_powerdown_state_for_l1_1`) -> P1 -> P0 OR L1.2.Exit power state (as defined by PL configuration class attribute `pipe_powerdown_state_for_l1_2_exit`) -> P1 -> P0. • When set to 1 and PL configuration class attribute `enable_additional_power_state_transition_during_l1ss_exit` is set, LTSSM carries out `PowerDown` transitions as L1.0 power state (as defined by PL configuration class attribute `pipe_powerdown_state_for_l1_0`) -> P1 -> P0. This attribute is only utilized by the active component and only in PIPE models with a revision 4.0 or greater. Default value of this attribute is set to 0.
async_power_change_ack_deassertion_delay	Specifies the delay in ns after which MPIPE VIP will de-assert `AsyncPowerChangeAck` signal after sampling phystatus de-assertion on all lanes for power state transitions without PCLK. If randomized, the variable will resolve to a value within the range specified by the constraint `#reasonable_async_power_change_ack_delay`.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
disable_invalid_request_on_reserved_dir	Specifies if MPIPE will consider 'Reserved' direction change feedback corresponding to any of the coefficients as out of range and assert `InvalidRequest` in response to that. Significance: • When set to 1, the VIP will not consider 'Reserved' direction change feedback as out of range value and perform evaluation considering it as 'No change' for the corresponding coefficient. • When set to 0, the VIP will assert `InvalidRequest` in response to 'Reserved' direction change feedback received on `LinkEvaluationFeedbackDirectionChange` for any of the applicable coefficients. Note: Only utilized by the MPIPE active component.
ignore_cursor_coefficient_dir_in_16g	Specifies if MPIPE will consider cursor coefficient direction change feedback and evaluate the coefficient settings in 16G rate. Significance: • When set to 1, the VIP will ignore direction change feedback for cursor coefficient and perform evaluation after recalculating the coefficients based on direction change feedback for other coefficients in 16G rate. • When set to 0, the VIP will consider the direction change feedback value for cursor coefficient in 16G rate as per PIPE specification version 4.4 or above and perform evaluation. Note: Utilized by active as well as passive component.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
allow_mac_rate_pclk_rate_width_signaling_changes_when_prev_signal_changes_not_acknowledged_by_phy	Specifies if the VIP would cater the scenario where MAC changes Rate/Width/PCLK Rate one at a time or one after the other when VIP has not provided `PhyStatus` for earlier change that MAC already did and still VIP would be able to generate PCLK at the correct frequency depending on the Rate/Width/ PCLK Rate changes at mutually exclusive time stamps. Significance: • When set to 0, VIP will not consider any further signaling rate changes (Rate/Width/PCLK Rate change) for PCLK frequency calculation until it has indicated that the earlier signaling rate change has completed via `PhyStatus` assertion. • When set to 1, VIP will consider any further signaling rate changes (Rate/Width/PCLK Rate change) for PCLK frequency calculation even if it has not indicated that the earlier signaling rate change has completed via `PhyStatus` assertion and provided VIP has not acknowledged the previous signaling rate change request yet with `PhyStatus` on any of the active lanes. Setting a value of 1 would violate the PIPE specification which states that any combination of at least two of the rate and width and PCLK rate, can be changed simultaneously. The MAC is not allowed to change only one of the three. When the MAC changes the Rate signal, and/or the Width signal, and/or the PCLK rate signal in PCLK as PHY Output mode, the PHY performs the rate change and/or the width change and/or the PCLK rate change and signals its completion with a single cycle assertion of `PhyStatus`. The MAC must not perform any operational sequences, power state transitions, de-assert `TxElecIdle` or `RxStandby`, or further signaling rate changes until the PHY has indicated that the signaling rate change has completed. Only utilized by the Active component when VIP is a PIPE Slave. The default value of this attribute is 0.
ignore_dir_on_preset_request	Specifies if MPIPE will consider direction change feedback during preset requests. Significance: • When set to 1, the VIP will not consider direction change feedback values to evaluate the corresponding coefficients during a preset request. Values will only be considered when coefficients requests are done. • When set to 0, the VIP will consider direction change feedback values to evaluate the corresponding coefficients during a preset request. Note: Only utilized by the MPIPE active component.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

<table>
<tr><th>Member</th><th>Description</th></tr>
<tr><td>repeat_count_for_rxeqeval_after_first_invalid_phy_response</td><td>Specifies the number of times MPIPE VIP will repeat RxEqEval assertion after first RxEqEval assertion has received an invalid PHY response.
Significance:
• When set to -1, MPIPE VIP will do RxEqEval assertion continuously until a valid response is received or equalization state times out.
• When set to 0, MPIPE VIP does not do a second RxEqEval request after first RxEqEval assertion has received an invalid PHY response. MPIPE VIP will not assert InvalidRequest signal on reception of an invalid PHY response in this case.
• When set to N (where N >= 1), MPIPE VIP carries out up to N additional RxEqEval requests after first RxEqEval assertion has received invalid PHY response. If any RxEqEval assertion has received a valid PHY response before counter of N requests gets over, VIP chooses to move on.
This attribute is only utilized by the active component when in MPIPE mode for PIPE specifications 4.0 or higher. Default value of this attribute is set to 0.</td></tr>
<tr><td>enable_optional_pclkchangeok_pclkchangeack_handshake</td><td>Specifies whether to enable optional PclkChangeOk/ PclkChangeAck handshake for Rate change.
Significance:
• When set to 1, handshake is enabled for both PCLK Rate change and Rate change.
• When set to 0, handshake is disabled for Rate change but is enabled for PCLK Rate change.
This attribute is only utilized by the active component and only in PIPE models with a revision 5.1 or greater. Default value of this attribute is set to 0.</td></tr>
<tr><td>enable_optional_ref_clk_required_handshake</td><td>Specifies whether to enable optional RefClkRequired# handshake so that the reference clock can be safely removed in low power states. This attribute is only utilized by the active component and only in PIPE models with a revision 5.1 or greater. Default value of this attribute is set to 0.</td></tr>
<tr><td>phy_time_to_provide_rx_link_eval_status_mbi_fld_ns</td><td>Specifies the maximum amount of time, in ns, MAC would wait for PHY to respond with either or both Direction Change or Figure Merit feedback using MAC Rx Link Evaluation Status0/1 register in response to RxEqEval request of MAC.
Applicable only to MBI transactions for PIPE v5.1 or higher.
Note: Utilized by both Active and Passive components.</td></tr>
</table>

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
num_of_pclk_cycles_to_wait_for_stable_pclk	Specifies the number of PCLK cycles VIP waits after Rate change to consider PCLK is stable and PCLK frequency is matching with the expected value. When PCLK is stable, MBI transactions are initiated. Significance: When `PclkChangeOk` and `PclkChangeAck` handshake is expected, the configuration is effective after `PclkChangeAck` is asserted. Otherwise, effective as soon as Rate changes. Note: Currently, utilized by Active component when `pipe_spec_ver` is set to a value greater than or equal to 5.1
dut_configured_physical_lanes_vector	A bit-mapped vector where bit N corresponds to physical lanes of DUT which are connected to VIP. This is required for SPIPE VIP to expect `WriteAck` in response to `LocalFS` and `LocalLF` initiated after `Reset#` de-assertion on those lanes which are connected to MAC DUT. After this SPIPE will de-assert `PhyStatus` and eventually expect `TxDetectRx` assertion on the same lanes if connected per-lane basis (`svt_pcie_pl_configuration::enable_shared_sideband_signals` is set to 0). Note: This should be constant in a simulation. If DUT physical lanes are changed using `Hot.Unplug` during mid simulation, then you must reconfigure this as well. Value should be less than or equal to `link_width` of VIP. If programmed with a greater value, then only the bits until `link_width` are considered and the rest are ignored. Note: Only applicable for SPIPE VIP when attribute `pipe_spec_ver` is set to a value greater than or equal to 5.1.
tx_eios_check_enable	Specifies control to verify that transmitters must transmit all symbols of an EIOS. This attribute is only utilized by the active component. Default value of this attribute is set to 0.
data_signals_common_mode_value	Specifies whether `datap` and `datan` signals will be pulled up or pulled down or tristated while in electrical idle when common mode voltage requirements are to be met. This attribute is applicable only in SERDES mode. • When in Passive mode; this attribute sets the expectation of a protocol check which checks DUT's behavior. • When in Active mode; this attribute controls the value driven on `txdatap` and `txdatan` signals while in electrical idle. - When "0", 1 is driven in electrical idle on `txdata` signals. - When "1", 0 is driven in electrical idle on `txdata` signals. - When "2", z is driven in electrical idle on `txdata` signals. - When "3", the value driven in electrical idle on `txdata` signals is dictated by legacy mechanism—that is, parameter settings from `pciesvc_serdes` module.

Table 16-7 PHY Layer Functions and Configuration Members (Continued)

Member	Description
pipe_rx_width	`pipe_rx_width` specifies the PIPE RxWidth to be used at various data rates for PIPE specification version 5.1 or greater: • `pipe_rx_width[0]` represents the RxWidth used at 2.5 GT/s data rate. • `pipe_rx_width[1]` represents the RxWidth used at 5.0 GT/s data rate. • `pipe_rx_width[2]` represents the RxWidth used at 8.0 GT/s data rate. • `pipe_rx_width[3]` represents the RxWidth used at 16.0 GT/s data rate. • `pipe_rx_width[4]` represents the RxWidth used at 32.0 GT/s data rate.
evaluation_feedback_response	Specifies if SPIPE will respond with FOM (`FigureMerit`) or DIR (`DirectionChange`) or both feedback response to link equalization evaluation (`RxEqEval`) request from the MAC. Significance: • When set to 00, the VIP will respond with both FOM and DIR feedback response (`WriteUncommitted` followed `WriteCommitted`). • When set to 01, the VIP will respond with only FOM feedback response (only one `WriteCommitted`). • When set to 10, the VIP will respond with only DIR feedback response (only one `WriteCommitted`). Note: Only utilized by the SPIPE active component when attribute `pipe_spec_ver` is set to a value greater than or equal to 5.1.
enable_use_preset_bit_in_ts1_with_accepted_rejected_tx_setting	Enables the VIP to set `use_preset` bit in outgoing TS1s in `Eq.Ph2/Eq.Ph3` while responding with acceptance/rejection to received transmitter settings. • When set to 1, VIP will set `use_preset` bit to 1 in outgoing TS1s if it has received a preset request. • When set to 0, VIP will set `use_preset` bit to 0 in outgoing TS1s if it has received a preset request.

16.6 External Tx Bit Clk Use Model

This is a special use model which can be used in common ref clk mode. In this mode, the VIP is capable of transmitting serial data with respect.to serial bit clocks provided from the Test Bench. To enable this model VIP has to be configured in following manner from the Test Bench.

```
defparam <vip_top_level_inst_path>.port0.USE_EXTERNAL_BIT_CLK = 1;
assign <vip_top_level_inst_path>.ext_bit_clk_gen1 =
endpoint0.port0.SVC_CLKGEN4_INST.clkgen_txclk0.bit_clk;clk;
assign <vip_top_level_inst_path>.ext_bit_clk_gen2 =
endpoint0.port0.SVC_CLKGEN4_INST.clkgen_txclk0.bit_clk;clk;
assign <vip_top_level_inst_path>.ext_bit_clk_gen3 =
endpoint0.port0.SVC_CLKGEN4_INST.clkgen_txclk0.bit_clk;clk;
assign <vip_top_level_inst_path>.ext_bit_clk_gen4 =
endpoint0.port0.SVC_CLKGEN4_INST.clkgen_txclk0.bit_clk;clk;
```

Where root0 is the top level instantiation absolute path in the Test Bench.

If a device does not support Gen2/Gen3/Gen4, then leave following wires open:

- ext_bit_clk_gen2
- ext_bit_clk_gen3
- ext_bit_clk_gen4

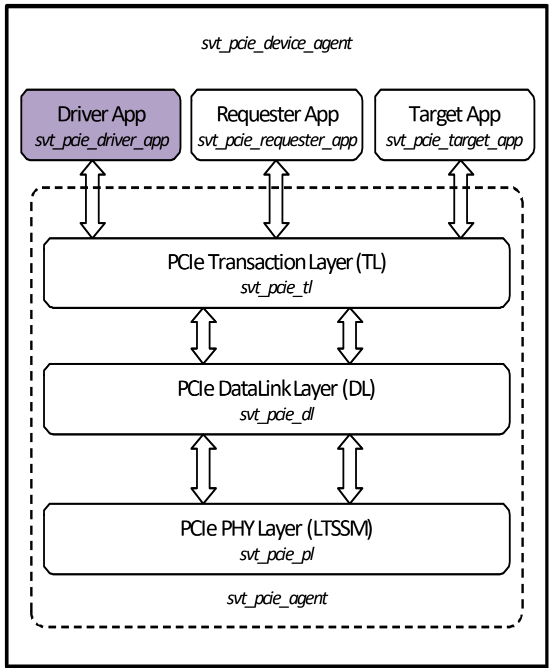
17 Using the Driver Application

This chapter discusses the following topics:

17.1 Introduction

The driver application is an abstraction layer that allows a test to generate PCIe transactions with an abstract description of a PCIe transaction request. The driver application is modeled using class `svt_pcie_driver_app` and is instanced inside the `svt_pcie_device_agent` class as `driver[0]`. The `svt_pcie_driver_app` class inherits the properties of the UVM driver and so it can initiate transaction requests.

Figure 17-1 Block Diagram – PCIe UVM VIP



Apart from the obvious convenience of abstraction provided for test writing, the driver application also provides the following features:

- Data-integrity checks: The driver application has self-checks to ensure that the data returned for a memory read is the expected value.
- Compliance checks: The driver application has checks to ensure the completions returned by the link partner for requests generated by the driver conforms to the specification. It tracks requests from start to completion and ensures all requests receive a successful completion.
- Automatic error recovery: The driver application has a pre-defined set of errors that can be injected by a test. The driver application recovers from those errors by suppressing the related errors and also checks for the correctness in the behavior of the DUT in response to the error.

The driver application constructs Transaction Layer Packets (TLP) from the abstract description provided to it by the test and transmits these TLPs to the Transaction Layer (TL). The TL then processes these TLPs as per the specification before handing them down to the layer below it. It is also possible to bypass the driver application and directly queue TLPs on the TL. To do this, you require a more descriptive TLP class, `svt_pcie_tlp`. With this approach, a test will end up being much longer and it would need to perform the task of tracking and checking the related completions. And so it is highly recommended to use the driver application as the testing interface for generating TLPs.

17.2 Driver Application Configuration

The Driver Application is configured using an object of class type `svt_pcie_driver_app_configuration`.

```
svt_pcie_device_configuration

|--------> driver_cfg[0]  (type=svt_pcie_driver_app_configuration)
|
|--------> requester_cfg (type=svt_pcie_requester_app_configuration)
```

```
|
|--------> target_cfg[0]   (type=svt_pcie_target_app_configuration)
|
|--------> pcie_cfg        (type=svt_pcie_configuration)
|
                            |--------> tl_cfg  (type=svt_pcie_tl_configuration)
                            |
                            |--------> dl_cfg  (type=svt_pcie_dl_configuration)
                            |
                            |--------> pl_cfg  (type=svt_pcie_pl_configuration)
```

The configuration object of the driver application resides inside `svt_pcie_device_configuration`, the configuration class of the PCIe Device Agent class. The class object is instanced as `driver_cfg[0]`.

For more details, such as members, class inheritance UML, list of tasks, list of constraints and so on, see the HTML class description of `svt_pcie_device_configuration` at:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_driver_app_configuration.html

17.2.1 Initial Configuration

The initial configuration of the driver application is established using the configuration database (`uvm_config_db`) class defined in UVM. The driver gets configured when the PCIe device agent gets configured by the test as illustrated in Example 12-1. The configuration attributes defined inside `svt_pcie_device_configuration::driver_cfg[0]` can be programmed to user-desired values before calling `uvm_config_db::set()`.

17.2.2 Dynamic Configuration

The configuration of the driver application set during the `build_phase()` can be dynamically (during the `run_phase`) modified if the test requires a change. This dynamic change can be made using any of the three mechanisms which is described in section Dynamic Configuration.

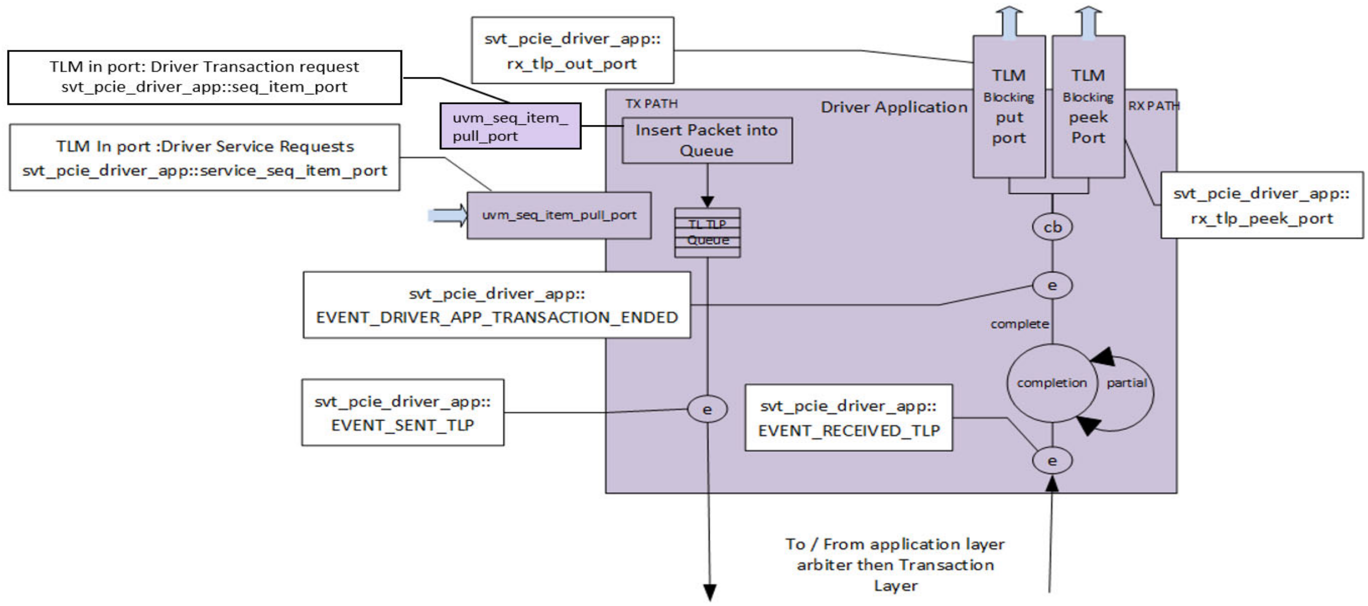
17.3 Driver Application Data Flow

The Driver Application is responsible for accepting transaction items which are converted into TLP data. On the receive side, it will accumulate completions for the end user.

17.3.1 Flow Diagram

Figure 17-2 highlights the key components of data flow through the VIP's Driver. Available TLMs, events (e), and callbacks (cb) are shown in the diagram.

Figure 17-2 Flow Digram – Driver Application



17.3.2 Sequencer Ports

Sequencer ports relevant to the driver have already been discussed in chapter PCIe Device Agent.

17.3.3 Sequences

The main purpose of the driver application is the generation of PCIe TLP traffic. And so it deals with the sequencing transaction layer packets, a function which is facilitated by the UVM sequencers of the driver application. The sequencers of the driver application will accept sequences queued on it by the UVM test and will arbitrate the driving of these sequences. The driver has two kinds of sequences:

1. Transaction sequencers: A transaction sequencer is used to schedule sequences that result in PCIe transactions (TLPs) on the PCIe bus. The transaction sequencer of the driver application is defined inside `svt_pcie_device_agent` class. It is of class type `svt_pcie_driver_app_transaction_sequencer` and is instanced as `driver_transaction_seqr[0]`. The transaction sequencer feeds transactions of type `svt_pcie_driver_app_transaction` to the SIPP (sequence item Pull Port) `seq_item_port` of UVM driver `svt_pcie_driver_app` class.
2. Service sequencers: A service sequencer is used to schedule sequences that are referred to as services on the driver application. An example of a service request would be a request to reset the driver application, which would drop all queued or outstanding transactions. The service sequencer is of class type `svt_pcie_driver_app_service_sequencer` and is instanced as `driver_seqr`. The sequencer feeds transactions of type `svt_pcie_driver_service` to the SIPP (sequence item Pull Port) `service_seq_item_port` of UVM driver `svt_pcie_driver_app` class.

   The tests can use these sequencers to generate a desired pattern of traffic or seek a service of the kinds supported by the driver application. The subsequent sections will discuss these in detail.

17.3.3.1 Transaction Sequences

The driver application of the PCIe VIP is the recommended testing interface while developing TLP traffic tests. The transaction sequences are sequenced via `svt_pcie_device_agent::driver_transaction_seqr[0]`. The application understands a description of the transaction layer packet as described by transaction class `svt_pcie_driver_app_transaction`.

To view the complete list of members and constraints in class `svt_pcie_driver_app_transaction` and their definitions, see the HTML class reference documentation at the following location:

> *DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_driver_app_transaction.html*

Table 17-1 Transaction Fields

											Config only		Message only			Responses only		
Transaction Type	**address**	**length**	**first_dw_be**	**last_dw_be**	**traffic_class**	**address_translation**	**ep**	**payload**	**exception_list**	**block**	**cfg_type**	**rgister_ number**	**routing_type**	**message_code**	**vendor_fields**	**payload**	**command_num**	**completion_status**
MEM_RD	**x**	**x**	**x**	**x**	**x**	**x**			**x**	**x**						**x**	**x**	**x**
MEM_RD_LK	x	x	x	x	x	x			x	x						x	x	x
MEM_WR	x	x	x	x	x	x	x	x	x	x							x	x
IO_RD	x		x						x	x						x	x	x
IO_WR	x							x	x									
CFG_RD	x*		x						x	x	x	x				x*	x	x
CFG_WR	x*						x	x*	x	x	x	x					x	x
MSG					x		x		x	x				x	x		x	x
ATOMIC_OP_FETCH_ADD	x	x	x	x	x	x	x	x	x	x							x	x
ATOMIC_OP_SWAP	x	x	x	x	x	x	x	x	x	x							x	x
ATOMIC_OP_CAS	x	x	x	x	x	x	x	x	x	x							x	x

*CFG types: address is {B,D,F}, payload is payload[0] only

The `svt_pcie_driver_app_transaction` class is used to create UVM sequences that are then scheduled on the transaction sequencer (`svt_pcie_device_agent::driver_transaction_seqr[0]`) of the driver application. Synopsys delivers a library of predefined sequences that can be readily used.

The list of these sequences can be found at:

> *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/sequencepages.html*

To get the list of all driver application transaction sequences, navigate to the following sequence:

Group	Subgroup
`DRIVER_TRANSACTION_PRIMITIVE_TYPE`	`DRIVER_TRANSACTIONS`

To view the complete list of constraints in each of the above sequence classes and the constraint definitions, see the HTML class reference documentation.

Example 12-8 illustrates the usage of the predefined sequence `svt_pcie_driver_app_transaction_mem_write_sequence`.

17.3.3.2 User-Defined Transaction Sequences

You can also define custom UVM sequences and create a library of sequences of your own. Example 17-1 shows the definition of a sequence that generates a constrained random sequence that issues a memory write followed by a memory read TLP.

Example 17-1

```
// Extend from VIP's base sequence class
class pcie_device_pseudo_random_sequence extends
svt_pcie_driver_app_transaction_base_sequence;
   rand svt_pcie_driver_app_transaction write_transaction;
   rand svt_pcie_driver_app_transaction read_transaction;

   `svt_xvm_object_utils(pcie_device_pseudo_random_sequence)

  // Constraints to ensure write TLP and read TLP requests are similar
   constraint c_read_follows_write {
      write_transaction.transaction_type == svt_pcie_driver_app_transaction::MEM_WR;
      read_transaction.transaction_type == svt_pcie_driver_app_transaction::MEM_RD;
      write_transaction.address == read_transaction.address;
      write_transaction.length == read_transaction.length;
      write_transaction.first_dw_be == read_transaction.first_dw_be;
      write_transaction.last_dw_be == read_transaction.last_dw_be;
      write_transaction.traffic_class == read_transaction.traffic_class;
      write_transaction.address_translation == read_transaction.address_translation;

      write_transaction.ep == 0;
      read_transaction.ep == 0;
      write_transaction.block == 1;
      read_transaction.block == 1;
   }

   extern virtual task body();
   extern function new(string name = "pcie_device_pseudo_random_sequence");

endclass: pcie_device_pseudo_random_sequence

function pcie_device_pseudo_random_sequence::new(string name);
   super.new(name);

  // Construct/create transactions
   `uvm_create(write_transaction)
   `uvm_create(read_transaction)
endfunction: new

task pcie_device_pseudo_random_sequence::body();
  // send the write followed by a read
   `uvm_info("body", "Entered...", UVM_LOW)
```

```
  // Issue the write transaction followed by a read
    `uvm_send(write_transaction);
    `uvm_send(read_transaction);

    `uvm_info("body", "Exiting...", UVM_LOW)
endtask
```

17.3.3.2.1 Using Transaction Sequences in UVM Tests

You can use any of the predefined sequences or user-defined sequences to generate TLP traffic from the driver application in the UVM tests. Example 17-2 illustrates an example test that generates TLP traffic using the sequence that is defined in Example 17-1.

Example 17-2

```
class pseudo_random_serdes_test extends uvm_test;

 /** UVM Component Utility macro */
  `uvm_component_utils(pseudo_random_serdes_test)

 /** Class Constructor */
  function new(string name = "pseudo_random_serdes_test", uvm_component parent=null);
    super.new(name,parent);
  endfunction: new

   virtual task run_phase(uvm_phase phase);
      pcie_device_pseudo_random_sequence p_rand_seq;
      svt_pcie_driver_app_service_wait_until_idle_sequence
wait_until_driver_idle_service_seq;

      `uvm_info("run_phase", "Entered...", UVM_LOW)

   // Raise UVM objection
      phase.raise_objection(this);

      …
   // Assumptions:
   // The UVM test has an instance of UVM environment named 'env'
   // The UVM environment has an instance of the PCIe device agent named 'root'
   // VIP's DL is enabled.
   // LTSSM is in L0
      p_rand_seq = new();
      p_rand_seq.cfg = env.endpoint_cfg;
      p_rand_seq.write_transaction.cfg = env.endpoint_cfg;
      p_rand_seq.read_transaction.cfg = env.endpoint_cfg;

   // Randomize the UVM sequence
      p_rand_seq.randomize with {
         write_transaction.address == 'h0000_0000;
         write_transaction.length == 2;
         write_transaction.first_dw_be == 4'b1111;
         write_transaction.last_dw_be == 4'b1111;
         write_transaction.traffic_class == 0;
```

```
            write_transaction.address_translation == 2'b00;
            foreach(write_transaction.payload[i])
               write_transaction.payload[i] == 'hbaadbaad;
         };
```

// Start the UVM sequence

```
         p_rand_seq.start(env.endpoint.driver_transaction_seqr[0]);

         wait_until_driver_idle_service_seq = new();
```

// Check for the driver is idle before ending test.

```
         wait_until_driver_idle_service_seq.start(env.endpoint.driver_seqr[0]); // End of
Test
```

// Drop UVM objection

```
         phase.drop_objection(this);
         `uvm_info("run_phase", "Exiting...", UVM_LOW)
      endtask

endclass //pseudo_random_serdes_test
```

In Example 17-2, the user-defined sequence `pcie_device_pseudo_random_sequence` is explicitly sequenced via the driver using the `svt_pcie_device_agent::driver_seqr[0]`. An alternative way to run the sequence is to configure the UVM sequencer to execute the `pcie_device_pseudo_random_sequence` as a `default_sequence` at the beginning of its `main_phase()`. This approach is illustrated in Example 17-3.

Example 17-3

```
class pseudo_random_serdes_test extends uvm_test;

 /** UVM Component Utility macro */
   `uvm_component_utils(pseudo_random_serdes_test)

 /** Class Constructor */
   function new(string name = "pseudo_random_serdes_test", uvm_component parent=null);
     super.new(name,parent);
   endfunction: new

    virtual function build_phase(uvm_phase phase);
       pcie_device_pseudo_random_sequence p_rand_seq;

       `uvm_info("build_phase", "Entered...", UVM_LOW)

    // Sets the default_sequence of the driver transaction sequencer

       uvm_config_db#(uvm_object_wrapper)::set(this,
"env.endpoint.driver_transaction_seqr[0].main_phase", "default_sequence",
pcie_device_pseudo_random_sequence::type_id::get());

       `uvm_info("build_phase", "Exiting...", UVM_LOW)
    endtask

endclass //pseudo_random_serdes_test
```

17.3.3.3 Service Sequences

Service sequences, unlike transaction sequences, will not generate transactions on the PCIe bus and are used to request a change in the behavior or sample a state value of the driver application. The service sequences are sequenced via `svt_pcie_device_agent::driver_seqr[0]`. The requested service is described by service transaction class `svt_pcie_driver_app_service`.

To view the complete list of members and constraints in class `svt_pcie_driver_app_service` and their definitions, see the HTML class reference documentation at the following location:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_driver_app_service.html

The `svt_pcie_driver_app_service` class is used to create UVM sequences that can then be scheduled on the service sequencer (`svt_pcie_device_agent::driver_seqr[0]`) of the driver application. Synopsys delivers a library of predefined sequences that can be readily used.

Because the predefined library of driver application service sequences cover all of the possible service types, you can meet all of the testing needs with the predefined driver application service sequences without having to develop a custom service sequence that is built using `svt_pcie_driver_app_service`. Example 17-2 illustrates the use of `svt_pcie_driver_app_service_wait_until_idle_sequence` to wait for the driver application to be idle as an end criterion for the test. Similarly, Example 17-4 shows the usage of `svt_pcie_driver_app_service_wait_for_compl_sequence`.

Example 17-4

// A custom sequence that generates a fairly random sequence of MemWr followed by a MemRd.

// Note: the sequence is non-blocking, the sequence will not wait for the completion of the posted MemRd.

```
class pcie_device_pseudo_random_non_blocking_sequence extends
svt_pcie_driver_app_transaction_base_sequence;
   rand svt_pcie_driver_app_transaction write_transaction;
   rand svt_pcie_driver_app_transaction read_transaction;

   `svt_xvm_object_utils(pcie_device_pseudo_random_non_blocking_sequence)

  // Constraints to ensure write TLP and read TLP requests are similar

   constraint c_read_follows_write {
      write_transaction.transaction_type == svt_pcie_driver_app_transaction::MEM_WR;
      read_transaction.transaction_type == svt_pcie_driver_app_transaction::MEM_RD;
      write_transaction.address == read_transaction.address;
      write_transaction.length == read_transaction.length;
      write_transaction.first_dw_be == read_transaction.first_dw_be;
      write_transaction.last_dw_be == read_transaction.last_dw_be;
      write_transaction.traffic_class == read_transaction.traffic_class;
      write_transaction.address_translation == read_transaction.address_translation;

      write_transaction.ep == 0;
      read_transaction.ep == 0;
      write_transaction.block == 0;
      read_transaction.block == 0;
   }

   extern virtual task body(); // Same implementation as in Example 17-1
   extern function new(string name = "pcie_device_pseudo_random_non_blocking_sequence");

endclass: pcie_device_pseudo_random_non_blocking_sequence
```

```
function pcie_device_pseudo_random_non_blocking_sequence::new(string name);
   super.new(name);

  // Construct/create transactions
   `uvm_create(write_transaction)
   `uvm_create(read_transaction)
endfunction: new

task pcie_device_pseudo_random_non_blocking_sequence::body();
  // send the write followed by a read
   `uvm_info("body", "Entered...", UVM_LOW)

  // Issue the write transaction followed by a read
   `uvm_send(write_transaction);
   `uvm_send(read_transaction);

   `uvm_info("body", "Exiting...", UVM_LOW)
endtask


class pseudo_random_serdes_test extends uvm_test;

 /** UVM Component Utility macro */
  `uvm_component_utils(pseudo_random_serdes_test)

 /** Class Constructor */
  function new(string name = "pseudo_random_serdes_test", uvm_component parent=null);
    super.new(name,parent);
  endfunction: new

   virtual task run_phase(uvm_phase phase);
      pcie_device_pseudo_random_non_blocking_sequence p_rand_seq;
      svt_pcie_driver_app_service_wait_for_compl_sequence wait_for_compl_service_seq;

      `uvm_info("run_phase", "Entered...", UVM_LOW)

    // Raise UVM objection
      phase.raise_objection(this);

      …
    // Assumptions:
    // The UVM test has an instance of UVM environment named 'env'
    // The UVM environment has an instance of the PCIe device agent named 'root'
    // VIP's DL is enabled.
    // LTSSM is in L0

      p_rand_seq = new();
      p_rand_seq.cfg = env.endpoint_cfg;
      p_rand_seq.write_transaction.cfg = env.endpoint_cfg;
      p_rand_seq.read_transaction.cfg = env.endpoint_cfg;
```

// Randomize the UVM sequence

```
p_rand_seq.randomize with {
   write_transaction.address == 'h0000_0000;
   write_transaction.length == 2;
   write_transaction.first_dw_be == 4'b1111;
   write_transaction.last_dw_be == 4'b1111;
   write_transaction.traffic_class == 0;
   write_transaction.address_translation == 2'b00;
   foreach(write_transaction.payload[i])
      write_transaction.payload[i] == 'hbaadbaad;
};
```

// Start the UVM sequence

```
   p_rand_seq.start(env.endpoint.driver_transaction_seqr[0]);

   wait_for_compl_service_seq = new();
   wait_for_comple_service_seq.command_number =
p_rand_seq.read_transaction.command_num;
```

// Start the service sequence.

// The sequence will block until the completion of the MemRd transaction of the pseudo random sequence to complete.

```
   wait_for_compl_service_seq.start(env.endpoint.driver_seqr[0]); // End of Test
```

// Drop UVM objection

```
   phase.drop_objection(this);
   `uvm_info("run_phase", "Exiting...", UVM_LOW)
 endtask

endclass
```
//pseudo_random_serdes_test

17.3.4 Callbacks and Events

As discussed in the Introduction section, the driver application is used to generate PCIe transaction requests targeting the link partner from a UVM test. The driver application tracks a transaction request from start to end and checks the completion of the transaction request for compliance to the PCIe specification. The application uses UVM callbacks and events to communicate the completion of a request. These UVM callbacks and events can be used for purposes such as functional coverage or scoreboarding.

17.3.4.1 Driver Application Callbacks

The driver application has a single callback function, which is issued by the driver application after the completion of every transaction request queued on the driver application.

> `transaction_ended` (`svt_pcie_driver_app` driver, `svt_pcie_driver_app_transaction` transaction): Callback issued by the driver application once the transaction request is completed. The completion information (for non-posted reads: `payload[]`, `completion_status`) returned by the link partner will be available inside `transaction`.

In UVM, callbacks are generally used to augment the UVM component behavior. Based on the implementation of the component, a callback could allow you to modify parameters of a transaction presented in the callback. Typical use of such augmentation would be an error injection. However, the callback in the driver only allows you to sample the end event of a transaction request along with completion status and data. And so the `transaction_ended()` callback can be used to capture transactions to be sent to a functional coverage model or a testbench scoreboard.

Steps to implement a driver callback

1. Define the callback class.

   The first step to implementing a callback in UVM is to define a callback that extends from the base `uvm_callback` class and implement the desired callback function. In case of the driver application callback the class would need to extend from `svt_pcie_driver_app_callback`, which falls under the `uvm_callback` inheritance hierarchy. Example 17-5 shows an example callback definition with `transaction_ended()` implemented to push the transaction into the queues of a scoreboard.

Example 17-5

```
class svt_pcie_driver_sb_callback extends svt_pcie_driver_app_callback;

 // Factory registration
  `svt_xvm_object_utils(svt_pcie_driver_sb_callback)

 // Testbench Scoreboard
  test_sb sb;

  function new(string name = "svt_pcie_driver_sb_callback");
    super.new();
  endfunction

  //
  virtual function void transaction_ended(svt_pcie_driver_app driver,
svt_pcie_driver_app_transaction transaction);
    `uvm_info("transaction_ended", $psprintf("driver transaction command number %0d has
ended - Queuing to scoreboard", transaction.command_num), UVM_MEDIUM)
    sb.driver_tx_queue.push_back(transaction); // Assuming the scoreboard has a smart queue to hold all
transmit packets from the driver
  endfunction

endclass
```

2. Register the callback.

   Once the callback class is implemented, one or more instances of the callback can be registered with the component, which in this case is the driver application. Example 17-6 shows an example test that illustrates how an instance of the callback defined in Example 17-5 is registered with the driver application.

Example 17-6

```
class pseudo_random_serdes_test extends uvm_test;

 /** UVM Component Utility macro */
  `uvm_component_utils(pseudo_random_serdes_test)

  svt_pcie_driver_sb_callback driver_sb_cb;

 /** Class Constructor */
  function new(string name = "pseudo_random_serdes_test", uvm_component parent=null);
    super.new(name,parent);
  endfunction: new

  virtual function void connect_phase (uvm_phase phase);
```

```
    super.connect_phase(phase);

    driver_sb_cb = new();
  // Assign the scoreboard instance inside the callback with a reference to the scoreboard in the testbench env
    driver_sb_cb.sb = this.env.sb;
    svt_pcie_driver_app_callback::add(env.root.driver[0], driver_sb_cb);
 endfunction
   …
endclass
```

17.3.4.2 Driver Events

The driver application triggers the following events that can be sampled by the testbench for controlling test sequences.

- `svt_pcie_driver_app::EVENT_DRIVER_APP_TRANSACTION_ENDED`: A `uvm_event` that is triggered when the transaction is completed by the driver application.
- `svt_pcie_driver_app::EVENT_RECEIVED_TLP`: A `uvm_event` that is triggered when a transaction is received by the driver application when configured as an application agent.
- `svt_pcie_driver_app::EVENT_SENT_TLP`: A `uvm_event` that is triggered when a transaction is sent by the driver application when configured as an application agent.

All of the UVM events listed in this section are of type `uvm_event` and hence they derive all the methods and features of the UVM event class. Table 17-2 lists the user methods and describes each of them.

Table 17-2 User Methods From the UVM Event Class

Method Name	Purpose	Description
`task wait_on(bit delta=0)`	Wait on event	This task waits for the event to be turned "on". This tasks will return immediately if the event has already been triggered.If 'delta=1', the caller will be forced to wait a single delta (#0) before returning. This prevents the caller from returning before previously waiting processes have had a chance to resume.Once an event is triggered, it will remain "on" until the event is reset().
`task wait_off(bit delta=0)`		This task waits for the event to be tuned "off" via a call to reset(). This task will return immediately if the event has already been triggered.If 'delta=1', the caller will be forced to wait a single delta (#0) before returning. This prevents the caller from returning before previously waiting processes have had a chance to resume.
`task wait_trigger()`		This task waits for the event to be triggered. If one process calls the wait_trigger() as another process, a race condition occurs. If the call to the wait_trigger() occurs after the trigger, this method will not return until the next trigger, which may never occur and thus cause a deadlock.
`task wait_ptrigger()`		This task waits for a 'persistent' trigger of the event. Unlike wait_trigger(), this task views the trigger as persistent within a given time-slice and thus avoids certain race conditions. If this method is called after the trigger but within the same time-slice, the caller returns immediately.

Table 17-2 User Methods From the UVM Event Class

Method Name	Purpose	Description
`function uvm_object get_trigger_data()`	Get trigger information	This function returns the transaction for which the event was triggered.This function would return an object of type `svt_pcie_driver_app_transaction` if called after the event `trigger()`.
`function time get_trigger_time()`		This function indicates the time when the event was last triggered. If the event has not been triggered, or the event has been reset, then the trigger time returned will be 0.
`function is_on()`	Sample/Reset state value	This function indicates whether the event has been triggered since it was last reset.A value 1 indicates the event has triggered.
`function is_off()`		This function indicates whether the event has not been triggered or been reset.A value 1 indicates that the event has not been triggered.
`function reset(bit wakeup=0)`		This function resets the event to its off state. If `wakeup=1`, then all processes currently waiting for the even are activated before the reset.
`function add_callback(uvm_event_callback cb, bit append=1)`	Attach/Detach callback	This function registers a callback object (type `uvm_event_callback`), `cb`, with the event. The callback object includes a `pre_trigger` and `post_trigger` functionality.If `append=1`, the default, `cb` is added to the back of the callback list. Otherwise, `cb` is placed at the front of the callback list.
`function delete callback(uvm_event_callback cb)`		This function unregisters the given callback, `cb`, from the event.
`function void cancel()`	Get/Set list of waiters	This function decrements the number of waiters on the event.This is used if a process that is waiting on an event is disabled or activated by some other means.
`function in get_num_waiters()`		This function returns the number of processes waiting on the event.

Example 17-7 illustrates how to wait for a trigger on event `EVENT_DRIVER_APP_TRANSACTION_ENDED` and collect trigger data post the event trigger. The test issues a non-blocking driver application transaction sequence before waiting on a trigger from the event and collecting the trigger data.

Example 17-7

```
class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_driver_app_mem_write_sequence mem_wr_seq;
      svt_pcie_driver_app_mem_read_sequence mem_rd_seq;
      svt_configuration temp_cfg = null;
         …
```

// Assumptions:

// The UVM test has an instance of UVM environment named 'env'

// The UVM environment has an instance of the PCIe device agent named 'root'

// VIP's DL is enabled.

// LTSSM is in L0

```
    mem_wr_seq = new();
    mem_rd_seq = new();

    env.root.get_cfg_via_task(temp_cfg);
    $cast(mem_wr_seq.cfg, temp_cfg.clone());
    $cast(mem_rd_seq.cfg, temp_cfg.clone());

    mem_wr_seq.randomize with {
       transaction_type == svt_pcie_driver_app_transaction::MEM_WR;
       address == 'h8000;
       length == 2;
       first_dw_be == 4'b1111;
       last_dw_be == 4'b1111;
       traffic_class == 0;
       address_translation == 2'b00;
       ep == 0;
       foreach(write_payload[i])
         write_payload[i] == 'hc0de_0000 + i;
       block == 0;
    };
    mem_wr_seq.start(env.root.driver_transaction_seqr[0]);
```

// Wait for a trigger on EVENT_DRIVER_APP_TRANSACTION_ENDED

```
    env.root.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.wait_trigger();
    `uvm_info("run_phase", $psprintf("EVENT_DRIVER_APP_TRANSACTION_ENDED just
triggered at %0t.",
env.root.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.get_trigger_time()), UVM_MEDIUM)
```

// Get data transaction object associated with the latest trigger of EVENT_DRIVER_APP_TRANSACTION_ENDED

```
    get_trg_uvm_obj =
env.root.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.get_trigger_data();
    $cast(get_trg_drv_trans_obj, get_trg_uvm_obj);
```

// We expect the following print() function to print MEM_WR transaction initiated by the test.

```
    get_trg_drv_trans_obj.print();

    mem_rd_seq.randomize with {
       transaction_type == svt_pcie_driver_app_transaction::MEM_RD;
       address == 'h8000;
       length == 2;
       first_dw_be == 4'b1111;
       last_dw_be == 4'b1111;
       traffic_class == 0;
       address_translation == 2'b00;
       ep == 0;
       block == 0;
    };
    mem_rd_seq.start(env.root.driver_transaction_seqr[0]);
```

// Wait for a trigger on EVENT_DRIVER_APP_TRANSACTION_ENDED

```
      env.root.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.wait_trigger();
      `uvm_info("run_phase", $psprintf("EVENT_DRIVER_APP_TRANSACTION_ENDED just
triggered at %0t.",
env.endpoint.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.get_trigger_time()),
UVM_MEDIUM)
```

// Get data transaction object associated with the latest trigger of EVENT_DRIVER_APP_TRANSACTION_ENDED

```
      get_trg_uvm_obj =
env.root.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.get_trigger_data();
      $cast(get_trg_drv_trans_obj, get_trg_uvm_obj);
```

// We expect the following print() function to print MEM_RD transaction initiated by the test.

```
      get_trg_drv_trans_obj.print();
```

// End of Test

```
   endtask
endclass
```

The UVM event class has a built-in callback mechanism that can be used by the testbench for performing additional book-keeping functions with each trigger of any of the events. Class `uvm_event_callback` is the base callback class which can be extended for user-defined purposes. The callback functions provided are:

1. `pre_trigger` (`uvm_event e, uvm_object data`): This callback is called just before triggering the associated event. In a derived class, this method should be overridden with an implement of a user desired pre-trigger functionality. If the implementation of the function returns 1, then the event will not trigger and the post-trigger callback will not be called. This provides a way for a callback to prevent the event from triggering. In the function, `e` is the `uvm_event` that is being triggered, and `data` is the transaction object associated with the event being triggered.
2. `post_trigger` (`uvm_event e, uvm_object data`): This callback is called after triggering the associated event. In a derived class, this method should be overridden with an implement of a user desired post-trigger functionality. In the function, `e` is the `uvm_event` that is being triggered, and `data` is the transaction object associated with the event being triggered.

Example 17-8 illustrates how a `post_trigger()` callback is used to sample the data object associated with the event. The sampling of data object in this example can be compared to the sampling of the data object in Example 17-7, which does an in-line sample of the data object using `uvm_event::get_trigger_data()`.

Example 17-8

// Define the UVM event callback class extending from uvm_event_callback

```
class event_driver_app_transaction_ended_callback extends uvm_event_callback;
   `svt_xvm_object_utils(event_driver_app_transaction_ended_callback)

   function new(string name = "event_driver_app_transaction_ended_callback");
     super.new(name);
   endfunction
```

// Implementing post_trigger function for processing the related data object post-trigger of the UVM event

```
   virtual function void post_trigger (uvm_event e, uvm_object data);
      `uvm_info("post_trigger", "EVENT_DRIVER_APP_TRANSACTION_ENDED has triggered",
UVM_LOW)
```

// Printing the data associated with the event that is about to be triggered.

// Here we only print the object. This object could be posted to a coverage model to sample functional coverage or a scoreboard to check against data received at the remote receiver.

```
      data.print();

      return;
   endfunction

endclass

class my_pcie_test extends uvm_test;
   `uvm_component_utils(my_pcie_test)
   event_driver_app_transaction_ended_callback transaction_ended_cb;
   …
   function void connect_phase(uvm_phase phase);
      super.connect_phase(phase);
```

// Construct and register the event callback with EVENT_DRIVER_APP_TRANSACTION_ENDED

```
      transaction_ended_cb = new("transaction_ended_cb");

env.endpoint.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.add_callback(transaction_ended
_cb);
   endfunction

   task run_phase (uvm_phase phase);
      svt_pcie_driver_app_mem_write_sequence mem_wr_seq;
      svt_pcie_driver_app_mem_read_sequence mem_rd_seq;
      svt_configuration temp_cfg = null;
         …
```

// Assumptions:

// The UVM test has an instance of UVM environment named 'env'

// The UVM environment has an instance of the PCIe device agent named 'root'

// VIP's DL is enabled.

// LTSSM is in L0

```
      mem_wr_seq = new();
      mem_rd_seq = new();

      env.root.get_cfg_via_task(temp_cfg);
      $cast(mem_wr_seq.cfg, temp_cfg.clone());
      $cast(mem_rd_seq.cfg, temp_cfg.clone());

      mem_wr_seq.randomize with {
         transaction_type == svt_pcie_driver_app_transaction::MEM_WR;
         address == 'h8000;
         length == 2;
         first_dw_be == 4'b1111;
         last_dw_be == 4'b1111;
         traffic_class == 0;
         address_translation == 2'b00;
         ep == 0;
         foreach(write_payload[i])
           write_payload[i] == 'hc0de_0000 + i;
         block == 0;
      };
```

```
        mem_wr_seq.start(env.root.driver_transaction_seqr[0]);
```

// Wait for a trigger on EVENT_DRIVER_APP_TRANSACTION_ENDED

```
        env.root.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.wait_trigger();
        `uvm_info("run_phase", $psprintf("EVENT_DRIVER_APP_TRANSACTION_ENDED just 
triggered at %0t.", 
env.root.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.get_trigger_time()), UVM_MEDIUM)
```

// post_trigger() callback for EVENT_DRIVER_APP_TRANSACTION_ENDED of MEM_WR is expected to occur at this point in time.

```
        mem_rd_seq.randomize with {
           transaction_type == svt_pcie_driver_app_transaction::MEM_RD;
           address == 'h8000;
           length == 2;
           first_dw_be == 4'b1111;
           last_dw_be == 4'b1111;
           traffic_class == 0;
           address_translation == 2'b00;
           ep == 0;
           block == 0;
        };
        mem_rd_seq.start(env.root.driver_transaction_seqr[0]);
```

// Wait for a trigger on EVENT_DRIVER_APP_TRANSACTION_ENDED

```
        env.root.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.wait_trigger();
        `uvm_info("run_phase", $psprintf("EVENT_DRIVER_APP_TRANSACTION_ENDED just 
triggered at %0t.", 
env.endpoint.driver[0].EVENT_DRIVER_APP_TRANSACTION_ENDED.get_trigger_time()), 
UVM_MEDIUM)
```

// post_trigger()callback for EVENT_DRIVER_APP_TRANSACTION_ENDED of MEM_RD is expected to occur at this point in time.

// End of Test

```
     endtask
endclass
```

17.3.5 Exceptions

The PCIe VIP provides a mechanism to create error scenarios with the use of a methodology that includes:

- Exception Object: An object that encapsulates the error information such as kind, corrupted data value (if applicable), error weights (for injection based on statistics) and so on. In the driver application, class `svt_pcie_driver_app_transaction_exception` models the exception.

  To view the complete list of constraints in class `svt_pcie_driver_app_transaction_exception` and their definitions, see the HTML class reference documentation:

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_driver_app_transaction_exception.html

  Class `svt_pcie_driver_app_transaction_exception` has functions that you can use to query/modify the exception object. The most commonly used ones are:

  - `has_*_error()`: A group of functions to query if the exception object has a specific kind of error. For example, `has_auto_corrupt_lcrc_error()` indicates if the exception object has

AUTO_CORRUPT_LCRC error. The function returns a bit value, 1 indicates that error is present and 0 indicates error is not present.

Note To view the full list or predefined error types supported by the driver application, see the HTML class reference documentation:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_driver_app_transaction_exception.html#item_error_kind_enum

- reasonable_constraint_mode (bit on_off): A function that can be used to turn OFF/ON the reasonable constraints defined in svt_pcie_driver_app_transaction_exception class. By default, all reasonable constraints are ON. The function will return a value -1 if it failed to perform the operation of turning the constraint ON/OFF, else it would return a value same as the input bit on_off.
- set_constraint_weights (int new_weight): A function that can be used to set the relative weight distribution for all exception kinds as a block. Sets the specified weight for all _wt attributes except NO_ERROR_wt.

  This is not an exhaustive list. For the full list, see HTML description of the svt_pcie_driver_app_transaction_exception class:

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_driver_app_transaction_exception.html

- Exception List Object: An object that encapsulates an instance/multiple instances of exception object/objects and other information such as number of errors, exception factory (for randomized error injection), and so on. In the driver application, class svt_pcie_driver_app_transaction_exception_list models the exception list. Class svt_pcie_driver_app_transaction_exception_list extends from svt_exception_list, a SVT base class.

  To view the complete list of members and constraints in class svt_pcie_driver_app_transaction_exception_list and their definitions, see HTML class reference documentation:

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_driver_app_transaction_exception_list.html

  Class svt_pcie_driver_app_transaction_exception_list has a number of user functions, which finds use in writing exception tests. The most commonly used ones are:

  - setup_randomized_exception (svt_pcie_device_configuration cfg, svt_pcie_driver_app_transaction xact): A function used to adjust the object reference xact, an instance of type svt_pcie_driver_app_transaction, inside svt_pcie_driver_app_transaction_exception to make it point to the transaction which is being injected with an error/exception. This function should be called if exception is being injected using the factory object randomized_exception and it should be called before the call to randomize the exception list object.
  - num_exception_first_randomize(): A function used to randomize the exception list. The function is different from a simple SystemVerilog randomize() since it first randomizes num_exceptions first and then rest of the exception list class, populating the list with an array of random exceptions. The function returns a bit value indicating the result of the randomization. A value indicates a success and a value 0 indicates failure.

- `reasonable_constraint_mode`(bit `on_off`): A function that can be used to turn `OFF/ON` the reasonable constraints defined in `svt_pcie_driver_app_transaction_exception_list` class. By default, all reasonable constraints are `ON`. The function will return a value -1 if it fails to perform the operation of turning the constraint `ON/OFF`, else it would return a value same as the input bit `on_off`.
- `set_constraint_weights` (int `new_weight`): A function that can be used to set the relative weight distribution for all exception kinds of every exception object in the array of exception objects that are part of the exception list class. Sets the specified weight for all `_wt` attributes except `NO_ERROR_wt`.

  Note: This is not an exhaustive list. For the full list, see HTML descriptions of `svt_pcie_driver_app_transaction_exception_list` class:

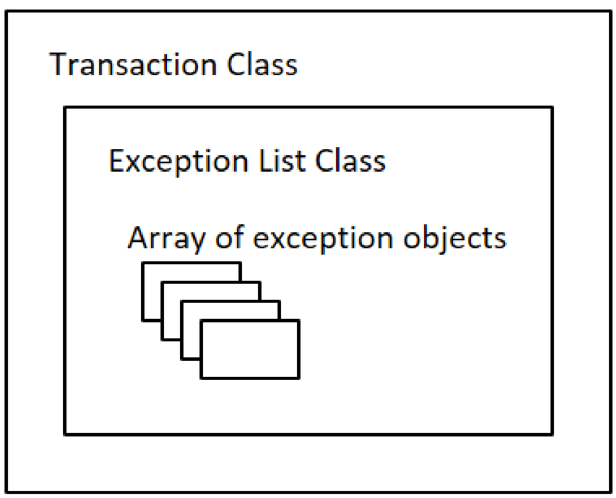
  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_driver_app_transaction_exception_list.html

  `svt_exception_list` class:

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_exception_list.html#item_add_exception

The following diagram shows how an exception list is related to an exception object and how a transaction object relates to an exception list object.

Figure 17-3 Exception List, Exception, Transaction

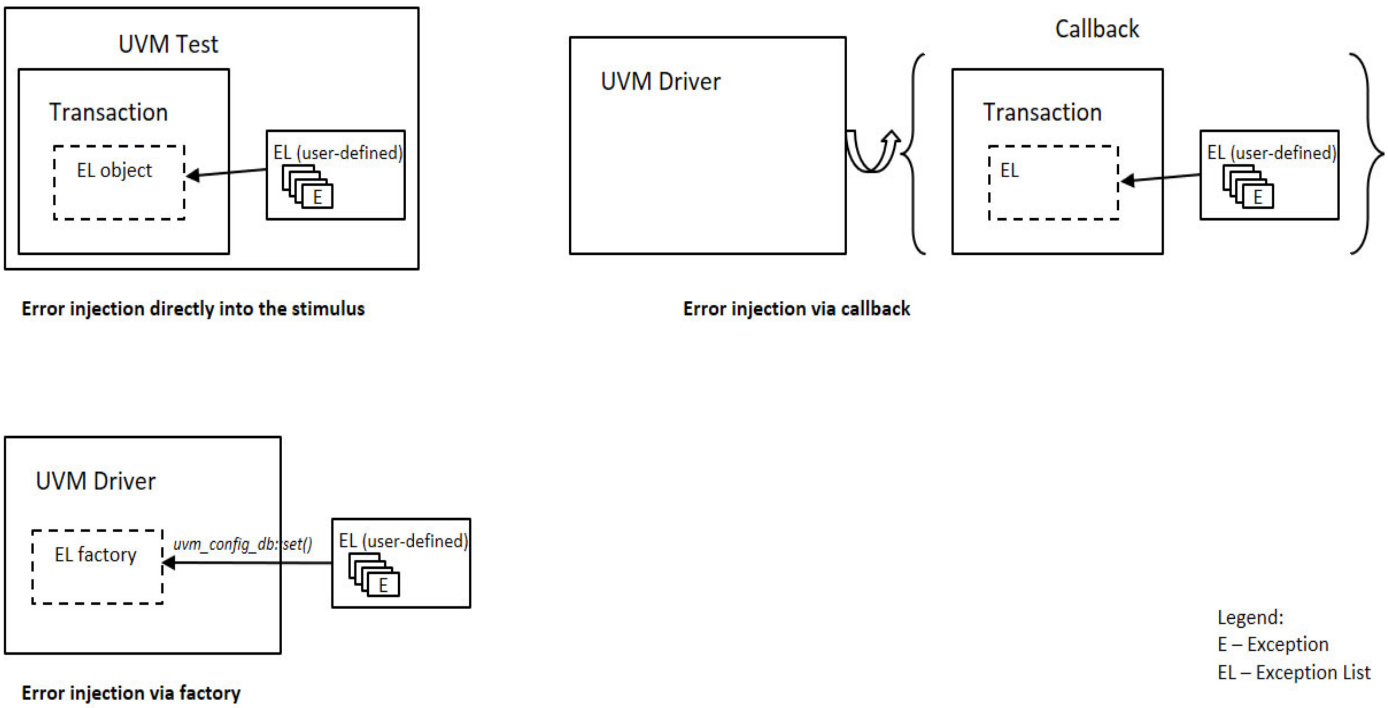


The exception methodology in the PCIe VIP as a whole allows three different methods to inject errors:

1. Exception in stimulus: This method involves the implementation of a test that attaches a list of errors, or exceptions as they are known in the VIP, to a transaction while it is being queued on the VIP.
2. Exception via Callbacks: This method involves the implementation of a callback method to attach errors/exceptions to a transaction that is transiting via the callback method. As discussed in the driver application callbacks section, the driver application has a single callback function `transaction_ended()`, which is issued after a transaction ends, so it cannot serve the purpose of error/exception injection. Thus, this mechanism is not applicable to the driver application. However, other layers/application such as the DL, PL, target application, and so on, have callbacks that can be used for the purpose of injecting error/exceptions.

3. Exception via Component (Driver Application) Factory: This method involves the implementation of a user-defined exception list class with constraints that will cause constrained random pattern of errors and the setting of the exception list factory object inside the VIP to the user-defined exception list.

Figure 17-4 Error Injection Methods



17.3.5.1 Exception in Stimulus

One of the two mechanisms of introducing errors or exceptions in packets generated by the driver application is to attach an exception list object to the driver application transaction being queued on the driver application. Example 17-9, a modified form of Example 17-2, illustrates the injection of a directed error (LCRC corruption) into the memory read transaction of the UVM sequence.

Example 17-9

```
class pseudo_random_serdes_test extends uvm_test;

 /** UVM Component Utility macro */
  `uvm_component_utils(pseudo_random_serdes_test)

 /** Class Constructor */
  function new(string name = "pseudo_random_serdes_test", uvm_component parent=null);
    super.new(name,parent);
  endfunction: new

   virtual task run_phase(uvm_phase phase);
      pcie_device_pseudo_random_sequence p_rand_seq;
```

```
        svt_pcie_driver_app_service_wait_until_idle_sequence
wait_until_driver_idle_service_seq;
```

// Declare an exception list object for error injection.

```
        svt_pcie_driver_app_transaction_exception_list my_rand_exc_list;

        `uvm_info("run_phase", "Entered...", UVM_LOW)
```

// Raise UVM objection

```
        phase.raise_objection(this);

        …
```

// Assumptions:

// The UVM test has an instance of UVM environment named 'env'

// The UVM environment has an instance of the PCIe device agent named 'root'

// VIP's DL is enabled.

// LTSSM is in L0

```
        p_rand_seq = new();
        p_rand_seq.cfg = env.endpoint_cfg;
        p_rand_seq.write_transaction.cfg = env.endpoint_cfg;
        p_rand_seq.read_transaction.cfg = env.endpoint_cfg;
```

// Randomize the UVM sequence

```
        p_rand_seq.randomize with {
           write_transaction.address == 'h0000_0000;
           write_transaction.length == 2;
           write_transaction.first_dw_be == 4'b1111;
           write_transaction.last_dw_be == 4'b1111;
           write_transaction.traffic_class == 0;
           write_transaction.address_translation == 2'b00;
           foreach(write_transaction.payload[i])
              write_transaction.payload[i] == 'hbaadbaad;
        };
```

// Construct the exception list object

```
        my_rand_exc_list = new();
```

// Initialize the exception list inside the read transaction, the transaction targeted for error injection

```
        p_rand_seq.read_transaction.exception_list = my_rand_exc_list;
```

// Populate the expcetions[] array inside the exception list object.

// The VIP will create a single element inside exceptions[] by virtue of the svt_pcie_driver_app_transaction_exception_list::max_num_exceptions being initialized to 1.

// Set svt_pcie_driver_app_transaction_exception_list::max_num_exceptions to a higher value if more than a single error is desired.

```
        p_rand_seq.read_transaction.exception_list.populate_exceptions();
```

// Specify the desired error kind.

```
        p_rand_seq.read_transaction.exception_list.exceptions[0].error_kind =
svt_pcie_driver_app_transaction_exception::AUTO_CORRUPT_LCRC;
```

// Inject the specified error.

```
        p_rand_seq.read_transaction.exception_list.inject_exceptions();
```

```
        // Start the UVM sequence
            p_rand_seq.start(env.root.driver_transaction_seqr[0]);

            wait_until_driver_idle_service_seq = new();
        // Check for the driver is idle before ending test.
            wait_until_driver_idle_service_seq.start(env.root.driver_seqr[0]); // End of Test

        // Drop UVM objection
            phase.drop_objection(this);
            `uvm_info("run_phase", "Exiting...", UVM_LOW)
        endtask
    endclass //pseudo_random_serdes_test
```

The simulation log will print the following messages while it injects the LCRC corruption error and as part of the error handling:

```
UVM_INFO @ 17565152.50 ps: uvm_test_top.env.endpoint.driver0 [report_message]
SetEiCodeForNextTransaction: EI - 'TX_TLP_EI_CORRUPT_LCRC' requested with corrupted_data
= 0
…
UVM_INFO @ 17585152.50 ps: uvm_test_top.env.endpoint.port0.dl0 [report_message]
GenerateTxTLPEi:  EI - Injecting LCRC violation.  Expect DUT to send NAK.
…
UVM_INFO @ 17765152.50 ps: uvm_test_top.env.endpoint.port0.dl0 [report_message]
ProcessReceivedDLLP: EI - Received NAK as expected.
```

The transaction log will tag the error packet as shown below. The retry following the error packet is also tagged.

```
                            Start     End   D
  Reporter        Time       Time    I TLP Type R_ID/Tag|ST   Seq ....           Prefix / Header / Data          E    ECRC        LCRC       TX/RX
                  (ns)       (ns)    R       DLLP Type        Num ....           (All values in Hex)             P                 CRC        Status
------------ --------- --------- - --------------------- ---- ....  --------- --------- --------- --------- - ---------- ---------- --------
  ...
endpoint0    17589.000 17613.000 T MWr32       0x0001/12     1 ....  H40000002 H000112ff H00000000    ---    0            0x7f477092
                                                                 ....   baadbaad  baadbaad    ---        ---
  ...
endpoint0    17621.000 17637.000 T MRd32       0x0001/1b     2 ....  H00010002 H00011b34 H00000003    ---                 0xda8d515d LCRC
endpoint0    17733.000 17765.000 R NAK                        1 ....                                                           0xf91e
  ...
endpoint0    17805.000 17821.000 T MRd32       0x0001/1b     2 ....  H00010002 H00011b34 H00000003    ---                 0xda8d515c Retry
  ...
endpoint0    17921.000 17973.000 R CplD        0x0001/1b     1 ....  H4a000002 H01000008 H00011b00    ---    0            0xedaf072c
                                                                 ....   baadbaad  baadbaad    ---        ---
 ...
```

Note

The error handling in the VIP will vary for different error kinds. The log snippets shown above are specific to the `AUTO_CORRUPT_LCRC` error kind. Error injection related VIP messages are generally tagged with "`EI -`".

In Example 17-9, the `error_kind` is set to a specific value, making the test a very directed one. A more randomized approach can also be used by using the `randomized_exception`, factory object inside the exception list class, and setting the relative weight distribution variables provided in the exception class. Example 17-10, a modified form of example, illustrates the injection of an error (LCRC corruption) using `randomized_exception` object in the exception list class.

Example 17-10

```
class pseudo_random_serdes_test extends uvm_test;

 /** UVM Component Utility macro */
  `uvm_component_utils(pseudo_random_serdes_test)

 /** Class Constructor */
  function new(string name = "pseudo_random_serdes_test", uvm_component parent=null);
    super.new(name,parent);
  endfunction: new

   virtual task run_phase(uvm_phase phase);
      pcie_device_pseudo_random_sequence p_rand_seq;
      svt_pcie_driver_app_service_wait_until_idle_sequence
wait_until_driver_idle_service_seq;
   // Declare an exception list object for error injection.
      svt_pcie_driver_app_transaction_exception_list my_rand_exc_list;

      `uvm_info("run_phase", "Entered...", UVM_LOW)

   // Raise UVM objection
      phase.raise_objection(this);

      …
   // Assumptions:
   // The UVM test has an instance of UVM environment named 'env'
   // The UVM environment has an instance of the PCIe device agent named 'root'
   // VIP's DL is enabled.
   // LTSSM is in L0
      p_rand_seq = new();
      p_rand_seq.cfg = env.endpoint_cfg;
      p_rand_seq.write_transaction.cfg = env.endpoint_cfg;
      p_rand_seq.read_transaction.cfg = env.endpoint_cfg;

   // Randomize the UVM sequence
      p_rand_seq.randomize with {
         write_transaction.address == 'h0000_0000;
         write_transaction.length == 2;
         write_transaction.first_dw_be == 4'b1111;
         write_transaction.last_dw_be == 4'b1111;
         write_transaction.traffic_class == 0;
         write_transaction.address_translation == 2'b00;
         foreach(write_transaction.payload[i])
            write_transaction.payload[i] == 'hbaadbaad;
      };

      my_rand_exc = new();
   // Set the constraint weight of all error kinds & NO_ERROR_wt to 0
```

```
        my_rand_exc.set_constraint_weights(0);
        my_rand_exc.NO_ERROR_wt = 0;
    // Set the constraint weight of AUTO_CORRUPT_LCRC to a non-zero value
        my_rand_exc.AUTO_CORRUPT_LCRC_wt = 100;

        my_rand_exc_list = new();
    // Set the factory exception inside my_rand_exc_list with the my_rand_exc
        my_rand_exc_list.randomized_exception = my_rand_exc;
    // Associate the transaction and the exception object. Note: configuration of the device must also be provided.
        my_rand_exc_list.setup_randomized_exception(env.cfg.endpoint_cfg,
p_rand_seq.read_transaction);
    // Randomize 'num_exceptions' first, this will result in num_exceptions will result in a random value.
    // By default the constraints sets a cap of 1 by virtue of
svt_pcie_driver_app_transaction_exception_list::max_num_exceptions being set to 1
    // If a highler count of errors are required program
svt_pcie_driver_app_transaction_exception_list::max_num_exceptions to a higher value before
    // calling num_exceptions_first_randomize()
        void'(my_rand_exc_list.num_exceptions_first_randomize());
    // Note: num_exceptions_first_randomize() could result in num_exception = 0 or 1 (default constraints), causing a
random possibility of error.
        `uvm_info("run_phase", $psprintf("num_exceptions_first_randomize() resulted in
my_rand_exc_list.num_exceptions = %0d", my_rand_exc_list.num_exceptions), UVM_LOW)
    //p_rand_seq.read_transaction.exception_list = my_rand_exc_list;


    // Start the UVM sequence
        p_rand_seq.start(env.root.driver_transaction_seqr[0]);

        wait_until_driver_idle_service_seq = new();
    // Check for the driver is idle before ending test.
        wait_until_driver_idle_service_seq.start(env.root.driver_seqr[0]); // End of Test

    // Drop UVM objection
        phase.drop_objection(this);
        `uvm_info("run_phase", "Exiting...", UVM_LOW)
    endtask

endclass //pseudo_random_serdes_test
```

In Example 17-10, the relative weight distribution of errors is set to resolve to `AUTO_CORRUPT_LCRC` error kind always. The weights can be modified to pick and choose from different error kinds.

The VIP also provides a predefined set of driver application transaction sequences that can be readily used to generate all error kinds supported by the driver application. Table 17-3 lists these sequence classes.

Table 17-3 : Predefined UVM Sequences with Built-in Exceptions

Sequence class name	Description
`svt_pcie_driver_app_mem_request_exception_sequence`	This is sequence is used to generate error injected `MemRd` and `MemWr` transactions from the driver application. The UVM sequence has the following members that users can use to program the number of errors and the kind of error: • `svt_pcie_driver_app_transaction_exception::error_kind_enum transaction_error_kind`: A rand type variable, which can be constrained to generate any of the error kinds listed in `svt_pcie_driver_app_transaction_exception` class, will determine the kind of error that will introduced into the memory transaction. • `int num_exceptions`: A rand type variable that will determine the number of errors to be introduced into the memory transaction. The UVM sequence has the following members that users can use to program the type of the memory transaction: - `rand svt_pcie_driver_app_transaction::transaction_type_enum transaction_type` - `rand bit[63:0] address` - `rand bit[9:0] length` - `rand bit[31:0] write_payload[]` - `rand bit[3:0] first_dw_be` - `rand bit[3:0] last_dw_be` - `rand bit[2:0] traffic_class` - `rand bit[1:0] address_translation` - `rand bit ep` - `rand bit block` - `rand bit th = 0` - `rand bit ln = 0` - `rand svt_pcie_driver_app_transaction::ph_enum ph` - `rand bit [7:0] st` - `int command_num` - `rand int unsigned pkt_delay_ns` - `rand bit reserved_byte1_bit7` - `rand bit reserved_byte1_bit1` - `rand bit reserved_byte1_bit3` - `rand bit reserved_byte1_bit0` - `rand bit[1:0] reserved_byte11_bit1_to_bit0` - `rand bit[1:0] reserved_byte15_bit1_to_bit0` For description of each of these data members, see class `svt_pcie_driver_app_transaction` in the HTML class reference documentation. Also note that most of these data members are all rand type. To find out the constraints that govern the randomization of these data members check lookup the definition of the class in file: *svt_pcie_driver_app_transaction_sequence_collection.sv* in the installation area.

Table 17-3 : Predefined UVM Sequences with Built-in Exceptions

Sequence class name	Description
svt_pcie_driver_app_io_cfg_request_exception_sequence	This sequence is used to generate error injected IO (`IO_WR, IO_RD`) and Config (`CFG_WR, CFG_RD`) transactions from the driver application. The UVM sequence has the following members that users can use to program the number of errors and the kind of error: • `svt_pcie_driver_app_transaction_exception::error_kind_enum transaction_error_kind`: A rand type variable, which can be constrained to generate any of the error kinds listed in `svt_pcie_driver_app_transaction_exception` class, will determine the kind of error that will introduced into the memory transaction. • `int num_exceptions`: A rand type variable that will determine the number of errors to be introduced into the memory transaction. The UVM sequence has the following members that users can use to program the type of the memory transaction: - `rand bit[63:0] address` - `rand bit[31:0] write_payload` - `rand bit[3:0] first_dw_be` - `rand bit ep` - `rand bit blockint command_num` - `rand int unsigned pkt_delay_ns` - `rand bit [9:0] register_number` - `rand bit cfg_type` - `rand bit reserved_byte1_bit7` - `rand bit reserved_byte1_bit1` - `rand bit reserved_byte1_bit3` - `rand bit reserved_byte1_bit0` - `rand bit[1:0] reserved_byte11_bit1_to_bit0` For description of each of these data members, see class `svt_pcie_driver_app_transaction` in the HTML class reference documentation. Also note that most of these data members are all rand type. To find out the constraints that govern the randomization of these data members check lookup the definition of the class in file: *svt_pcie_driver_app_transaction_sequence_collection.sv* in the installation area.

Example 17-11 illustrates the use of `svt_pcie_driver_app_mem_request_exception_sequence` to generate a memory read transaction with a corrupted LCRC.

Example 17-11

```
class pseudo_random_serdes_test extends uvm_test;

 /** UVM Component Utility macro */
  `uvm_component_utils(pseudo_random_serdes_test)

 /** Class Constructor */
  function new(string name = "pseudo_random_serdes_test", uvm_component parent=null);
    super.new(name,parent);
  endfunction: new

   virtual task run_phase(uvm_phase phase);
  // Pre-defined Driver App transaction sequence with built-in exception.
```

```
      svt_pcie_driver_app_mem_request_exception_sequence mem_req_exc_seq
      svt_pcie_driver_app_service_wait_until_idle_sequence
wait_until_driver_idle_service_seq;

      `uvm_info("run_phase", "Entered...", UVM_LOW)

   // Raise UVM objection
      phase.raise_objection(this);

      …
   // Assumptions:
   // The UVM test has an instance of UVM environment named 'env'
   // The UVM environment has an instance of the PCIe device agent named 'root'
   // VIP's DL is enabled.
   // LTSSM is in L0
      mem_req_exc_seq = new();
      mem_req_exc_seq.cfg = env.cfg.endpoint_cfg;

      mem_req_exc_seq.randomize with {
      // Programming the transaction type to a memory read TLP.
             transaction_type == svt_pcie_driver_app_transaction::MEM_RD;
      // Selecting the exception kind to AUTO_CORRUPT_LCRC to corrupt the LCRC
             transaction_error_kind ==
svt_pcie_driver_app_transaction_exception::AUTO_CORRUPT_LCRC;
      // Selecting the number of exceptions to 1
             num_exceptions == 1;
             address == 'h0000_0000;
             length == 2;
             first_dw_be == 4'b1111;
             last_dw_be == 4'b1111;
             traffic_class == 0;
             address_translation == 2'b00;
             block == 1;
      };

   // Start the UVM sequence
      mem_req_exc_seq.start(env.root.driver_transaction_seqr[0]);

      wait_until_driver_idle_service_seq = new();
   // Check for the driver is idle before ending test.
      wait_until_driver_idle_service_seq.start(env.root.driver_seqr[0]); // End of Test

   // Drop UVM objection
      phase.drop_objection(this);
      `uvm_info("run_phase", "Exiting...", UVM_LOW)
   endtask

endclass //pseudo_random_serdes_test
```

17.3.5.2 Exception via Component Factory

In the previous section, we discussed the injection of exception directly on the stimulus stream. While this option provides a fine-grain control that allows you to pick and choose transactions for error/exception injection, it doesn't allow a purely random selection of transactions for error/exception injection. For tests cases that target a purely random selection of tests, the UVM VIP has a factory object defined inside the driver application, which can be initialized with a user-defined (constraints) to enable not just a pure random selection but also a random pattern of errors.

Example 17-12

```
// Define an exception class to define error weights of desired choice
class my_driver_app_transaction_exception_list extends
svt_pcie_driver_app_transaction_exception_list;

  svt_pcie_driver_app_transaction_exception xact_exc = new("xact_exc");

  function new(string name = "dl_tlp_exception_list_via_factory",
svt_pcie_driver_app_transaction_exception xact_exc = null);
    super.new(name,xact_exc);
    xact_exc = this.xact_exc;
  // Set all error weights to 0
    xact_exc.set_constraint_weights(0);
  // Setting NO_ERROR weight to 4 and AUTO_TX_CORRUPT weight to 1
  // 20% error (corrupted LCRC) and 80% no error packets
    xact_exc.NO_ERROR_wt = 4;
    xact_exc.AUTO_CORRUPT_LCRC_wt = 1;
  // Set the exception factory of the exception list class
    this.randomized_exception = xact_exc;
  endfunction
endclass

// UVM Test
class pseudo_random_serdes_test extends uvm_test;

 /** UVM Component Utility macro */
  `uvm_component_utils(pseudo_random_serdes_test)

 // Declare an exception list object for error injection.
  svt_pcie_driver_app_transaction_exception_list drv_app_exc_list_factory;


 /** Class Constructor */
  function new(string name = "pseudo_random_serdes_test", uvm_component parent=null);
    super.new(name,parent);
  endfunction: new

  virtual function void build_phase(uvm_phase phase);
    `uvm_info("build_phase", "Entered...", UVM_LOW)
    super.build_phase(phase);
```

// Create an object of drv_app_exc_list_factory

```
drv_app_exc_list_factory = new("drv_app_exc_list_factory");
```

// Setting a limit on the maximum possible exceptions attached on a single transaction.

// Setting to 1 will result in a random behavior of no error or 1 error on the transaction.

```
drv_app_exc_list_factory.max_num_exceptions = 1;
```

// Weight distribution for a short (less than or equal to num_exceptions/2) list. Set to 1 so that the driver application generates a single error.

```
drv_app_exc_list_factory.EXCEPTION_LIST_SINGLE_wt = 1;
```

// Weight distribution for an empty list. Set to 0 since the test intends to generate a single error.

```
drv_app_exc_list_factory.EXCEPTION_LIST_EMPTY_wt = 0;
```

// Weight distribution for a short (less than or equal to num_exceptions/2) list. Set to 0 since the test intends to generate a single error.

```
drv_app_exc_list_factory.EXCEPTION_LIST_SHORT_wt = 0;
```

// Weight distribution for a short (greater than num_exceptions/2) list. Set to 0 since the test intends to generate a single error.

```
drv_app_exc_list_factory.EXCEPTION_LIST_LONG_wt = 0;
```

// Set the factory exception list inside the driver application using UVM config DB

```
    uvm_config_db#(svt_pcie_driver_app_transaction_exception_list)::set(this,
"env.endpoint.driver0", "driver_xact_exception_list", drv_app_exc_list_factory);

    `uvm_info("build_phase", "Exiting...", UVM_LOW)
  endfunction: build_phase

   virtual task run_phase(uvm_phase phase);
      pcie_device_pseudo_random_sequence p_rand_seq;
      svt_pcie_driver_app_service_wait_until_idle_sequence
wait_until_driver_idle_service_seq;

      `uvm_info("run_phase", "Entered...", UVM_LOW)
```

// Raise UVM objection

```
      phase.raise_objection(this);

      …
```

// Assumptions:

// The UVM test has an instance of UVM environment named 'env'

// The UVM environment has an instance of the PCIe device agent named 'root'

// VIP's DL is enabled.

// LTSSM is in L0

```
      p_rand_seq = new();
      p_rand_seq.cfg = env.endpoint_cfg;
      p_rand_seq.write_transaction.cfg = env.endpoint_cfg;
      p_rand_seq.read_transaction.cfg = env.endpoint_cfg;

      for (int i=0; i<5 ; i++) begin
```

// Randomize the UVM sequence

```
            p_rand_seq.randomize with {
               write_transaction.address == 'h0000_0000;
               write_transaction.length == 2;
               write_transaction.first_dw_be == 4'b1111;
               write_transaction.last_dw_be == 4'b1111;
               write_transaction.traffic_class == 0;
               write_transaction.address_translation == 2'b00;
               foreach(write_transaction.payload[i])
                  write_transaction.payload[i] == 'hbaadbaad;
            };

      // Start the UVM sequence
            p_rand_seq.start(env.root.driver_transaction_seqr[0]);
         end

         wait_until_driver_idle_service_seq = new();
     // Check for the driver is idle before ending test.
         wait_until_driver_idle_service_seq.start(env.root.driver_seqr[0]); // End of Test

     // Drop UVM objection
         phase.drop_objection(this);
         `uvm_info("run_phase", "Exiting...", UVM_LOW)
      endtask

   endclass //pseudo_random_serdes_test
```

Example 17-12 illustrates the injection of a LCRC corruption error via the component factory. In the example, the weight distribution in the exception list object is set to randomly introduce the LCRC corruption on 20% of the transactions. Note how the specification of the error is very abstract compared to the specification used in the previous section. Also, the weights can be modified to specify a mix of different kinds of error, for example, 80 percent no error, 5 percent corrupt LCRC error, 5 percent duplicate sequence number error, 5 percent corrupt length error and 5 percent corrupt transaction type error.

The exception list also provides parameters to control the number of exceptions being injected. For instance, `svt_pcie_driver_app_transaction_exception_list::max_num_exceptions` can be used to specify the maximum possible exceptions. This parameter is a non-rand type integer variable which constrains the rand type integer variable `svt_pcie_driver_app_transaction_exception_list::num_exceptions`. When exceptions are introduced using the factory, the driver randomizes the factory exception list object and attaches it to every transaction being transmitted by the driver. The randomization can yield a value anywhere between 0 and `svt_pcie_driver_app_transaction_exception_list::max_num_exceptions` in `svt_pcie_driver_app_transaction_exception_list::num_exceptions`. Once the number of exceptions is resolved by the driver, it then decides the kinds of errors as described above.

The exception list also provides additional parameters that weigh the resolution of `svt_pcie_driver_app_transaction_exception_list::num_exceptions`. Table 17-4 provides the details of these parameters.

Table 17-4 Weight Distribution Parameters

Variable name	Description
`EXCPETION_LIST_SINGLE_wt`	Sets the weight distribution for the generation of `svt_pcie_driver_app_transaction_exception_list::num_exceptions` to a value of 1.

Table 17-4 Weight Distribution Parameters

Variable name	Description
EXCEPTION_LIST_EMPTY_wt	Sets the weight distribution for the generation of svt_pcie_driver_app_transaction_exception_list::num_exceptions to a value of 0.
EXCEPTION_LIST_SHORT_wt	Sets the weight distribution for the generation of num_exceptions to a value less than or equal to svt_pcie_driver_app_transaction_exception_list::max_num_exceptions/2.
EXCEPTION_LIST_LONG_wt	Sets the weight distribution for the generation of num_exceptions to a value greater than svt_pcie_driver_app_transaction_exception_list::max_num_exceptions/2.

17.4 Status

The driver application provides a set of state values representing its status at any given time in the test simulation. These state values are encapsulated within `svt_pcie_driver_app_status` class. The driver application class has an object of type `svt_pcie_driver_application_status` instanced as status.

For more details, such as members, class inheritance UML, list of tasks, list of constraints and so on, see the HTML class description of `svt_pcie_device_configuration` at:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/class_svt_pcie_driver_app_status.html

There are two ways in which a test can access the status object of the driver application.

1. Directly accessing object `svt_pcie_driver_app::status` as illustrated in Example 17-13.
2. Using `svt_pcie_device_agent::get_device_status()` function to retrieve the status of the agent and refer to `svt_pcie_device_status::driver_status[0]` object as shown in Example 17-14.

Note
The `get_device_status` function takes an input which is passed by reference.

Example 17-13

```
class pcie_traffic_test extends uvm_test;
   `uvm_component_utils(pcie_traffic_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_driver_app_service_wait_until_idle_sequence drv_is_idle_serv_seq;

      super.run_phase(phase);
      …
   // Assumptions:
   // The UVM test has an instance of UVM environment named 'env'
   // The UVM environment has an instance of the PCIe device agent named 'root'
   // VIP's DL is enabled.
   // LTSSM gets operational at 2.5G and the model issued a number of requests.
```

```
    // A number of requests were queued on the driver app.
// About to end the test. Make sure the driver app is idle.
      drv_is_idle_serv_seq = new();
      drv_app_serv_seq.start(env.root.virt_seqr.driver_seqr[0]);

// Print the number of transaction sent, completed and completions received by the driver
      `uvm_info("run_phase", $sformatf("Num trans sent = %0d",
env.root.driver[0].status.num_transaction_sent))
      `uvm_info("run_phase", $sformatf("Num trans completed = %0d",
env.root.driver[0].status.num_transaction_completed))
      `uvm_info("run_phase", $sformatf("Num completions received = %0d",
env.root.driver[0].status.num_completion_tlps_received))
   endtask
endclass
```

Example 17-14

```
class pcie_traffic_test extends uvm_test;
   `uvm_component_utils(pcie_traffic_test)
   …
   task run_phase (uvm_phase phase);
      svt_pcie_driver_app_service_wait_until_idle_sequence drv_is_idle_serv_seq;

      super.run_phase(phase);
      …
   // Assumptions:
   // The UVM test has an instance of UVM environment named 'env'
   // The UVM environment has an instance of the PCIe device agent named 'root'
   // VIP's DL is enabled.
   // LTSSM gets operational at 2.5G and the model issued a number of requests.
   // A number of requests were queued on the driver app.


   // About to end the test. Make sure the driver app is idle.
      drv_is_idle_serv_seq = new();
      drv_app_serv_seq.start(env.root.virt_seqr.driver_seqr[0]);

      env.root.get_device_status(root_dev_status);

   // Print the number of transaction sent, completed and completions received by the driver
      `uvm_info("run_phase", $sformatf("Num trans sent =
%0d",root_device_status.driver_status[0].num_transaction_sent))
      `uvm_info("run_phase", $sformatf("Num trans completed = %0d",
root_device_status.driver_status[0].num_transaction_completed))
      `uvm_info("run_phase", $sformatf("Num completions received = %0d",
root_device_status.driver_status[0].num_completion_tlps_received))
   endtask
endclas
```

18 Functional Coverage

This chapter discusses the following topics:

- Introduction
- Usage Notes

Attention
The legacy coverage model has been moved to Appendix “Functional Coverage” on page 470. For more details on the mapped covergroups, see “Mapping Legacy Covergroups to Corresponding New Covergroups” on page 504.

18.1 Introduction

This Section provides the overview of PCIe functional coverage features, usage mechanism, coverage generation and coverage back annotation process.

18.1.1 Key Features

Table 18-1 lists the Gen4 protocol coverage features in each layer of the coverage model.

Table 18-1 Protocol Coverage Features - Gen4

Coverage	Description
TL	Covers the fields in prefixes and headers in the transmitted and received TLPS.
DL	Covers different transmitted and received DLLPs. For example, ACK, NAK, NOP, VENDOR DLLP and Flow control DLLPs.
PL	Covers the features in the PL layer such as Ordered sets (for example, SKP, Control SKP, TS OS, EIOS, FTS), link width, speed negotiations, 8b/10b K codes, and lane skew ranges.
LTSSM	Covers the LTSSM state transitions along with the transition reasons, path coverage such as link width negotiations, initial bring up and partial Equalization scenarios and transitional coverage of low power states.
PIPE	Covers signals on PIPE interface.

Table 18-2 lists the Gen5 protocol coverage features in each layer of the coverage model.

Table 18-2 Protocol Coverage Features - Gen5

Coverage	Description
PL	Covers new ordered sets (`MODIFIED_TS_OS`), ordered set updates (`TS1, TS2,SKP,EIEOS`), Precoding for Error protection, Redo Equalization scenarios, extended lane skew ranges, Link speed negotiations to Gen5,equalization coefficients, presets at Gen5 and cross combinational coverage with Gen5 speed.
LTSSM	Covers new Loop back equalization architecture for multi-lane Bert testing support, initial link bring up scenarios at Gen5, L0s Tx and Rx transitions at Gen5, L1 substates at Gen5.
PIPE	Covers the Equalization signals at Gen5 (for example, `preset_index`, `tx_deemphasis,Figure of merit, Feedback_direction_change, rx_eval_equalization, Local LF,Local FS,LF, FS`), `Rx_Status`, `Rx_Polarity`, and `Tx_Block_Align`, `PowerDown` signals at Gen5.

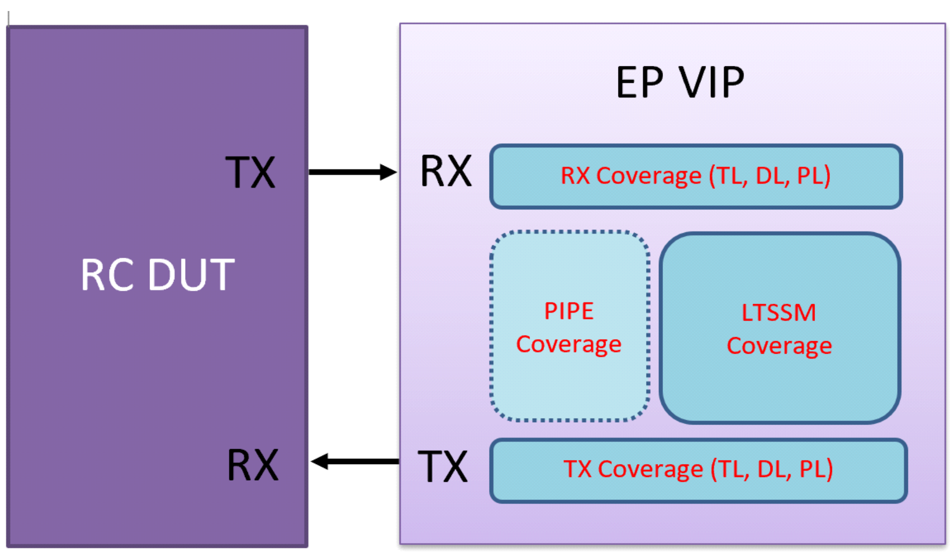
18.1.2 Coverage Model Interpretation

The coverage is collected by VIP instances based on DUT type as shown in Figure 18-1.

For example, if the DUT is Root Complex, then in the coverage report under the `ltssm*_ep_cg`, LTSSM state Recovery Equalization phase 0 is registered as hit after EP VIP enters this state, even though Recovery Equalization phase 0 is not a legal state for Root Complex DUT LTSSM.

It can be inferred that, if all the VIP LTSSM states are covered, then DUT LTSSM must have gone through its respective states.

Figure 18-1 Function Coverage Block Diagram



18.1.3 Covergroup Naming Conventions

Following are the naming conventions followed for covergroups:

- Covergroup names beginning with a common prefix. For example, for all the LTSSM covergroups, *ltssm_* is the prefix.
- Covergroups with *_ep_* and *_rc_* in the name are generated by EP VIP and RC VIP respectively.

- A Covergroup without *_rc_* or *_ep_* in the name are applicable for both RC and EP VIPs.
- Covergroup names with *_tx_* and *_rx_* denote the direction. The direction is with respect to VIP—that is, *_rx_* is the coverage based on observations at the RX path of the VIP. For example, *pl_rx_skp_os_lane0_cg* covergroup covers SKP ordered set received on lane 0 by the VIP.
- TL and DL covergroup names with *_downstream_* and *_upstream_* denote the coverage of downstream and the upstream TLP or DLLPs respectively.

  For more details on covergroups and coverpoints, see "Coverage" tab in the HTML class reference documentation:

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/index.html

18.1.4 DUT Type and Coverage Applicability

Covergroups applicability based on DUT type is as follows:

- TL and DL covergroup names with **_downstream_*cg* denotes the downstream travelling DLLPs or TLPs which are transmitted by Root Complex and received at Endpoint VIP.
  - For EP DUT in coverage report, they are identified by <*Root VIP instance*>*.*_tx_*.*_downstream_*_cg()*.
  - For RC DUT in coverage report, they are identified by <*Endpoint VIP instance*>*.*_rx_*.*_downstream_*_cg()*.
  - PHY DUT is an interoperability testing between two VIPs based on the interest these covergroups are applicable for assessing whether the PHY is transferring TLPs or DLLPs successfully under Normal state of operation.
- TL and DL covergroup names with **_upstream_*cg* denotes the upstream travelling DLLPs or TLPs which are transmitted by Endpoint and received at Root complex VIP.
  - For EP DUT in coverage report, they are identified by < *Root VIP instance* >*.*_rx_*.*_upstream_*_cg()*.
  - For RC DUT in coverage report, they are identified by < *Endpoint VIP instance* >*.*_tx_*.*_upstream_*_cg()*.
  - PHY DUT is an interoperability testing between two VIPs based on the interest these covergroups are applicable for assessing whether the PHY is transferring TLPs or DLLPs successfully under Normal state of operation.
- The covergroup names with a suffix of **_rc_cg* are applicable for EP and PHY DUTs as they are created based on RC VIP.
- The covergroup names with a suffix of **_ep_cg* are applicable for RC and PHY DUTs as they are created based on EP VIP.

18.1.5 Coverage Hierarchical Verification Plans

The PCIe VIP provides Verdi compatible top-level HVP for Root Complex, Endpoint and PHY DUT. You can create your own HVP and use the PCIe VIP HVP as subplans.

Top-level HVPs are available at the following locations:

- RC DUT

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/dut_rc/pcie_dut_rc_fc_toplevel_plan.hvp

- EP DUT

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/dut_ep/pcie_dut_ep_fc_toplevel_plan.hvp
- PHY DUT

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/dut_phy/pcie_dut_phy_fc_toplevel_plan.hvp

The top-level HVP instantiates respective protocol layer coverage subplans based on the DUT.

For example, RC DUT top-level plans includes the subplans with the following names.

- *pcie_dut_rc_pci_express_base_spec_r3_1_data_link_layer_fc_subplan.hvp*
- *pcie_dut_rc_pci_express_base_spec_r3_1_physical_layer_fc_subplan.hvp*
- *pcie_dut_rc_pci_express_base_spec_r3_1_power_management_fc_subplan.hvp*
- *pcie_dut_rc_pci_express_base_spec_r3_1_system_architecture_fc_subplan.hvp*
- *pcie_dut_rc_pci_express_base_spec_r3_1_transaction_layer_fc_subplan.hvp*
- *pcie_dut_rc_pipe_ver_4_4_subplan.hvp*

The Filters (for example, *lane_number_filter.hvpmod*, excludes unsupported lanes based on lane count supported by the DUT) and HVP source code of each subplan is available at the following location: *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/*

- For Gen4 Coverage with RC DUT/EP DUT, following filters must be loaded along with the HVP plan.
  - *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/lane_number_filter.hvpmod*
  - *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/pipe_filter.hvpmod*
  - *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/gen4_coverage_filter.hvpmod*
- For Gen4 Coverage with PHY DUT, following filters must be loaded based on MAC along with the HVP plan.
  - RC MAC
    - *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/rc_mac_pipe_filter.hvpmod*
    - *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/lane_number_filter.hvpmod*
  - EP MAC
    - *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/ep_mac_pipe_filter.hvpmod*
    - *$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/lane_number_filter.hvpmod*
- For Gen5 Coverage with EP DUT following filters to be loaded along with the HVP plan.

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/EP_DUT_Gen5_Cov_filter.hvpmod

- For Gen5 Coverage with RC DUT following filters to be loaded along with the HVP plan.

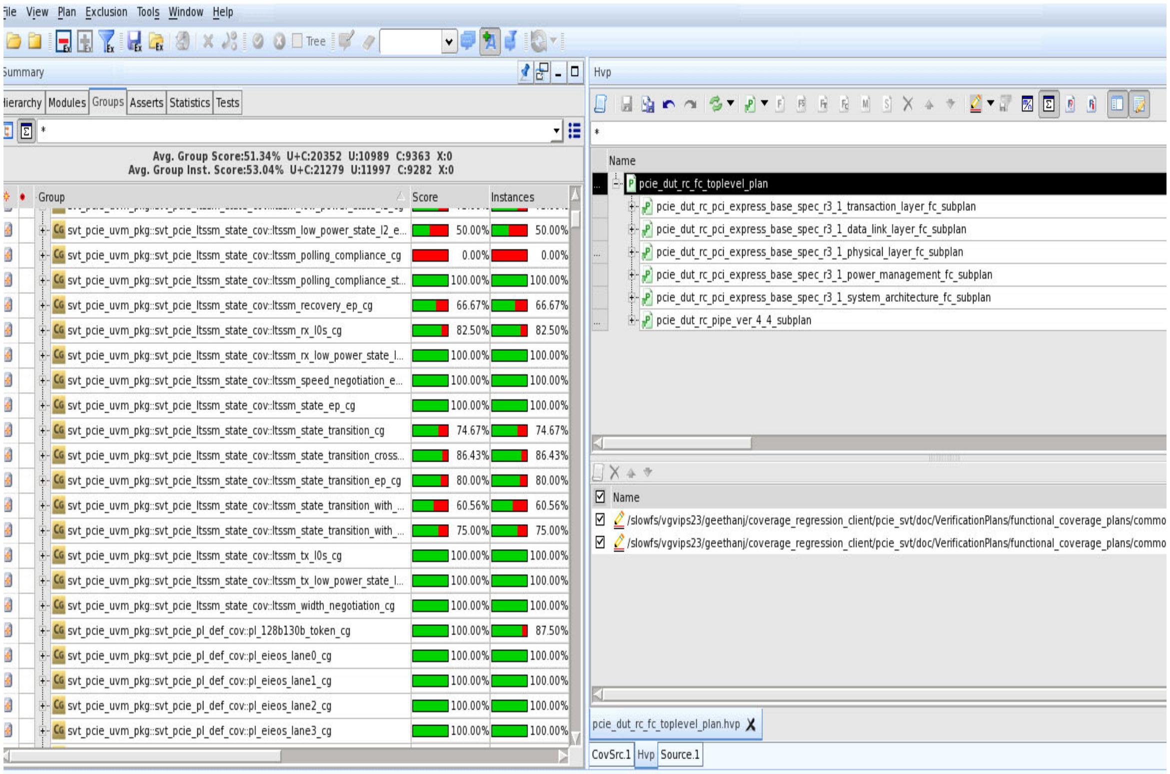
  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/RC_DUT_Gen5_Cov_filter.hvpmod

- For Gen5 Coverage with PHY_DUT following filters should be loaded based on MAC along with the HVP plan.
  - RC MAC
    $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/rc_mac_pipe_filter.hvpmod

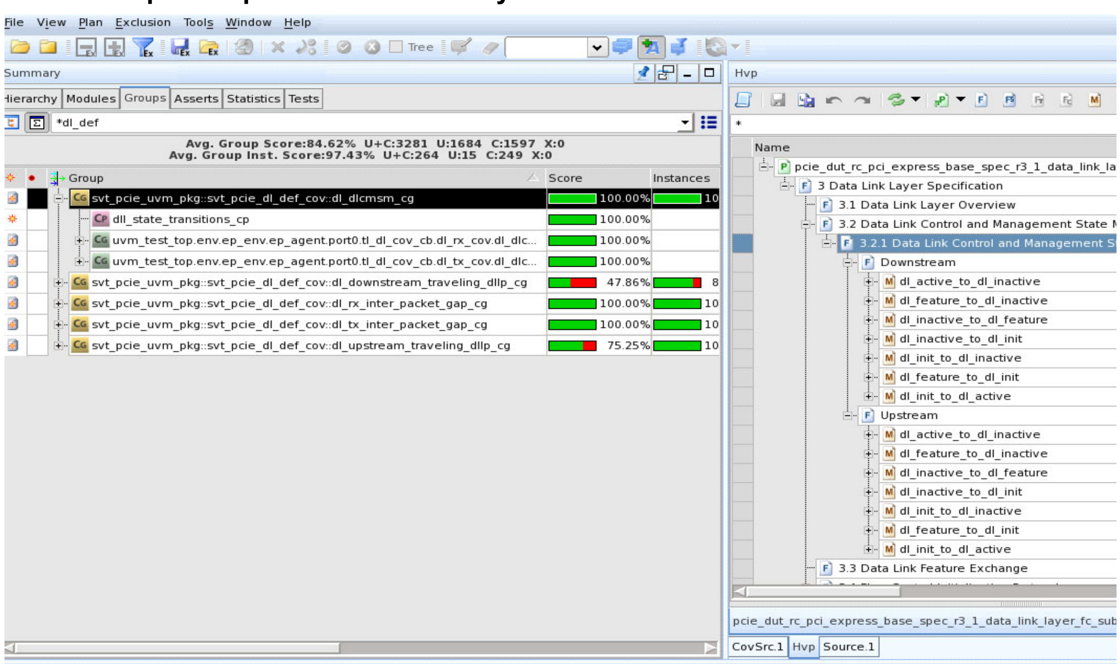
    $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/lane_number_filter.hvpmod

    $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/gen5_coverage_filter.hvpmod
  - EP MAC
    $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/ep_mac_pipe_filter.hvpmod
    $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/lane_number_filter.hvpmod
    $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans/functional_coverage_plans/common/gen5_coverage_filter.hvpmod

Example 18-1 Sample Top Plan



Example 18-2 Sample Subplan of Data Link Layer



Note

The HVP plans provided with the PCIe VIP should be used as-is, because the HVP plan will be periodically updated based on the coverage enhancement.

18.2 Usage Notes

18.2.1 Enabling Function Coverage

To enable the new function coverage, define the macro `SVT_PCIE_ENABLE_COMMON_COV` at compile time and set the `enable_cov` variable in `svt_pcie_configuration` class to a desired value.

The configuration variable `enable_cov` is a 6-bit vector, where each bit corresponds to enabling coverage for subsections listed in the table.

Table 18-3 Coverage Controls

Bit Position	Coverage Control
`enable_cov[0]`	When set to 1'b1, enables PIPE functional coverage.
`enable_cov[1]`	When set to 1'b1, enables functional coverage in PL Layer
`enable_cov[2]`	When set to 1'b1, enables functional coverage in DL Layer.
`enable_cov[3]`	When set to 1'b1, enables functional coverage in TL Layer.
`enable_cov[4]`	When set to 1'b1, enables LTSSM functional coverage.
`enable_cov[5]`	When set to 1'b1, enables 8b/10b symbol functional coverage (only valid for SERDES interface).

Note

- Do not define the macros `SVT_PCIE_INCLUDE_AC_COVERAGE` and `SVT_PCIE_ENABLE_COMMON_COV` macros at the same time. Defining both the macros together at compile time will generate undesirable coverage results.
- The legacy coverage model and `SVT_PCIE_INCLUDE_AC_COVERAGE` macro will be deprecated from 0-2018.12 release.

The Covergroup creation depends on following configuration in addition to `enable_cov`, changing these configurations after initial `build_phase ()` causes incorrect coverage.

- `supported_speed` field in `svt_pcie_pl_configuration`
- `link_width` field in `svt_pcie_pl_configuration`
- `pipe_spec_ver` field in `svt_pcie_pl_configuration`
- `enable_pclk_as_phy_input` field in `svt_pcie_pl_configuration`
- `enable_l1ss_pipe_handshake` field in `svt_pcie_pl_configuration`

Based on DUT type and interface, the function coverage control will change for RC and EP DUT.

	RC DUT or EP DUT	
	PIPE	SERDES
PIPE Coverage Enable - enable_cov[0]	Y	NA
PL Coverage Enable - enable_cov[1]	Y	Y
DL Coverage Enable - enable_cov[2]	Y	Y
TL Coverage Enable - enable_cov[3]	Y	Y

	RC DUT or EP DUT	
LTSSM Coverage Enable - enable_cov[4]	Y	Y
8b/10b K-code & data Coverage - enable_cov[6]	NA	Y

A typical PHY DUT environment will consist of two PCIe VIPs, one on either side of PHY DUT. One VIP will have PIPE Interface connecting to PHY DUT and the other VIP will be connected with Serial interface. Below is the quick reference for function coverage controls for PHY DUT environment.

	PHY DUT	
	VIP with PIPE Interface	VIP with SERDES Interface
PIPE Coverage Enable - enable_cov[0]	Y	NA
PL Coverage Enable - enable_cov[1]	Y	Y
DL Coverage Enable - enable_cov[2]	Y	Y
TL Coverage Enable - enable_cov[3]	Y	Y
LTSSM Coverage Enable - enable_cov[4]	Y	Y
8b/10b K-code & data Coverage - enable_cov[6]	NA	Y

18.2.2 HVP Coverage Back Annotation with PCIe VIP RC DUT

Back annotating coverage with PCIe VIP HVP for RC DUT.

1. Create a top-level HVP.

   A key variable for back annotating the HVP is the path to coverage groups in VDB. The coverage path can be divided into two sections:

   - User environment path - Path to VIP agent in the user environment.
   - Fixed path - Path for covergroup/coverpoint/bin from VIP agent.

     To use the HVP, you must update your path to coverage. For RC DUT HVP, the user environment path can be defined by the `SVT_PCIE_HVP_RC_DUT_VIP_AGENT_PATH` attribute. To set the path for VIP agent, create a top-level HVP plan and instantiate *pcie_dut_rc_fc_toplevel_plan.hvp* as a subplan, and pass the correct path to the HVP attribute.

     ```
     `include
     "<DESIGNWARE_HOME_PATH>/vip/svt/pcie_svt/latest/doc/VerificationPlans/function
     al_coverage_plans/dut_rc/pcie_dut_rc_fc_toplevel_plan.hvp"

     plan Customer_Plan;

      subplan pcie_dut_rc_fc_toplevel_plan #(SVT_PCIE_HVP_RC_DUT_VIP_AGENT_PATH =
     "<VIP_AGENT_PATH>");

     endplan
     ```

2. Back annotating HVP

   HVP can be back annotated in Verdi for analysis in GUI or can be back annotated to HTML.

   a. Command to back annotate to HTML

```
urg -lca -full64 -dir <Coverage_Database>.vdb -plan Customer_Plan.hvp [-mod
lane_number_filter.hvpmod] -report rc_dut_back_annoated
```

Switch Description

`plan`	Top-level user plan
`dir`	Coverage database
`report`	Back annotate HTML files directory, *rc_dut_back_annoated/dashboard.html* will be top-level HTML page.

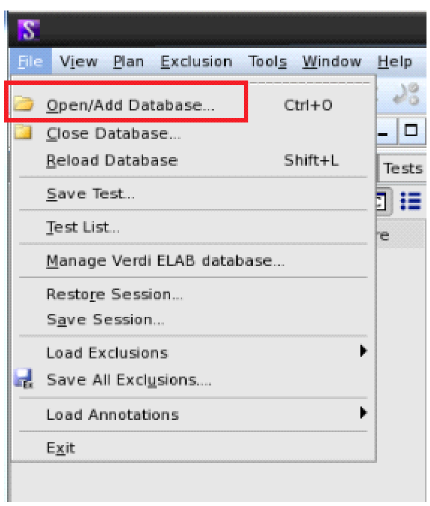
`mod`	Plan modified [Optional]. To load plan modified
	For example, PL coverage for Ordered Set is per lane coverage and HVP plan lists all possible 32 lanes. But if the DUT only supports a maximum of 8 lanes, then with plan modified you can remove lanes from 8–31 from the HVP.
	`filter lane_number_filter;`
	`remove feature where SVT_PCIE_HVP_LANE_NUM > 7;`
	`endfilter`

b. Back annotating with Verdi

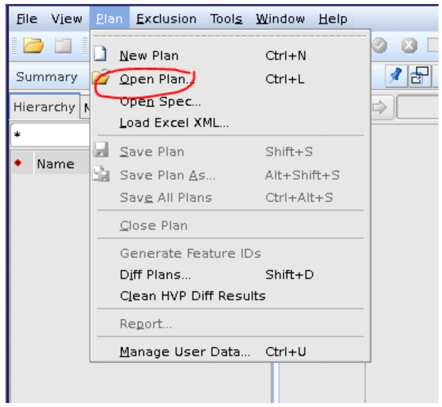
i. To invoke Verdi coverage, you need to add the `-cov` option to the Verdi command line. You can also specify the directory name in the command line while invoking Verdi coverage as shown below:

```
> verdi -cov &
> verdi -cov -covdir <filename>.vdb &
```

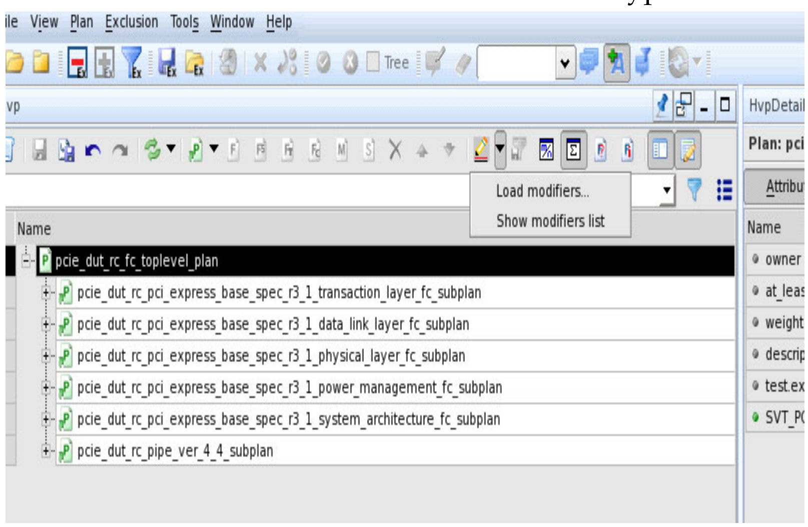
In the Verdi GUI, select **File > Open/Add Database** to load the database.

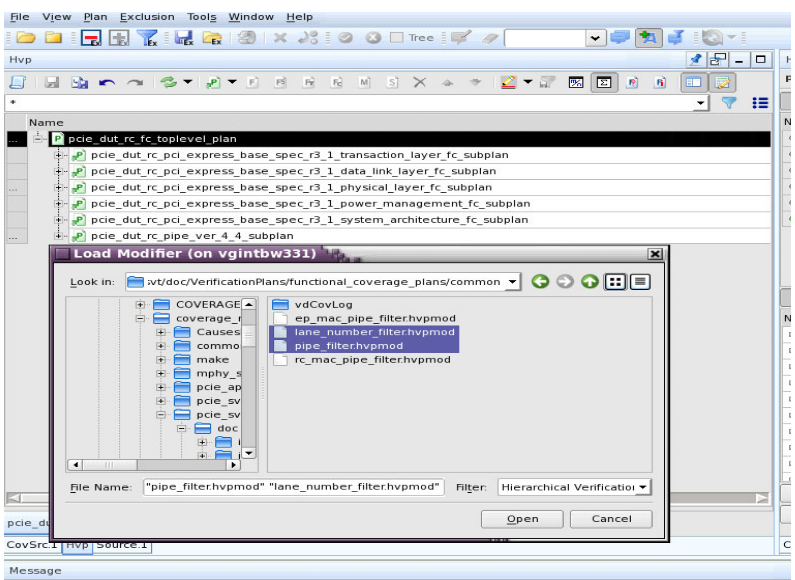


ii. Select **Plan > Open Plan** to load the HVP plan.

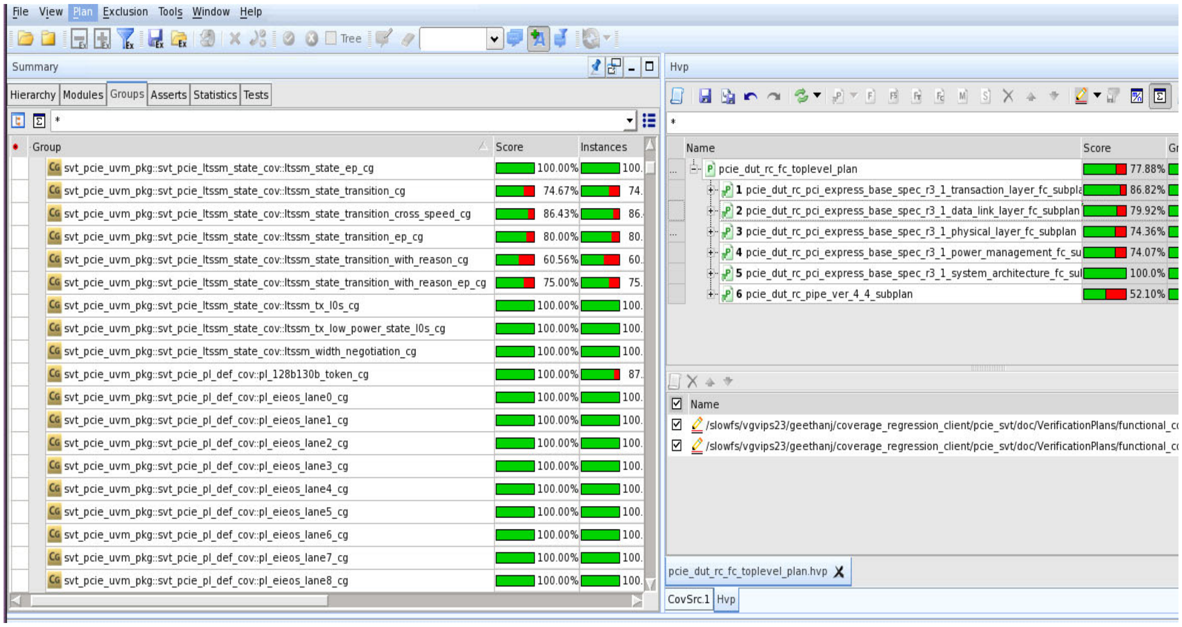


iii. Select **Load modifiers > load the Filter** based on the DUT type.





iv. Opening the plan will automatically back annotate the coverage as per the HVP.



19 Using Callbacks

19.1 Introduction

Callbacks provide a means to examine or modify transactions at various points in the protocol stack. This chapter describes their basic usage, provides some examples, and gives tips for debugging them.

Within a trans ac at ion, you can use the transaction handle to:

- Understand where in the protocol layers a particular transaction is being processed.
- Examine a particular field that was added to a transaction (for example, the Link Layer Sequence Number).
- Collect statistics and functional coverage or provide transactions to a scoreboard.
- Modify a transmitted TLP to cause a particular error to occur in the DUT and then examine the received TLP to verify that the error actually occurred as planned.
- Modify a received transaction to cause the VIP to respond abnormally to that transaction (for example, by injecting an illegal CRC value.)

19.2 How Callbacks Are Used

Callbacks occur at specific places within the VIP, where callback “hooks” have been provided by the VIP. Follow these steps to implement and use a callback:

1. Identify which callback you need to use.
2. Sub-class the appropriate data type and implement the appropriate callback method with a user-defined action each time the callback is made.
3. Create an object of this type and add it to the queue of callback objects on the appropriate VIP instance.

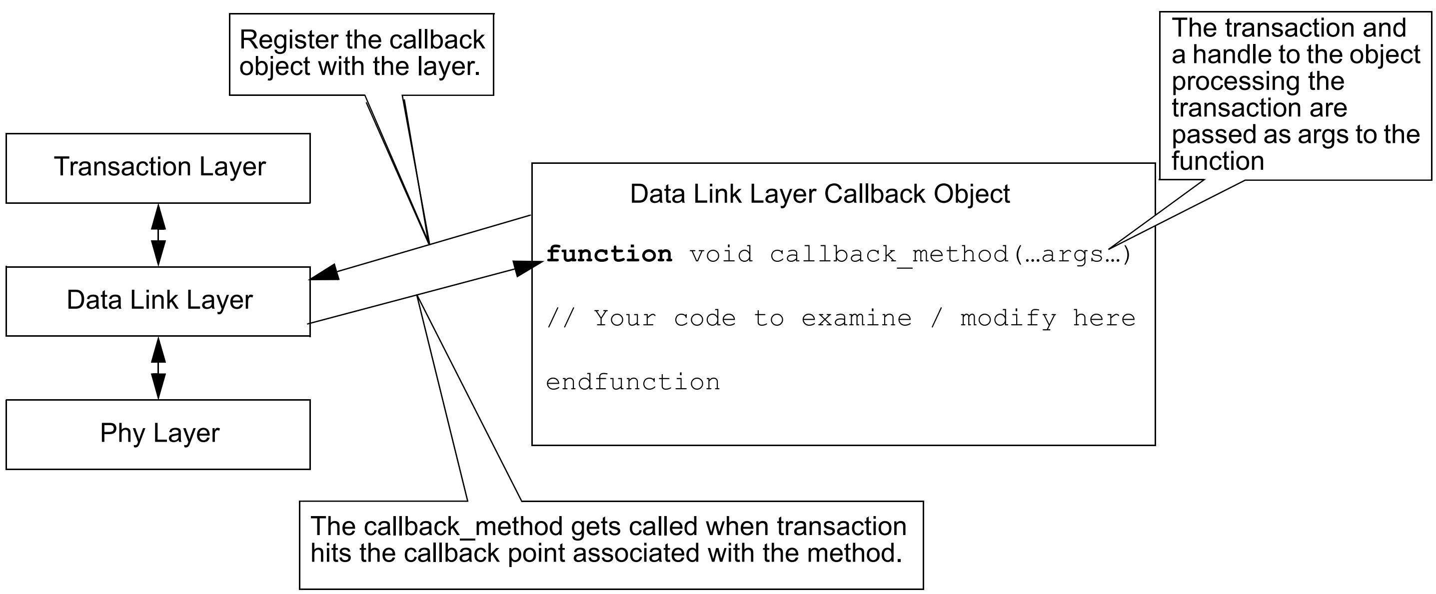
You can request a callback by specifying:

1. What VIP layer you are interested in (for example, the Data Link Layer)
2. Which direction (transmit or receive) and location within that layer you want to get the callback from. (Typically, there is more than one callback point you can specify).
3. The particular object you want registered to receive callbacks.

4. Code within a method of the above object to examine and modify the transaction.

While a test is running, whenever the callback you have implemented occurs (that is, when the transaction reaches the point in the protocol flow that causes the callback to occur), your registered callback method will be called. Once your method has finished its processing, it returns from the callback and the protocol processing continues. Figure 19-1 is a diagram of the callback process for a callback that you request in the Data Link Layer.

Figure 19-1 Callback operation for a callback in the Data Link Layer



19.2.1 Other Uses for Callbacks

Although callbacks are typically used to examine and modify the transactions as they move through the protocol stack, there are additional uses for callbacks:

- Examining if a previous callback or error injection has occurred on the transaction
- Causing an exception to occur on the associated transaction

19.2.2 Callback Hints

Callbacks are a tool that should be used conservatively:

- Limit callback modifications to one per callback.
- If multiple modifications are truly needed, use multiple callbacks.
- If multiple callbacks are used, it is important to understand the order in which the callbacks will be called.

19.2.3 When *Not* to Use a Callback

Although callbacks are a flexible mechanism, UVM analysis ports are preferred for statistics, coverage, scoreboarding, and so on. However UVM ports do not allow modification of the transaction within the caller, so use callbacks if you want to modify the transaction within the caller.

The VIP has many mechanisms for examining transactions, altering frame data, timing, and so on that might be easier to use. Here are some examples:

- When an exception already exists for the modification you have in mind, use the exception. It will not only inject the error, it will also verify that the response is correct for that error, and automatically recover. If a control already exists in the configuration or via a service request, use that control instead of a callback.
- Statistics – There already are statistics available that count many protocol items. There is no need to rewrite these.
- Delays of packets – Callbacks must be called in "zero time". They are coded as functions to enforce this.
- Scoreboards – Do not use a callback to send data to your scoreboard if there is an available analysis port at the same location.

19.3 Detailed Usage

This section describes how callbacks are used in several example testcases. The first example is a working testcase, although it is missing many features that you would normally use. The second example expands on those additional useful mechanisms.

19.3.1 A Basic Testcase

These are the main mechanisms, classes and methods used in the basic testcase example:

- Test Case Class – Each method is a different UVM phase:
  - new() constructor
  - build_phase() – Set up a test-specific configuration.
  - end_of_elaboration_phase() – Create and register the callback object
  - run_phase() – A placeholder. Nothing to do here.
- Callback Class – Each method is a specific callback from the given layer:
  - new() constructor
  - pre_tlp_framed_out_put() – Callback called just prior to framing the TLP.

Example 19-1 Code for a basic testcase

```
typedef class tx_dl_tlp_callback; // Forward declaration

// The testcase class:
class dl_tlp_basic_callback_test extends basic ;
   // Factory registration
   `uvm_component_utils( dl_tlp_basic_callback_test )

   // Callback instance:
```

```
   tx_dl_tlp_callback dl_tlp_cb;

   // Constructor:
   function new(string name="dl_tlp_basic_callback_test",uvm_component parent=null);
      super.new(name,parent);
   endfunction : new

   // Configure/build various components:
   virtual function void build_phase(uvm_phase phase);
      super.build_phase(phase);

      // Set test-specific cfg values:
      cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_width_values(4);
      cust_cfg.endpoint_cfg.pcie_cfg.pl_cfg.set_link_width_values(4);

      //Register the cfg with the registry so that it will be picked up by the env.
      svt_config_object_db#(pcie_device_system_configuration)::set(this,  , {"env",
         ".", "cfg"}, cust_cfg);
   endfunction : build_phase

   function void end_of_elaboration_phase(uvm_phase phase);
      super.end_of_elaboration_phase(phase);

      // Create a callback object instance:
      dl_tlp_cb = tx_dl_tlp_callback::type_id::create("dl_tlp_cb");

      // Register the callback object with the DL component:
      uvm_callbacks#(svt_pcie_dl,svt_pcie_dl_callback)::add(env.endpoint.port.dl,
         dl_tlp_cb) ;
   endfunction : end_of_elaboration_phase

   task run_phase( uvm_phase phase );
      // In this case, there is not much to do in this phase
      `uvm_info(get_full_name(), "Running test dl_tlp_basic_callback_test .\n",
         UVM_LOW);
   endtask : run_phase

endclass : dl_tlp_basic_callback_test

// This is the callback class that is instantiated in the test class above:
class tx_dl_tlp_callback extends svt_pcie_dl_callback;

   // Factory registration:
   `uvm_object_utils(tx_dl_tlp_callback)

   // Error counter - static to allow it to act globally across all callback instances:
   static int error_count = 0;

   function new(string name = "tx_dl_tlp_callback");
      super.new(name);
   endfunction : new

   // pre_tlp_framed_out_put: this is the actual callback function. It will be
   // called every time a DL/TLP is ready to be framed for transmission:
```

```
    virtual function void pre_tlp_framed_out_put( svt_pcie_dl dl,
            svt_pcie_dl_tlp transaction, ref bit drop);

      `uvm_info(get_full_name(),$sformatf("\nA callback prior to transmitting TLP.
        Current TC=%0d and ECRC=0x%x, error count=%0d.\n",
        transaction.tlp.traffic_class, transaction.tlp.ecrc, error_count), UVM_LOW);

      if(error_count < 1 ) begin    // Just corrupt one TLP
        // Force a new traffic class field. Note that the traffic class (since it's in
        // the TLP header) is not directly a member of the transaction, but of the
        // tlp object encapsulated by the DL/TLP object.  The only directly
        // modifiable member of the svt_pcie_dl_tlp_transaction class is the sequence
        // number.
        transaction.tlp.traffic_class = 3;
        error_count++;
      end
    endfunction : pre_tlp_framed_out_put
  endclass : tx_dl_tlp_callback
```

19.4 Advanced Topics in Callbacks

As mentioned above, there are other features you can incorporate (or, depending on what you need to do – must incorporate) in the testcase. This section discusses the concept of *exceptions*, a mechanism to request changes to a transaction that works a bit differently than directly modifying the fields of the transaction. (For example that is how the traffic class is modified in Example 19-1).

19.4.1 Exceptions – a "Delayed" Transaction Modification Request

When you make direct changes to fields in the transaction, they take effect immediately. In many cases, this works fine, but there are situations where you want to hold off on updating the transaction until all the changes are in place. A typical example of this is CRC fields. It makes little sense to calculate a CRC field if another transaction field on which the CRC depends will be changing.

There is a mechanism that allows you to schedule that CRC change (for example) when all of the transactions changes have been incorporated. This delayed transaction update is handled with an object called an *exception.*

An exception is created to request a particular change to the transaction. Once an exception is created, it is then associated with the transaction by placing it on that transaction's *exception list.* Just prior to the callback transaction being placed back into the data flow, the exception is examined and the transaction is updated based on the contents of that exception.

19.4.2 Creating an Exception

As with any object, a handle first must be declared for the exception. Its type will be based on the type of transaction with which it is associated. In the HTML class reference, follow the svt_pcie_dl class to the class attributes and find the exception class svt_pcie_dl_tlp_transaction_exception.

To create an exception, first create a handle of this type and an object associated with it:

Perhaps show the complete callback class?

```
// Exception handle and object
svt_pcie_dl_tlp_exception dl_tlp_exc = new("my_dl_tlp_exc");
```

Next, create the handle and object for the exception list:

```
// Exception list handle and object
dl_tlp_exception_list_via_callback my_dl_tlp_exc_list = new("my_dl_tlp_exc_list");
```

Now set the appropriate fields in the exception to make the request (in this case, a DL/TLP LCRC error):

```
my_dl_tlp_exc.error_kind = svt_pcie_dl_tlp_exception::CORRUPT_LCRC;

// The CORRUPT_LCRC causes the 'corrupted_data' value to be XOR'd with the
// (correctly) calculated LCRC field.
my_dl_tlp_exc.corrupted_data = 32'h0000_0001; // This will invert bit 0 of LCRC.
```

Put the exception into the exception list:

```
my_dl_tlp_exc_list.add_exception(my_dl_tlp_exc);
```

Finally, cast the exception list into the transaction:

```
$cast(transaction.tx_exception_list, my_dl_tlp_exc_list.`SVT_DATA_COPY());
```

Now every time the transaction is handled, if there is an exception in the transaction's exception list, it will be evaluated just prior to the transaction being put back into the data flow.

19.4.3 Creating a Factory Exception

In addition to the above callback exceptions, there is another type of exception: Instead of it being associated with user-specified callback code, it occurs automatically each time the callback function is called (even if there is no user-specified callback code). This allows you to generate randomized error injections.

Note that factory exceptions and callbacks are mutually exclusive: If a user modifies the transaction in the callback, the factory exception will not occur.

The code to set up a factory exception is similar to the code in "Creating an Exception" above, with a few minor differences.

First, define a derived exception list class:

```
class dl_tlp_exception_list_via_factory extends
   svt_pcie_dl_tlp_transaction_exception_list;
   svt_pcie_dl_tlp_exception xact_exc = new("xact_exc");
   function new(string name = "dl_tlp_exception_list_via_factory",
         svt_pcie_dl_tlp_exception xact_exc = null);
      super.new(name,xact_exc);
      xact_exc = this.xact_exc;
      xact_exc.NO_ERROR_wt = 0;
      xact_exc.AUTO_CORRUPT_LCRC_wt = 0;
      xact_exc.ILLEGAL_SEQ_NUM_wt = 0;
      xact_exc.DUPLICATE_SEQ_NUM_wt = 0;
      xact_exc.NULLIFIED_TLP_wt = 0;
      xact_exc.NULLIFIED_TLP_GOOD_LCRC_wt = 0;
      xact_exc.NULLIFIED_TLP_AUTO_CORRUPT_LCRC_wt = 0;
      xact_exc.MISSING_START_wt = 0;
      xact_exc.MISSING_END_wt = 0;
      xact_exc.CORRUPT_DISPARITY_wt = 0;
```

```
        xact_exc.CODE_VIOLATION_wt = 0;
        xact_exc.CORRUPT_8G_HEADER_CRC_wt = 0;
        xact_exc.CORRUPT_8G_HEADER_PARITY_wt = 0;
        xact_exc.RETAIN_LCRC_wt = 0;
        xact_exc.RECALC_LCRC_wt = 0;
        xact_exc.FORCE_LCRC_wt = 0;
        xact_exc.CORRUPT_LCRC_wt = 1; // This will cause a corrupt LCRC
        xact_exc.corrupted_data=32'hffff_ffff;  // This value will be XOR'd with the LCRC
        this.randomized_exception = xact_exc;
   endfunction : new
endclass
```

Now, in the testcase, create an exception in the list handle:

```
  rand dl_tlp_exception_list_via_factory dl_tlp_exc_list;
```

In the build_phase of the testcase, create and randomize the exception list object, then set it to the per-layer component configuration database:

```
dl_tlp_exc_list = new("dl_tlp_exc_list");
dl_tlp_exc_list.randomize();
uvm_config_db#(svt_pcie_dl_tlp_exception_list)::set(this,
   "env.endpoint.port0.dl0",
   "tlp_xact_exception_list",
   dl_tlp_exc_list);
```

Whenever the transaction callback is called, (assuming there is no user callback), the transaction will be modified based on the exception above. If there is a user callback, it will be called *after* the exception code has modified the transaction and you then have the option to further modify the transaction.

19.4.4 Error Injection Using Application Layer TX Callbacks

This section shows you how to use Application Layer TX callbacks for error injection.

19.4.4.1 Basic Target Application Completion Callbacks

Whenever a target has built a completion TLP for transmission (to the TL layer, and ultimately to the remote device) the model calls a completion callback (class: svt_pcie_target_app_callback , method: pre_tx_tlp_put()). The testbench can use this callback by creating an object of a derived class and using uvm_callbacks() and add() to include that callback object in the list of callback objects. This, of course, is standard UVM. Some minor additions are available.

19.4.4.2 Methods to Modify the Transaction

There are three ways to modifity the transaction:

- Simple Modifications.
- Exceptions
- Error Injection Exceptions

19.4.4.2.1 Simple Modifications

When a callback is called, it is passed a TLP object (of type *svt_pcie_tlp*) transaction that contains various fields that can be modified in several ways; the transaction can be modified directly, for example:

```
virtual function void pre_tx_tlp_put(svt_pcie_target_app target_app,
                                     svt_pcie_tlp transaction,
                                     ref bit drop);
    transaction.ep = 1;// Set the Poison Bit in the TLP
endfunction  // pre_tx_tlp_put()
```

In the example, you simply set the value of the TLP poison bit (ep) to 1. The callback will hand the TLP back to the protocol stack , and that TLP's Poison bit will be set.

Note that there is not an explicit Error Injection code being added. However , an implicit virtual EI code *is* added: *USER_MODIFIED_TLP*. This code does two main things:

1. Marks the TLP as having been modified, so later callbacks can know.
2. Keeps any later "automatic" error injections from occurring.

Note that although you should not need to add USER_MODIFIED_TLP manually, it will be placed on the TLP automatically when the TLP has been modified in a callback. (As you will see below, there is an analogous EI code *MALFORMED_TLP* that will be added to a TLP that is explicitly Malformed.)

19.4.4.2.2 Exceptions

Another way the transaction can be modified is via an *exception* – essentially a delayed update to that transaction, for example:

```
virtual function void pre_tx_tlp_put(svt_pcie_target_app target_app,
                                     svt_pcie_tlp transaction,
                                     ref bit drop);
  svt_pcie_tlp_exception_list my_tlp_exc_list;    // Exception list
  svt_pcie_tlp_transaction_exception my_tlp_exc;  // Exception obj
  my_tlp_exc_list = new("my_tlp_exc_list");
  my_tlp_exc = new("my_tlp_exc");
  my_tlp_exc.error_kind =    // Set the exc type
      svt_pcie_tlp_transaction_exception::CORRUPT_ECRC;
  my_tlp_exc.corrupted_data = 32'hffff_ffff; // XOR with ECRC
  my_tlp_exc_list.add_exception(my_tlp_exc);  // Add exc to list
  $cast(transaction.exception_list,
        my_tlp_exc_list.`SVT_DATA_COPY() );   // Add exc list to TLP
endfunction  // pre_tx_tlp_put()
```

In the previous example, the ECRC will be corrupted (see the documentation for details, but essentially the ECRC will be calculated, and the XOR'd with the provided *corrupted_data* (i.e. 32'hffff_ffff). However, this does not occur immediately. Since this is a calculation that is done on the entire TLP, you need to ensure that all the fields that you may intend to change have been changed prior to the ECRC calculation. Using an exception allows you to do this – an exception is applied to the TLP until that TLP is actually getting *pack*'ed, just prior to its being handed back to the protocol stack.

As previous stated, once the exception is applied and the TLP modified, it will be tagged with the USER_MODIFIED_TLP EI code.

Error Injection Exceptions

In addition to modifying the TLP contents directly, an exception can be used to propagate an *Error Injection* (EI) to the layers below it. For example, although the Application layer isn't able to modify the LCRC of a TLP to be transmitted, you can request via an EI that the LCRC be corrupted:

```
virtual function void pre_tx_tlp_put(svt_pcie_target_app target_app,
                                     svt_pcie_tlp transaction,
                                     ref bit drop);
  svt_pcie_tlp_exception_list my_tlp_exc_list;    // Exception list
  svt_pcie_tlp_transaction_exception my_tlp_exc;  // Exception obj
  my_tlp_exc_list = new("my_tlp_exc_list");
  my_tlp_exc = new("my_tlp_exc");
  my_tlp_exc.error_kind =    // Set the exc type
      svt_pcie_tlp_transaction_exception::AUTO_TX_CORRUPT_LCRC;
  my_tlp_exc_list.add_exception(my_tlp_exc);  // Add exc to list
  $cast(transaction.exception_list,
        my_tlp_exc_list.`SVT_DATA_COPY() );   // Add exc list to TLP
endfunction  // pre_tx_tlp_put()
```

The previous example provides the exception, which will be 'translated' to an Error Injection to be handed down the protocol stack. Once it goes into the DataLink Layer (where the LCRC is calculated), then the LCRC is corrupted (as instructed above).

In addition, since the prefix to the exception is ***AUTO_***, this implies that not only will the error injection occur, but that the VIP will automatically do the following:

- Determine if the LCRC actually was recognized correctly by the remote device.
- Recover from the error, and retransmit any required TLPs.
- Suppress any error messages associated with the above.

Note: once a TLP has an EI Exception attached to it, you should not attempt to modify in any other way. Error injections work due to a controlled injection of the error – if multiple errors are attempted simultaneously, the EI will generally will fail in a typically difficult-to-debug way!

Unlike the previous examples note that since the user has *not* changed the TLP (adding the EI request doesn't count as a change to the actual TLP header/data), only the actual EI is attached; the USER_MODIFIED_TLP is **not.**

19.4.4.3 Malformed and Nullified TLPs

If you intend to create a TLP that is *Malformed* (see PCIe spec for details), it is up to the testbench to inform the VIP of that fact. This is done simply via the transactions *set_malformed(value)* method. For example:

```
virtual function void pre_tx_tlp_put(svt_pcie_target_app target_app,
                                     svt_pcie_tlp transaction,
                                     ref bit drop);
 … various manipulations on TLP …// Corrupt the TLP
   // Set TLP to be treated as Malformed
   transaction.set_malformed(1);
endfunction  // pre_tx_tlp_put()
```

Note that if an Error Injection is set on a TLP marked as Malformed, that Error Injection will be canceled (for reasons given above – the requirement of a controlled Error Injection has been broken.) Note also that in the TL layer, credits are not counted for a Malformed transaction – as it is assumed that (being Malformed) the receiver will neither recognize nor count credits for the associated transaction.

Recall previously that we added USER_MODIFIED_TLP as a virtual EI code to TLPs that a user has modified. In this (Malformed) case, the virtual EI code MALFORMED_TLP will be added instead. It has the same basic effect to remind lower layers that this TLP has been modified; in addition it also tells those same lower layers (including callbacks in those layers) that this has been modified so as to be Malformed.

Note

If the transaction did not already have an error injected, then the TL would have consumed credits as appropriate. If the TLP is malformed or nullified via callback (using exceptions), then credit consumption by the VIP cannot be changed. Similarly, a TL layer exception cannot be canceled or changed via this callback.

19.4.5 A More Comprehensive Example

In addition to the Test Case and Callback classes used in Example 19-1, three other classes are added in Example 19-2:

- Exception Classes – Each transaction type has its own exception class, which contains objects of its exception class:
  - new() constructor
  - Various per-transaction fields to set up the exception
- Exception List Classes – Each transaction has a class for its exception list
- Error Handler – Extended from uvm_report_catcher. It has several important methods:
  - new() constructor
  - pattern_match() – Matches the actual error string with the “expected” string
  - catch() – Called upon an error message being potentially displayed. You can filter with this.

Example 19-2 Code for a more comprehensive testcase

```
// Forward declarations
typedef class tlp_exc_list_via_callback;
typedef class dl_tlp_exception_list_via_callback;
typedef class dl_tlp_err_catcher;
typedef class tx_dl_tlp_callback;

// The testcase class:
class dl_tlp_example_callback extends basic ;
   // Factory registration:
   `uvm_component_utils( dl_tlp_example_callback )
   // Callback instance:
   tx_dl_tlp_callback dl_tlp_cb;
   // Exception list class instance:
   dl_tlp_exception_list_via_callback dl_tlp_exc_list;
   // Error catcher:
   dl_tlp_err_catcher err_catcher;
   // Constructor "new":
   function new(string name="tlp_exception_via_callback", uvm_component parent=null);
      super.new(name,parent);
   endfunction : new

   // build_phase: To build various components of class:
   virtual function void build_phase(uvm_phase phase);
      super.build_phase(phase);
      // Load test specific cfg values:
      cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_width_values(4);
      cust_cfg.endpoint_cfg.pcie_cfg.pl_cfg.set_link_width_values(4);
```

```
      // Register config with the registry so that it will be picked up by the env:
      svt_config_object_db#(pcie_device_system_configuration)::set(this,  ,
         {"env", ".", "cfg"}, cust_cfg);

      // Create dl_tlp_exc_list:
      dl_tlp_exc_list = new("dl_tlp_exc_list");

      // Pass the constrained exception list to Data Link Layer:
      uvm_config_db#(svt_pcie_dl_tlp_exception_list)::set(this,
         "env.endpoint.port0.dl0",
         "dl_tlp_xact_exception_list",
         dl_tlp_exc_list);

      // Create error report catcher object and register it:
      err_catcher = new();
      uvm_report_cb::add(null, err_catcher);
   endfunction : build_phase

   function void end_of_elaboration_phase(uvm_phase phase);
      super.end_of_elaboration_phase(phase);
      // Create the callback object:
      dl_tlp_cb = new("dl_tlp_cb");
      dl_tlp_cb.randomize();
      // Register the callback object with the DL component:
      uvm_callbacks#(svt_pcie_dl,svt_pcie_dl_callback)::add(env.endpoint.port.dl,
         dl_tlp_cb);
      `uvm_info(get_full_name(), "Exiting...", UVM_HIGH);
   endfunction : end_of_elaboration_phase

   task run_phase( uvm_phase phase );
      `uvm_info(get_full_name(), "Running test dl_tlp_example_callback .\n",
         UVM_LOW);
   endtask : run_phase
endclass : dl_tlp_example_callback

// This is the callback class. It is instantiated in the test class above:
class tx_dl_tlp_callback extends svt_pcie_dl_callback;
   // Factory registration:
   `uvm_object_utils(tx_dl_tlp_callback)
   //`svt_uvm_object_utils(tx_dl_tlp_callback)

   // There are two basic ways to modify a value in a transaction (which is
   // really what we are aiming to do with the callback):
   //   1. Explicitly modify the value (e.g. transaction.<field> = <value> )
   //   2. Use an exception to cause a modification (typically for 'calculated' fields
   //      such as CRC).
   // To use an exception, you need two things:
   //   1. The "exception" - one per modification to the transaction (note that the
   //      type [class] of this object is specific to the transaction).
   //   2. The "exception list" - holds the exception(s) (again, the
   //      exception-list class is specific to the transaction.)

   // Exceptions - Use one each for the TLP transaction and the DL/TLP (which is
   // essentially a TLP transaction with a sequence number and LCRC added):
```

```
   // For a TLP transaction (within the DL/TLP transaction):
   svt_pcie_tlp_exception my_tlp_exc = new("my_tlp_exc");
   // For DL/TLP transaction:
   svt_pcie_dl_tlp_exception my_dl_tlp_exc = new("my_dl_tlp_exc");

   // Exception lists - Use one per transaction type that you intend to modify:
   tlp_exc_list_via_callback my_tlp_exc_list = new("my_tlp_exc_list");
   dl_tlp_exception_list_via_callback my_dl_tlp_exc_list = new("my_dl_tlp_exc_list");

   // Error counter:
   static int error_count = 0;
   rand   bit do_lcrc_err;

  // Constructor:
   function new(string name = "tx_dl_tlp_callback");
      super.new(name);
   endfunction : new

   // pre_tlp_framed_out_put: this is the actual callback function. It will be
   // called every time a DL/TLP is ready to be framed for transmission:
   virtual function void pre_tlp_framed_out_put( svt_pcie_dl dl,
         svt_pcie_dl_tlp transaction,
         ref bit drop);
      `uvm_info(get_full_name(),
         $sformatf("\nA callback prior to transmitting TLP. Current TC=%0d and
            ECRC = 0x%x, error count=%0d.\n",
            transaction.tlp.traffic_class,
            transaction.tlp.ecrc, error_count),
            UVM_LOW);
      if(error_count < 1 ) begin    // Just corrupt one TLP
         if (do_lcrc_err) begin   // inject an LCRC err
            // The AUTO_TX_FORCE_LCRC causes the 'corrupted_data' value to
            // be forced into the LCRC field:
            my_dl_tlp_exc.corrupted_data = 32'hbaad_baad; // Forced LCRC value
            my_dl_tlp_exc.error_kind = svt_pcie_dl_tlp_exception::AUTO_TX_FORCE_LCRC;
         end
         else begin
           my_dl_tlp_exc.error_kind=svt_pcie_dl_tlp_exception::AUTO_TX_ILLEGAL_SEQ_NUM;
         end
         // Put the exc into the list, then copy/cast the list into the transaction:
         my_dl_tlp_exc_list.add_exception(my_dl_tlp_exc);
         `svt_note("pre_tlp_framed_out_put",
            $sformatf(" Attaching DL/TLP exception list .\n."));
         $cast(transaction.exception_list, my_dl_tlp_exc_list.`SVT_DATA_COPY());
         error_count++;
      end
   endfunction
endclass : tx_dl_tlp_callback

///////// Various helper classes: exception lists, error catcher ///////////////
// tlp_exc_list_via_callback: see above for details of exception lists.
// This is for the TLP transaction encapsulated in the DL/TLP transaction
class tlp_exc_list_via_callback extends svt_pcie_tlp_transaction_exception_list;
   // The default exception, in case we have no others defined:
```

```
    svt_pcie_tlp_exception xact_exc = new("xact_exc");

    // Constructor:
    function new(string name = "tlp_exc_list_via_callback",
       svt_pcie_tlp_exception xact_exc = null);
       super.new(name,xact_exc);
       xact_exc = this.xact_exc;
       xact_exc.NO_ERROR_wt = 1;             // Implies no errors
       xact_exc.set_constraint_weights(0);
       this.randomized_exception = xact_exc; // insert this exception into our list
    endfunction : new
 endclass : tlp_exc_list_via_callback

 // dl_tlp_exception_list_via_callback: see above for details of exception lists.
 // This is for actual the DL/TLP transaction.
 class dl_tlp_exception_list_via_callback extends svt_pcie_dl_tlp_exception_list;
    // The default exception - in case we have no others defined.
    svt_pcie_dl_tlp_exception xact_exc = new("xact_exc");
    // Constructor:
    function new(string name = "dl_tlp_exception_list_via_callback",
       svt_pcie_dl_tlp_exception xact_exc = null);
       super.new(name,xact_exc);
       xact_exc = this.xact_exc;
       xact_exc.NO_ERROR_wt = 1;
       xact_exc.set_constraint_weights(0);
       this.randomized_exception = xact_exc;
    endfunction : new
 endclass : dl_tlp_exception_list_via_callback

 // Class to demote UVM_WARNING and UVM_ERROR messages:
 class dl_tlp_err_catcher extends uvm_report_catcher;
    // Constructor:
    function new(string name="error catcher");
       super.new(name);
    endfunction

    // catch(): This function is where we handle the actual UVM WARNING/ERROR
    // messages; it suppresses the errors that would occur due to the corrupted
    // CRC, etc.
    virtual function action_e catch();
       // Error catcher required if CORRUPT_TC is injected:
       if(get_severity() == UVM_ERROR) begin
          if ((uvm_is_match("*Received TLP with invalid seq num*", get_message())))
             begin
                set_severity(UVM_INFO);
             end
       end
       else if (get_severity() == UVM_WARNING) begin
          if ((uvm_is_match("*Received TLP with bad LCRC*", get_message()))) begin
             set_severity(UVM_INFO);
          end
       end
       return THROW;
    endfunction : catch
```

```
endclass : dl_tlp_err_catcher
```

19.5 SVT VIP Callbacks Reference

The callbacks that currently are supported and their arguments are listed in Table 19-1.

Callbacks and their methods for each layer are listed in Table 19-2.

Table 19-1 Supported PCIe callbacks

Function Name	Arguments (type and name)	I/O	Values
pre_tlp_framed_out_put DL Layer Called just prior to framing a TLP for transmission to the Phy Layer.	svt_pcie_dl dl	I	The component object of the calling layer.
	svt_pcie_dl_tlp transaction	I	The DL/TLP transaction to be examined/modified. Note that it also contains a TLP object.
	drop	ref	When set, the transaction is dropped prior to transmission. **NOTE: Currently not implemented**
post_tlp_framed_in_get DL Layer Called just after reception from the Phy Layer.	svt_pcie_dl dl	I	The component object of the calling layer.
	svt_pcie_dl_tlp transaction	I	The DL/TLP transaction to be examined/modified. Note that it also contains a TLP object.
	bit drop	ref	When set, the transaction is dropped prior to transmission. **NOTE: Currently not implemented**
tx_dllp_started DL Layer Called just prior to transmitting a DLLP to the Phy Layer.	svt_pcie_dl dl	I	The component object of the calling layer.
	svt_pcie_dlllp transaction	I	The DLLP transaction to be examined/modified.
	bit drop	ref	When set, the transaction is dropped prior to transmission. **NOTE: Currently not implemented**
rx_dllp_started DL Layer Called just after reception from the Phy Layer.	svt_pcie_dl dl	I	The component object of the calling layer.
	svt_pcie_dlllp transaction	I	The dllp transaction t be examined/modified.
	bit drop	ref	When set, the transaction is dropped prior to transmission. **NOTE: Currently not implemented**

Table 19-1 Supported PCIe callbacks (Continued)

pre_tx_tlp_put Target Application Called by the component just after scheduling a TLP transaction for transmission on the link, just prior to framing.	svt_pcie_target_app target_app	I	The component object of the calling layer.
	svt_pcie_tlp transaction	I	The transaction to be examined/modified.
	bit drop	ref	When set, the transaction is dropped prior to framing and is not transmitted.
post_rx_tlp_get Target Application Called by the component after receiving a TLP transaction from the link.	svt_pcie_target_app target_app	I	The component object of the calling layer.
	svt_pcie_tlp transaction	I	The transaction to be examined/modified.
	bit drop	ref	When set, the transaction is dropped without any further processing.
transaction_ended Driver Application Called by the component when the transaction is complete	svt_pcie_driver_app driver	I	The component object issuing the callback
	svt_pcie_driver_app_transaction transaction	I	The transaction to be examined/modified.
tx_ts_os_started Phy Layer Called by the component after building a TS OS transaction.	svt_pcie_pl pl	I	The component object issuing this callback.
	svt_pcie_os transaction	I	The transaction to be examined/modified.
tx_os_started Phy Layer Called by the component after building a non-TS OS Transaction	svt_pcie_pl pl	I	The component object issuing this callback.
	svt_pcie_os transaction	I	The transaction to be examined/modified.
pre_symbol_out_put Phy Layer Called by the component at every symbol time after gathering all the symbols to be transmitted on the link	svt_pcie_pl pl	I	The component object issuing this callback.
	svt_pcie_symbol symbols[]	I	The array of symbols to be examined/modified.

Table 19-1 Supported PCIe callbacks (Continued)

symbol_stripe_ended Phy Layer Called by the component when the symbol stripe has been completed	svt_pcie_pl pl	I	The component object issuing this callback.
	svt_pcie_symbol_stripe symbol_stripe	I	The symbol stripe object to be examined.

Table 19-2 Callback classes and methods by layer

Layer	Callback Class	Method	Description
Driver App	svt_pcie_driver_app_callback	transaction_ended	Called by the component once the transaction is completed by the link partner. Completion data returned by the link partner is now available in the payload.
Target App	svt_pcie_target_app_callback	post_rx_tlp_get	Called by the component after recognizing a TLP transaction received immediately from the link.
		pre_tx_tlp_put	Called by the component after scheduling a TLP transaction for transmission on the link, just prior to framing.
TL Layer	svt_pcie_tl_callback	post_seq_item_get	Called by the component after pulling a TLP out of its TLP input, but before acting on the TLP.
		pre_tlp_out_put	Called by the component once the TLP is completely received and prior to putting the TLP on the rx port.
DL Layer	svt_pcie_dl_callback	post_tlp_framed_in_get	Called by the component after recognizing a TLP transaction received immediately from the link.
		pre_tlp_framed_out_put	Called by the component after scheduling a TLP transaction for transmission on the link, just prior to framing.
		rx_dllp_started	Called by the component after receiving user DLLP transaction from an input port.
		tx_dllp_started	Called by the component after constructing a DLLP transaction just prior to its further processing.

Table 19-2 Callback classes and methods by layer (Continued)

PL Layer	svt_pcie_pl_callback	pre_symbol_out_put	Called by the component at every symbol time after gathering all the symbols to be transmitted on the link. Last chance to corrupt any symbol before it goes on the link.
		tx_os_started	Called by the component after building an OS Transaction. The callback gives the user a chance to attach an exception list to the OS transaction prior to its transmission on the link.
		tx_ts_os_started	Called by the component after building a TS OS transaction. The callback gives the user a chance to attach an exception list to the TS OS transaction prior to its transmission on the link.
		symbol_stripe_ended	Called by the component when the symbol stripe has been completed. The symbol_stripe object contains the information necessary to log symbol data. Writes to the symbol_stripe object are ignore.

19.6 Transaction Layer Callbacks and Exceptions

The Transaction Layer provides a callback class, svt_pcie_tl_callback. This callback class is used to apply exceptions that can modify or observe data.

The Transaction Layer callbacks are used only in case of transactions that are generated externally to the model. These callbacks will not be executed for a transaction from internal applications (requests from the Driver or Requester Applications, or completions from the Target Application). Only testbenches that generate request objects and push them directly into the TL should use the TL callbacks.

```
class svt_pcie_tl_callback extends svt_callback;
extern function void new ( string name = "svt_pcie_tl_callback" );
extern virtual function void pre_tlp_out_put ( svt_pcie_tl tl , svt_pcie_tlp tlp ,
ref bit drop );
extern virtual function void post_seq_item_get(svt_pcie_tl tl, svt_pcie_tlp tlp,
ref bit drop);
endclass
```

Table 19-3 Transaction Layer Callbacks

Callback	VIP Direction	Behavior
post_seq_item_get	TX	Called by the component after pulling a TLP out of its TLP input port, but before acting on the TLP in any way. In other words, this callback is only used for transactions coming either directly from TL input port or from external user applications.

Table 19-3 Transaction Layer Callbacks

Callback	VIP Direction	Behavior
pre_tlp_out_put	RX	Callback issued by the component after a completion is received completely and prior to transmitting to the RX port. In other words, this callback is only used for completions that are in response to requests that come into model via TL input port.

19.6.1 Transaction Layer Exceptions

Transaction Layer exceptions can be applied using the svt_pcie_tl_callback class. The callback methods in this class support the svt_pcie_tlp transaction class. The svt_pcie_tlp transaction class includes an exception list, svt_pcie_tlp_exception_list, which contains an exception, svt_pcie_tlp_exception. The svt_pcie_tlp_exception exception is used to be specify the type of exception to be applied.

By default the exception list is null, thus indicating no exception is to be applied. Exceptions are added by adding a handle to an exception list that is not null. For more information on the available exception types, refer to the HTML class reference of the exception class, svt_pcie_tlp_exception.

19.7 Datalink Layer Callbacks and Exceptions

The Datalink Layer provides a callback class, svt_pcie_dl_callback. This callback class is used to apply exceptions that can modify or observe data.

```
class svt_pcie_dl_callback extends svt_callback;
   extern function new(string name = "svt_pcie_dl_callback");
   extern virtual function void pre_tlp_framed_out_put(svt_pcie_dl dl,
                                           svt_pcie_dl_tlp transaction, ref bit drop);
   extern virtual function void post_tlp_framed_in_get(svt_pcie_dl dl,
                                           svt_pcie_dl_tlp transaction, ref bit drop);
   extern virtual function void tx_dllp_started(svt_pcie_dl dl,
                                           svt_pcie_dllp transaction, ref bit drop);
   extern virtual function void rx_dllp_started(svt_pcie_dl dl,
                                           svt_pcie_dllp transaction, ref bit drop);
endclass
```

Table 19-4 Datalink Layer Callbacks

Callback	VIP Direction	Behavior
post_tlp_framed_in_get	RX	Callback issued by the component after recognizing a TLP transaction received immediately from a link
pre_tlp_framed_out_put	TX	Callback issued by the component after scheduling a TLP transaction for transmission on the link, just prior to framing
rx_dllp_started	RX	Callback issued by the component after receiving User DLLP transaction from an input port
tx_dllp_started	TX	Callback issued by the component after building a DLLP transaction just prior to its further processing

19.7.1 Datalink Layer Exceptions

Datalink Layer exceptions can be applied using the svt_pcie_dl_callback class. Each callback method in this class supports a particular transaction class. Each transaction class includes an exception list which contains an exception that is used to be specify the type of exception to be applied.

By default the exception list is null, thus indicating no exception is to be applied. Exceptions are added by adding a handle to an exception list that is not null.

For example, the post_tlp_framed_in_get callback method supports the svt_pcie_dl_tlp transaction class. The svt_pcie_dl_tlp transaction class includes an exception list, svt_pcie_dl_tlp_exception_list, that contains an exception, svt_pcie_dl_tlp_exception. The svt_pcie_dl_tlp_exception exception is used to specify the exception type. Table 19-5 highlights the relationships between the DL callback methods and their associated classes.

For more information on the available exception types, refer to the HTML class reference of the exception classes in Tables 19-5.

Table 19-5 Datalink Layer Callback Method and Associated Classes

Callback	Transaction Class	Exception List Class	Exception Class
post_tlp_framed_in_get	svt_pcie_dl_tlp	svt_pcie_dl_tlp_exception_list	svt_pcie_dl_tlp_exception
pre_tlp_framed_out_put	svt_pcie_dl_tlp	svt_pcie_dl_tlp_exception_list	svt_pcie_dl_tlp_exception
rx_dllp_started	svt_pcie_dllp	svt_pcie_dllp_exception_list	svt_pcie_dllp_exception
tx_dllp_started	svt_pcie_dllp	svt_pcie_dllp_exception_list	svt_pcie_dllp_exception

19.8 Physical Layer Callbacks and Exceptions

The Physical Layer provides a callback class, svt_pcie_pl_callback. This callback class is used to apply exceptions that can modify or observe data.

```
class svt_pcie_pl_callback extends svt_xactor_callback;
   extern function new(string name = "svt_pcie_pl_callback");
   extern virtual function void tx_ts_os_started(svt_pcie_pl pl,
                                                  svt_pcie_os transaction);
   extern virtual function void tx_os_started(svt_pcie_pl pl, svt_pcie_os transaction);
   extern virtual function void pre_symbol_out_put(svt_pcie_pl pl,
                                                  svt_pcie_symbol symbols[]);
   extern virtual function void symbol_stripe_ended(svt_pcie_pl pl,
                                          svt_pcie_symbol_stripe symbol_stripe);
   virtual function void pre_pipe_data_out_put(svt_pcie_pl pl, svt_pcie_pipe_data
                                                          pipe_data);
  virtual function void post_pipe_data_in_get(svt_pcie_pl pl, svt_pcie_pipe_data
                                                          pipe_data);
endclass
```

Table 19-6 Physical Layer Callbacks

Callback	VIP Direction	Behavior
pre_symbol_out_put	TX	Callback issued by the PL at every symbol time after gathering all the symbols to be transmitted on the PCIe link. This is the last chance the user has to corrupt any symbol before it goes on the link. Note, for muli-lane configures the symbols are striped per lane into symbols[].
tx_os_started	TX	Callback issued by the component after building an OS Transaction. The callback gives user a chance to attach exception list to the OS transaction prior to its transmission on the link
tx_ts_os_started	TX	Callback issued by the component after building a TS OS Transaction. The callback gives user a chance to attach exception list to the TS OS transaction prior to its transmission on the link
symbol_stripe_ended	TX	Called by the component when the symbol stripe has been completed. The symbol_stripe object contains the information necessary to log symbol data. Writes to the symbol_stripe object are ignore.
pre_pipe_data_out_put	TX	Callback issued by the component just before every `posedge` pclk after gathering all the `pipe_data` signals to be transmitted on the PCIe link. This is the last chance to force any pipe data signals before it goes on the wire. Note: MAC must be turned-off for this callback to work so that all the data is driven by the PIPE callback only. Service request `HOT_PLUG_UNPLUG` can be enabled to turn-off the MAC.
post_pipe_data_in_get	RX	Callback issued by the component after every posedge of pipe_clk after gathering all the pipe data signals that were received on the PIPE bus. Used solely to report values of the per-lane datapath signals received off the PIPE bus. Exceptions are not supported by this callback. Note: MAC must be turned-off as users are responsible for handling/demoting any errors that may occur if the MAC is turned on. Service request `HOT_PLUG_UNPLUG` can be enabled to turn-off the MAC.

19.8.1 Physical Layer Exceptions

Physical Layer exceptions can be applied using the svt_pcie_pl_callback class. Each callback method in this class supports a particular transaction class. Each transaction class includes an exception list which contains an exception that is used to be specify the type of exception to be applied.

By default the exception list is null, thus indicating no exception is to be applied. Exceptions are added by adding a handle to an exception list that is not null.

For example, the pre_symbol_out callback method supports the svt_pcie_symbol transaction class. The svt_pcie_symbol transaction class includes an exception list, svt_pcie_symbol_exception_list, which contains an exception, svt_pcie_symbol_exception. The svt_pcie_symbol_exception exception is used to specify the exception type. Tables 19-7 highlights the relationships between the PL callback methods and their associated classes.

For more information on the available exception types, refer to the HTML class reference of the exception classes in Tables 19-7.

Table 19-7 Physical Layer Callback Methods and Associated Classes

Callback	Transaction Class	Exception List Class	Exception Class
pre_symbol_out_put	svt_pcie_symbol	svt_pcie_symbol_exception_list	svt_pcie_symbol_exception
tx_os_started	svt_pcie_os	svt_pcie_os_exception_list	svt_pcie_os_exception
tx_ts_os_started	svt_pcie_os	svt_pcie_os_exception_list	svt_pcie_os_exception
pre_pipe_data_out_put	svt_pcie_pipe_data pipe	svt_pcie_pipe_data_exception_list	svt_pcie_pipe_data_exception
post_pipe_data_in_get	svt_pcie_pipe_data pipe	Not Applicable	Not Applicable

19.9 Controlling Completion Timing and Size Using Callbacks

There are two ways to control completion timing:

1. By specifying a delay for the current completion.
2. By specifying a delay for the next completion.

Control of completion size is applied only to the next completion, and not the current completion.

19.9.1 Controlling Delay for the Current Completion

To specify a delay for the current completion, use the completion_delay_in_ns attribute in the svt_pcie_tlp class.

This is used only in associated with the following callback:

- svt_pcie_target_app_callback::pre_tx_tlp_put

It is ignored in all other use.

When used at the specified callback this attribute controls the delay between the time the callback occurs (at creation of the completion) and the time the completion is sent from the target application to the transaction layer. It does not incorporate the delay through the layers or the time required to actually transmit the completion, which may be significant for large memory read completions on narrow links.

This value is ignored if:

- This TLP is not a completion.
- The value is not set during the specified callbacks.
- The value is negative.

19.9.2 Controlling Delay for the Next Completion

To specify a delay for the next completion, use the next_completion_delay_in_ns attribute in the svt_pcie_tlp class . It specifies the delay in ns to the creation of a completion that follows the current TLP.

This is used only in association with the following callbacks:

- svt_pcie_target_app_callback::post_rx_tlp_get
- svt_pcie_target_app_callback::pre_tx_tlp_put

It is ignored in all other use.

When used at the specified callbacks it controls the delay between:

- The execution of the callback, and
- The target application creating the subsequent completion

It does not incorporate the delay through the layers or the time required to actually transmit the completion, which may be significant for large memory read completions on narrow links.

The delay is used in two cases:

1. A non-posted request received by the vip: the value specifies the delay to the first completion transmitted by the vip in response.
2. A memory read completion transmitted by the vip, if there will be at least one subsequent completion for the same request: the value specifies the delay to the next completion transmitted by the VIP.

To control completion timing using this value, set it in either the pre_tx_tlp_put callback or the post_rx_tlp_get callback in the svt_pcie_target_app_callback class.

This value is ignored if:

- This TLP is neither a non-posted request received by the vip nor a memory read completion transmitted by the VIP.
- This TLP is the last completion of a memory read request.
- The value is not set during the specified callbacks.
- The value is negative.

19.9.3 Controlling Size for the Next Completion

You can control the size of completions by using callbacks and the class member next_completion_size_in_rcb. It specifies the size in RCB units of the completion that follows this TLP.

It is used only in associated with the following callbacks:

- svt_pcie_target_app_callback::post_rx_tlp_get
- svt_pcie_target_app_callback::pre_tx_tlp_put

The next_completion_size_in_rcb attribute is ignored in all other uses. It is used in two cases:

- A memory read request received by the vip: it specifies the size in RCB of the first completion transmitted by the vip in response.
- A memory read completion transmitted by the vip, if there will be at least one subsequent completion for the same request: it specifies the size of the next completion transmitted by the VIP.

If the value is set to 1, the VIP will transmit only enough data to reach the first RCB. In that case, the length of the first completion could be as small as one dword.

To control completion size using this value, set it in either the pre_tx_tlp_put callback or the post_rx_tlp_get callback in the svt_pcie_target_app_callback class.

This value is ignored if:

- This TLP is not a memory read request received by the vip or a completion transmitted by the VIP in response to a memory read request.
- This TLP is the last completion of a read request.
- The value is not set during the specified callbacks.

This value is not intended to produce error scenarios and will also be ignored if the value is less than 1. For large values the completion size will be truncated to either the max payload size or the max read completion data size, whichever is smaller.

If the specified size is larger than the number of dwords remaining to finish the request, the next completion will include all remaining data (unless limited by the max payload size or max read completion data size).

20 Passive Monitor

The Synopsys PCIe Passive Monitor Verification IP supports verification of designs that include interfaces implementing the PCIe Base Specification version 3.0a. This document describes the use of PCIe Passive Monitor VIP in testbenches that comply with the SystemVerilog Universal Verification Methodology (UVM).

This approach leverages advanced verification technologies and tools that provide:

- Protocol functionality and abstraction
- Constrained random verification
- Functional coverage
- Rapid creation of complex tests
- Modular testbench architecture that provides maximum reuse, scalability and modularity
- Proven verification approach and methodology
- Transaction-level models
- Self-checking tests
- Object oriented interface that allows OOP techniques

This document assumes that you are familiar with PCIe, object oriented programming, SystemVerilog, and UVM. For the Synopsys PCIe VIP class reference HTML documentation, see:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/index.html

The PCIe Passive Monitor VIP is a standalone UVM-based verification component that can optionally be used in conjunction with the PCIe active component UVM VIP. Both are compatible for use with SystemVerilog-Compliant testbenches. The PCIe VIP active component simulates PCIe transactions through its active agent, while the passive component (monitor) receives transactions from both directions on the PCIe link.

Note Contact Synopsys for details.

20.1 Supported Protocol Features

The PCIe Passive Monitor VIP currently supports the following protocol features:

- PCIe Gen 1/2/3/4
- PIPE (up to version 4.2)
- Serial Interfaces
- One to 32 lanes
- Tracking of LTSSM states for devices on both sides of the link including Equalization and Low Power States
- Configuration Space Tracking
- Atomic Ops
- Callbacks at PL/DL/TL for Test Bench scoreboard support
- Mid Simulation Reset

20.2 Early Adopter (EA) Supported Features

The following features are at an Early Adopter level:

- Gen 4 (0.5 specification version)
- Functional Coverage

20.3 Features Not Supported

The following verification features are not supported:

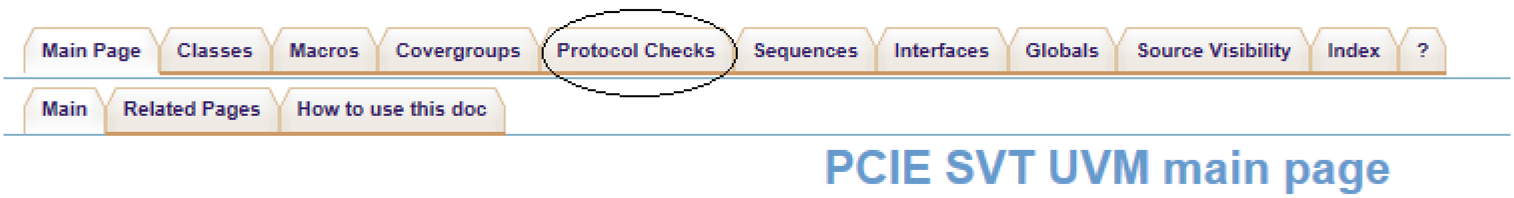
- Monitor issues ltssm_advertised_16g_data_rate_check protocol violation when active devices start 16G speed negotiation when 8G equalization stops at equalization phase 1. Issue will be fixed in next release.
- Monitor symbol log file displays blank columns for all those lanes which are not part of current active lanes. Issue will be fixed in next release.
- Passive Monitor VIP does not support OVM or VMM
- Loopback, Compliance Pattern, lane to lane skew, linkwidth upconfigure and downconfigure, lane reversal and reversed polarity are only partially tested.
- ECNs and features listed as optional in the spec other than those explicitly listed as supported in the User Guide are not supported.
- PHY DUT Monitoring.

20.4 UVM SVT Agent as Passive Monitor

Every UVM SVT agent can be made into a passive monitor. The PCIe agent follows UVM interface standards. The passive agent provides checking of the PCIe protocol at each of the transaction layers: Transaction Layer (TL), Data Link Layer (DL) and Physical Layer (PL) as well as checks specific to the LTSSM. To see a complete list of checks available go to:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/index.html

And click on the 'Protocol Checks' tab.



In addition, the monitor provides a transaction log file and functional coverage for the LTSSM state transitions. Coverage information (PASS/FAIL) is also provided for each of the error checks.

The PCIe monitor supports a non-standard symbol log that can be enabled by setting the 'enable_symbol_logging' property in the svt_pcie_pl_configuration class to '1'. The format of this symbol log mirrors the symbol log format defined by the PCIe active component and contains consolidated symbol lane information with additional annotations about synch headers in 128b/130b, and information about when the monitor detected speed changes, up-configure/down-sizing and lane reversal. Refer to "Verification Features" on page 39 for all log file formats.

The monitor only checks the device that is connected to Tx side.

Put a PCIe SVT UVM agent into passive mode with the following lines:

```
is_active =0 ;
monitor_enabled = 1;
```

NOTE. The PCIe Device agent cannot be used in the combined mode where active and passive features are supported in a single instance.

20.5 Methodology Features

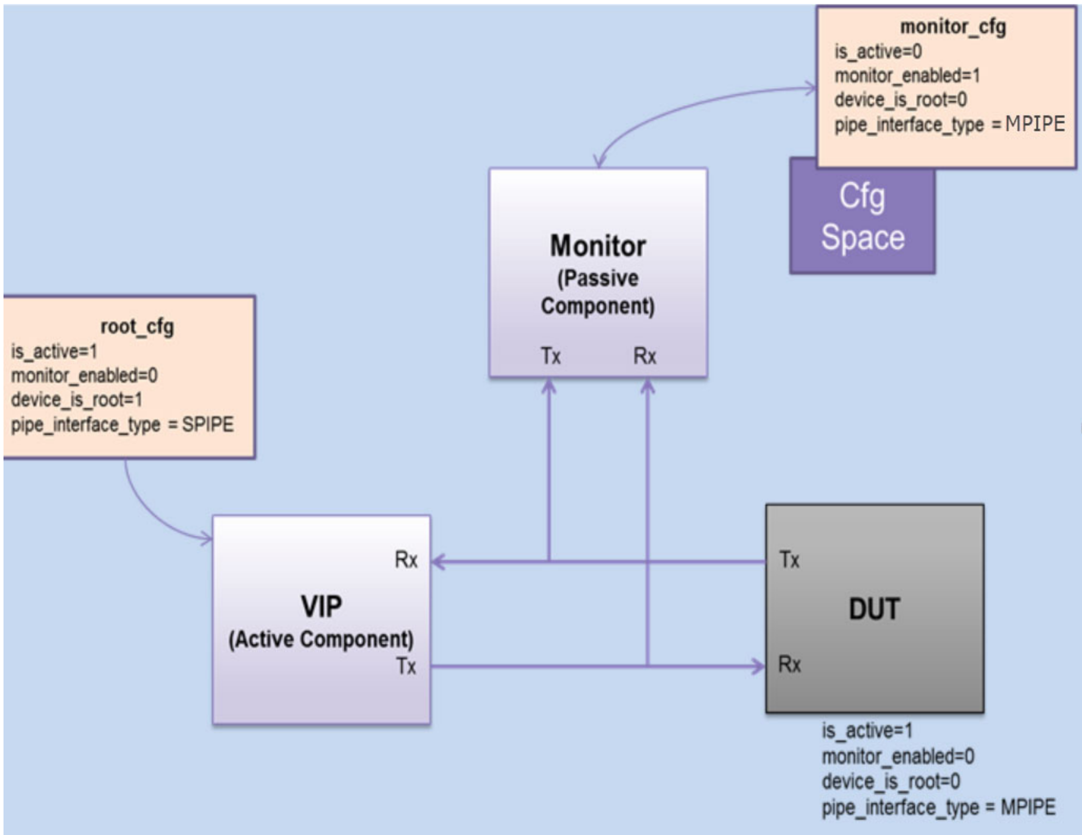
The PCIe Passive Monitor VIP currently supports the following methodology functions:

- Standalone Agent instantiation
- Analysis ports at each protocol layer for connecting to scoreboard.
- Callbacks for LTSSM Coverage
- Transaction Log File

20.6 Usage

Figure 20-1 on page 396 illustrates the connection and setup with a Root Complex and as a stand alone agent.

Figure 20-1 Monitor as a Stand Alone Agent



Note the following about Figure 20-1:

- The Monitor as a standalone agent takes has its own config object as shown in the diagram above as 'monitor_cfg'. This is an instance of svt_pcie_configuration, distinct from the instance used by the active component.
- The monitor must be programmed to match the position of the DUT or 'monitored device'. Specifically, the 'device_is_root' parameter of the monitor_cfg object should be set to '0' when the monitored device is an endpoint, and '1' when the monitored device is a root complex or downstream port of a switch device.
- The interface type of the monitor should match that of the monitored device. The Tx signals of the monitored device connect to the "Tx signals" of the passive monitor. The monitor does not drive these signals.
- The passive monitor does not generate nor derive any clocks. The testbench provides clocking to the passive monitor. Refer to the examples directory for how clocks are handled in the desired interface type.
- Configuration Space Tracking is provided for endpoint monitoring only (that is, when the monitored device is a PCIe endpoint device).
- The supported speeds (PL configuration variable `supported_speeds`) for monitor must match the highest supported speed by the DUT.

20.7 Types of Monitor Configuration Objects

Configuration data objects convey the system level and port level testbench configuration. The configuration of agents is done in the build() phase of environment or the testcase. If the configuration needs to be changed later, it can be done through reconfigure() method/

The configuration object properties can be of two types:

- Static configuration properties. Static configuration parameters specify configuration which cannot be changed when the system is running.
- Dynamic configuration properties. Dynamic configuration parameters specify configuration which can be changed at any time, irrespective of whether the system is running or not. Example of dynamic configuration parameter is timeout values.

20.8 Functional Coverage

20.8.1 LTSSM State Coverage

Coverage of LTSSM state transitions as tracked by the monitor is provided in the svt_pcie_ltssm_state_cov class. See the provided HTML documentation for further details.

20.8.2 Error Check Coverage

Each error check has PASS/FAIL coverage available. For example, for the check TXN_02_00_02 coverage is available via the svt_err_check_stats_cov_txn_02_00_02 class. See the provided HTML documentation for further details.

20.8.3 Transaction Coverage

Transaction coverage is not yet provided.

20.8.4 Coverage Callback Classes

The coverage data callback class defines default data and event information that are used to implement the coverage groups. The naming convention uses "def_cov_data" in the class names for easy identification of these classes. This class also includes implementations of the coverage methods that respond to the coverage requests by setting the coverage data and triggering the coverage events. This implementation does not include any coverage groups. The def_cov_data callbacks classes are extended from agent callback class.

The coverage data callback class is extended from respective callback class. Examples listed below.

- The class svt_pcie_ltssm_def_cov_data_callback is extended from svt_pcie_ltssm_callback.
- The class svt_pcie_tl_monitor_def_cov_data_callback is extended from svt_pcie_tl_monitor_callback.
- The class svt_pcie_dl_monitor_def_cov_data_callback is extended from svt_pcie_dl_monitor_callback.
- The class svt_pcie_pl_monitor_def_cov_data_callback is extended from svt_pcie_pl_monitor_callback.

Various methods are implemented for triggering coverage events.

20.8.5 Enabling Default Coverage

The default functional coverage can be enabled by setting the following attributes in the port configuration class svt_PCIe_port_configuration to '1'. To disable coverage, set the attributes to '0'. The attributes are:

- toggle_coverage_enable
- state_coverage_enable
- transaction_coverage_enable

Note: By default, the coverage is disabled.

20.8.6 Coverage Shaping and Control

You provide a handle to the port configuration class svt_pcie_port_configuration in the class svt_pcie_port_monitor_def_cov_callback, which implements the default cover groups. Based on the port configuration, the coverage bins are shaped. The unwanted bins are ignored.

In addition, you also have ability to disable coverage at a cover group level. Class svt_pcie_port_configuration provides members svt_pcie_port_configuration::<cover_group_name>_enable, to enable/disable cover groups. By default, the value to these members is 1.

20.9 Interfaces and modports

SystemVerilog models signal connections using interfaces and modports. Interfaces define the set of signals which make up a port connection. Modports define collection of signals for a given port, the direction of the signals, and the clock with respect to which these signals are driven and sampled.

List of all the monitor interfaces:

- svt_pcie_mpipe_x32_8g_if.svi
- svt_pcie_mpipe_x8_8g_if.svi
- svt_pcie_mpipe_x16_8g_if.svi
- svt_pcie_mpipe_x4_if.svi
- svt_pcie_spipe_x16_8g_if.svi
- svt_pcie_spipe_x32_8g_if.svi
- svt_pcie_spipe_x4_if.svi
- svt_pcie_spipe_x8_8g_if.svi
- svt_pcie_serdes_x4_if.svi
- svt_pcie_serdes_x8_if.svi
- svt_pcie_serdes_x32_if.svi
- svt_pcie_serdes_x16_if.svi

For link widths higher than x4 use the x32 interface and connect only the required lanes.

For PIPE instantiations, DUTs might implement certain PIPE defined signals as either per-lane or as a single wire, fanned out to all lanes. The testbench can be modified to accommodate differences between the DUT and the VIP pertaining to how these signals are implemented.

The following table shows how the PCIe VIP has implemented such signals:

Table 20-1 Passive Monitor Signals

Signal	PIPE Spec Version 4.0	Active Component	Monitor
Max PCLK	Shared	Shared	Shared
EncodeDecodeBypass	Optional - shared or on per-lane basis	Not implemented	Not implemented
BlockAlignControl	Optional - shared or on per-lane basis	Shared	Shared
FS	Optional - shared or on per-lane basis	Shared	Per-lane basis
LF	Optional - shared or on per-lane basis	Shared	Per-lane basis
TxSwing	Optional - shared or on per-lane basis	Shared	Shared
TxMargin	Optional - shared or on per-lane basis	Shared	Shared
TxDetectRx/Loopback	Optional - shared or on per-lane basis	Shared	Shared
Rate	Optional - shared or on per-lane basis	Shared	Shared
Width	Optional - shared or on per-lane basis	Shared	Shared
PCLK rate	Optional - shared or on per-lane basis	Shared	Shared
Reset#	Optional - shared or on per-lane basis	Shared	Shared
TxDataValid	Optional - shared or on per-lane basis	Per-lane basis	Per-lane basis
PCLK	Optional - shared or on per-lane basis	Shared	Shared
TxData, TxDataK	Per-lane basis	Per-lane basis	Per-lane basis
RxData, RxDataK	Per-lane basis	Per-lane basis	Per-lane basis
TxStartBlock	Per-lane basis	Per-lane basis	Per-lane basis
TxElecIdle	Per-lane basis	Per-lane basis	Per-lane basis
TxCompliance	Per-lane basis	Per-lane basis	Per-lane basis
RxPolarity	Per-lane basis	Per-lane basis	Per-lane basis

Table 20-1 Passive Monitor Signals (Continued)

Signal	PIPE Spec Version 4.0	Active Component	Monitor
RxValid	Per-lane basis	Per-lane basis	Per-lane basis
RxElecIdle	Per-lane basis	Per-lane basis	Per-lane basis
RxStatus	Per-lane basis	Per-lane basis	Per-lane basis
RxDataValid	Per-lane basis	Per-lane basis	Per-lane basis
RxStartBlock	Per-lane basis	Per-lane basis	Per-lane basis
TxDeemph	Per-lane basis	Per-lane basis	Per-lane basis
PowerDown	Per-lane basis	Shared	Shared
PhyStatus	Per-lane basis	Per-lane basis	Per-lane basis
RxPresetHint	Per-lane basis	Per-lane basis	Per-lane basis
RxEqEval	Per-lane basis	Per-lane basis	Per-lane basis
LinkEvaluationFeedbackFigureMerit	Per-lane basis	Per-lane basis	Per-lane basis
LinkEvaluationFeedbackDirectionChange	Per-lane basis	Per-lane basis	Per-lane basis
InvalidRequest	Per-lane basis	Per-lane basis	Per-lane basis
TxSyncHeader	Per-lane basis	Per-lane basis	Per-lane basis
RxSyncHeader	Per-lane basis	Per-lane basis	Per-lane basis
LocalPresetIndex	Per-lane basis	Per-lane basis	Per-lane basis
GetLocalPresetCoefficients	Per-lane basis	Per-lane basis	Per-lane basis
RxStandby	Per-lane basis	Per-lane basis	Per-lane basis
RxStandbyStatus	Per-lane basis	Per-lane basis	Per-lane basis
LocalFS	Per-lane basis	Per-lane basis	Per-lane basis
LocalLF	Per-lane basis	Per-lane basis	Per-lane basis
LocalTxCoefficientsValid	Per-lane basis	Per-lane basis	Per-lane basis
LocalTxPresetCoefficients	Per-lane basis	Per-lane basis	Per-lane basis

Table 20-2 Passive Monitor Signals (PIPE Version 4.2)

Signal	PIPE Spec Version 4.2	Active Component	Monitor
FS	Per-lane basis	Per-lane basis	Per-lane basis
LF	Per-lane basis	Per-lane basis	Per-lane basis

Table 20-2 Passive Monitor Signals (PIPE Version 4.2)

Signal	PIPE Spec Version 4.2	Active Component	Monitor
RxEqInProgress	Per-lane basis	Per-lane basis	Per-lane basis
PclkChangeOk	Per-lane basis	Shared	Per-lane basis
PclkChangeAck	Per-lane basis	Shared	Per-lane basis

Table 20-3 Passive Monitor Signals (PIPE Version 4.3)

Signal	PIPE Spec Version 4.3	Active Component	Monitor
AsyncPowerChangeAck	Per-lane basis	Shared	Per-lane basis

Table 20-4 Passive Monitor Signals (PIPE Version 4.4)

Signal	PIPE Spec Version 4.4	Active Component	Monitor
SRISEnable	Per-lane basis	Per-lane basis	Per-lane basis
Elasticity Buffer Mode	Per-lane basis	Shared	Shared
M2P_MessageBus	Per-lane basis	Per-lane basis	Per-lane basis
P2M_MessageBus	Per-lane basis	Per-lane basis	Per-lane basis
RxElDetectDisable	Optional - shared or on per-lane basis	Shared	Shared
TxCommonModeDisable	Optional - shared or on per-lane basis	Shared	Shared

20.10 Programming the Passive Monitor

This section gives you information on the various classes and members for programming the Passive Monitor.

20.10.1 Synopsys Passive Monitor Example

Synopsys has provided an example. It is located at:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/examples/sverilog/
tb_pcie_svt_uvm_monitor_basic_sys

The example shows the following:

- A top-level testbench in SystemVerilog
- A dummy EP DUT in the testbench, -
- A UVM verification environment
- Directed and random transaction generation.

- Programming of the Monitor's config space via the backdoor.

The README provides information about the structure of the example.

Below are the steps for installing and running the example tb_pcie_svt_uvm_monitor_basic_sys.

1. Install the example using the following command line:

   % cd <location where example is to be installed>

   % mkdir design_dir <provide any name of your choice>

   % $DESIGNWARE_HOME/bin/dw_vip_setup -path ./design_dir -e pcie_svt/tb_pcie_svt_uvm_monitor_basic_sys -svtb

   This installs the example under:

   <design_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_monitor_basic_sys

2. Use either one of the following to run the testbench:

   a. Use the Makefile:

      To run the ts.directed_test.sv test, for example, do following:

      gmake USE_SIMULATOR=vcsvlog base_pipe_test WAVES=1

      To see more options and tests you can run, invoke "gmake help".

   b. Use the sim script:

      To run the ts.random_wr_rd_test.sv test, for example, do following:

      ./run_pcie_svt_uvm_monitor_basic_sys -w base_pipe_test vcsvlog

      To see more options and tests you can run, invoke "./run_pcie_svt_uvm_monitor_basic_sys -help".

20.10.2 Configuring the Monitor in the Example Testbench

The configuration data objects contain built-in constraints, which come into effect when the configuration objects are randomized.

The PCIe VIP example testbench defines following configuration class:

- Shared Cfg. The pcie_shared_cfg class contains configuration information that is used by both the active and passive sides. Since the Passive part can only be instantiated as a standalone agent the test must provide the monitor with a unique instance of this class. In the provided examples this instance is called cfg_passive:

```
pcie_shared_cfg cfg_passive;
```

The monitor makes special use of the Cfg Space Tracking sub-class of the pcie_shared_cfg class. The monitor maintains this as a mirror of the Configuration Space of a monitored endpoint device. This space can be programmed via an enumeration sequence on the link or by the use an Analysis Port provided on the monitor.

20.10.3 Callbacks

Callbacks are an access mechanism that enable the insertion of user-defined code and allow access to objects for scoreboarding and functional coverage. Each PCIe Active Component and monitor is associated with a callback class that contains a set of callback methods. These methods are called as part of the normal flow of procedural code. There are a few differences between callback methods and other methods that set them apart.

- Callbacks are virtual methods with no code initially, so they do not provide any functionality unless they are extended. The exception to this rule is that some of the callback methods for functional coverage already contain a default implementation of a coverage model.
- The callback class is accessible to users so the class can be extended and user code inserted, potentially including testbench specific extensions of the default callback methods, and testbench specific variables and/or methods used to control whatever behavior the testbench is using the callbacks to support.
- Callbacks are called within the sequential flow at places where external access would be useful. In addition, the arguments to the methods include references to relevant data objects. For example, just before a monitor puts a transaction object into an analysis port is a good place to sample for functional coverage since the object reflects the activity that just happened on the pins. A callback at this point with an argument referencing the transaction object allows this exact scenario.
- There is no need to invoke callback methods for callbacks that are not extended. To avoid a loss of performance, callbacks are not executed by default. To execute callback methods, callback class must be registered with the component using `uvm_register_cb macro.

20.10.3.1 Callbacks in the PCIe Passive Agent

Please refer to class reference HTML documentation for details of these classes. Refer to:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/index.html

20.11 Configuration Space Tracking

The Configuration Space Tracking feature of the Passive Monitor purpose is to maintain a copy of the (Type 0) Configuration Space of the monitored device when the monitored device is a PCIe Endpoint. When the monitored device is a Root Complex or Switch downstream port, then the Passive Monitor maintains a Type 0 Configuration Space for programming purposes.

For the purpose of programming protocol checks, the Passive Monitor relies on values in its copy of the Configuration Space whenever possible. Note that there are many checks in the Passive Monitor which have parameters that are not covered by the PCIe Configuration Space; in these cases the Passive Monitor uses the configuration object instead.

When the monitored device is a PCIe Endpoint, the Configuration Space maintained by the PCIe Passive Monitor will be a copy of the Configuration Space in the DUT itself. The Passive Monitor's copy of the Configuration Space can be maintained completely by 'back door' accesses from the test bench or by a combination of back door accesses and by updates that the Passive Monitor performs by observing Configuration Read and Write transactions on the link, including those that occur as part of a discovery and enumeration process. For test benches that bypass enumeration it is typical to simply update the Passive Monitor's Configuration Space by back door accesses only.

When the monitored device is an RC or switch downstream port, then the monitor still maintains a Type 0 Configuration Space in order to program certain checks. In this instance, the Configuration Space is programmed through back door accesses only.

The Passive monitor currently implements a partial list of capabilities listed in PCI Express Specification. 'Implementing a capability set' means the passive monitor recognizes the capability ID, register list for the capability and fields for all the capability registers.

Attention The Passive Monitor does not support all the capability structures that it can recognize. For example, the PCI Express Root Complex Link Declaration Capability can be established and loaded through enumeration or through backdoor accesses, but the passive monitor does not have any checks or specific support for that capability.

20.11.1 Configuration Space Features

The following lists PCI Configuration Space Capabilities:

- Power management capability set
- MSI capability set
- MSI-X capability set
- PCI Express Capability set
- PCI Express Extended Capabilities
- Advanced Error Reporting Capability
- Virtual Channel Capability
- Multi-Function Virtual Channel Capability
- Device Serial Number Capability
- Power Budgeting Capability
- Vendor-Specific Capability

The following lists the ACS Extended Capability set for the Passive Monitor:

- Advanced Error Reporting Capability
- Virtual Channel Capability
- Multi-Function Virtual Channel Capability
- Device Serial Number Capability
- Power Budgeting Capability
- Vendor-Specific Capability
- ACS Extended Capability
- ARI Capability
- ATS (Address Translation Services) Capability
- Page Request Capability
- SR-IOV Capability
- Latency Tolerance Reporting (LTR) Capability
- PASID Extended Capability
- L1 PM Substates Extended Capability
- PCI Express Root Complex Link Declaration Capability
- PCI Express Root Complex Internal Link Control Capability
- RCRB Header Capability

20.11.2 Programming Passive Monitor's Configuration Space

Synopsys strongly recommends that the Passive Monitor Configuration Space be programed to correspond to the monitored device (DUT). In the case of an RC DUT, the Passive Monitor Configuration Space is still utilized for certain checks.

20.11.2.1 Understanding the Configuration Space Capabilities List (Linked-List) Structure

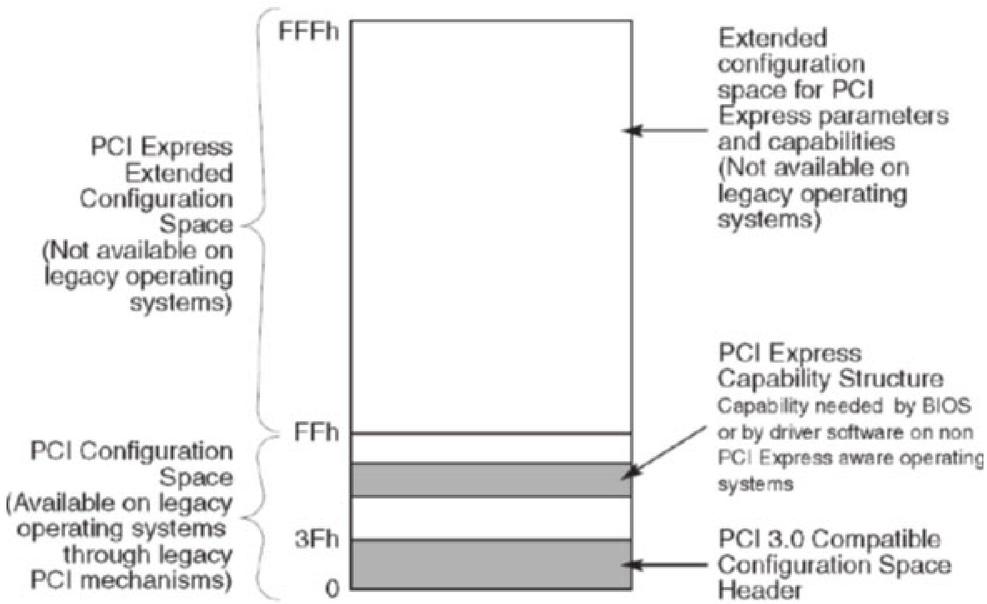
While the Passive Monitor can track link activity for the purpose of programming and maintaining its copy of an End Point DUT's configuration space (see section 2.2 Configuration Space Modes), you are still required to program the Configuration Space with at least the linked list structure of Configuration Space capabilities for all functions to be used during the simulation. This is required so that the Passive Monitor can perform checks on configuration space programming during the simulation.

This section briefly describes configuration space capabilities lists. See the PCIe base specification for more information. Start with the following:

- A capability is set of registers define in PCI, PCI express or PCI Express ECN specification. Every capability has a unique ID to identify itself in enumeration process.
- Certain parts of the PCI Express configuration space apply to the entire device and certain parts are repeated on a per-function basis.
- The Configuration Space for each function is comprised of the following parts:
  - PCI Compatible Configuration Space header
  - Legacy and PCI Express Capability
  - PCI Express Extended Configuration Space

Figure 20-2 shows the layout of Configuration Space for one function.

Figure 20-2 PCI Express Configuration Space Layout



All capabilities will have an ID and pointer to start of next capability in the list (Next Capability pointer/offset field). The last capability in the list will have all zeros in the Next Capability pointer/offset field, indicating there are no more capabilities to be discovered.

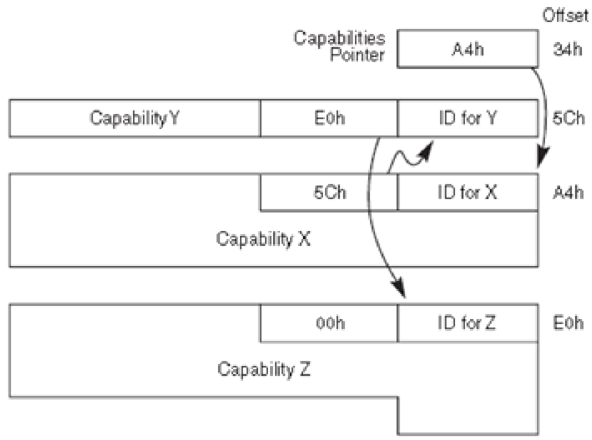
In PCI Express configuration space, there are two linked-lists for every function:

❖ PCI Configuration space linked list starting from offset address 16’h0000 to 16’h00FF. For Function 0 PCI Configuration space linked list start from 16’h0000 to 16’h00FF, for Function 1 Configuration space linked list start from 16’h1000 to 16’h10FF, similarly for other functions.

❖ PCI Express Extended Configuration Space linked list starting from offset 16’h0100 to 16’h0FFF for Function 0.

See Figure 20-3 for an example of capability list organization in PCI Configuration Space region and PCI Express Extended Configuration Space region.

Figure 20-3 Example PCI Capabilities list



20.11.3 Passive Monitor Configuration Space Modes

You can program passive monitor's configuration space in one of the three modes:

- CFG_SPACE_DISABLED
- CFG_SPACE_BACKDOOR_UPDATE
- CFG_SPACE_ENUMERATION_UPDATE

20.11.3.1 CFG_SPACE_DISABLED mode

For passive monitor CFG_SPACE_DISABLED is the default mode. In this mode configuration space programing is not needed, which also implies that monitor does not have a copy of the DUT's configuration space registers. This mode will affect the checks that rely on configuration space register values. Some checks will substitute a value from the svt_pcie_configuration class, some checks will use only the default value from specification and a few checks will be disabled altogether. While this is not the preferred mode of operation, it might be useful during initial bring-up of the Passive Monitor.

20.11.3.2 2.2.2 CFG_SPACE_BACKDOOR_UPDATE

In CFG_SPACE_BACKDOOR_UPDATE mode, the test bench programs the complete capability list structure of the DUT's configuration space and all the registers and fields which differ from their default value (as defined in the PCIe Base Specification). This programming occurs through backdoor methods before link activity begins. In this mode, configuration write or read transactions occurring on the link will not cause updates to the Passive Monitor's copy of the Configuration Space.

20.11.3.3 CFG_SPACE_ENUMERATION_UPDATE

In CFG_SPACE_ENUMERATION_UPDATE mode the Passive Monitor updates its copy of the Configuration Space based on configuration write & read transactions that occur on the link (ie, the discovery and enumeration process). As mentioned above the test bench must program the complete capability list structure the DUT's configuration space and all the registers and fields which differ from default value (as defined in the PCIe Base Specification) through backdoor methods before link activity begins. Once the link is up, the Passive Monitor will update its copy of the Configuration Space based upon the observation of successful configuration write or read transactions that occur on the link.

20.11.4 Test Bench access to PCIe Passive Monitor Configuration Space.

Access to the Passive Monitor's copy of the Configuration Space is through a transaction layer service request. Specifically the svt_pcie_tl_service object is passed to the TL level analysis port.

Table 20-5 List of Configuration Space Accessing Service Requests

Service Request	Description
MON_CONFIG_SPACE_WRITE_ADDR	Used to set a particular value at the specified address except for the capability set mappings. Once established, capability set mappings are not allowed to change. There is no other address validity checking.
MON_CONFIG_SPACE_READ_ADDR	Used to retrieve a particular value from the specified address.
MON_CONFIG_SPACE_SET_BAR_RO_MAP	Used to set the size of BAR through backdoor.
MON_CONFIG_SPACE_GET_BAR_RO_MAP	Used to retrieve size of BAR.
MON_CONFIG_SPACE_PRINT	Used to print the contents of register space in transcript when called.
MON_CONFIG_SPACE_DUMP	Used to dump the contents of register space to the file specified. The file is created if the write mode is used, else the contents are appended to the existing file when append mode is used.
MON_CONFIG_SPACE_SET_FIELD	Used to set a particular value to a register field, it provides the backdoor access to the register fields.
MON_CONFIG_SPACE_GET_FIELD	Used to retrieve the value of a particular register field of a function number from the monitor's image of the configuration space.

Note: The svt_pcie_tl_service object is used by both the PCIe active and passive agents, not all the service type and fileds are used by the passive monitor. Refer to the svt_pcie_tl_service class documentation (htlm docs) for full details.

NOTE: When the Passive Monitor receives a MON_CONFIG_SPACE_SET_FIELD or MON_CONFIG_SPACE_GET_FIELD service request, the Passive Monitor will check whether the complete configuration space capability list structure is programed. If the complete configuration space capability list is not set or incorrectly set the monitor will issue an error such as one of the following:

```
UVM_ERROR /<path>/vip/pcie_svt/src/svt_pcie_config_space_status.sv(1930) @ 205000.00 ps:
reporter [set_capability_base_addr] Configuration space detected unsupported PCIE
Capability set 00 while mapping the capability sets for function 0. This may result in
unexpected behavior.

UVM_ERROR/<path>/vip/pcie_svt/src/svt_pcie_config_space_status.sv(2467) @ 16071734.80
ps: reporter [manipulate_addr] Configuration space is trying to access unmapped
capability set SVT_PCIE_CONFIG_SPACE_PCI_EXPRESS_CAP for function 0. Hence the field
ltr_mechanism_en is unsupported.  This may result in unexpected behavior.

UVM_ERROR/<path>/vip/pcie_svt/src/svt_pcie_config_space_status.sv(1675) @ 16071734.80
ps: reporter [get_reg_field] Trying to read unmapped register field ltr_mechanism_en of
function 0 configuration space.
```

NOTE: The first service request call with MON_CONFIG_SPACE_SET_FIELD , MON_CONFIG_SPACE_GET_FIELD, MON_CONFIG_SPACE_PRINT or MON_CONFIG_SPACE_DUMP will initialize configuration space. That is, the Passive Monitor will check the validity of the linked list structure that has been programed, check that the capability ID's are from the list of recognized capabilities and create internal objects for all the capability registers & their corresponding fields. At this time the Passive Monitor will establish initial values for all the registers defined by the link list that has been established. There are two possible initial values for each register. If the testbench has written to the register address for a particular register (using the MON_CONFIG_SPACE_WRITE_ADDR service request), the monitor will use the provided value as the initial value for the register. Else the register will be initialized with the default value as per the PCI Express specification.

Once the Passive Monitor's copy of the Configuration Space is initialized, Read-Only fields cannot be changed. This means that no more capabilities can be added. The test bench should not call MON_CONFIG_SPACE_SET_FIELD, MON_CONFIG_SPACE_GET_FIELD, MON_CONFIG_SPACE_PRINT and MON_CONFIG_SPACE_DUMP service request before the full capability list is set.

The below table lists svt_pcie_tl_service fields relavent to Passive Monitor Configuration Space service requests.

Table 20-6 svt_pcie_tl_service Fields relavent to Configuration Space Access Service Requests

Service Request	Relevant field	Description
MON_CONFIG_SPACE_WRITE_ADDR	mon_cfg_space_ecam_addr	PCIe ECAM register address
	mon_cfg_space_bit_mask	mask for write Dword
	mon_cfg_space_dword_data	Dword to update the register with
MON_CONFIG_SPACE_READ_ADDR	mon_cfg_space_ecam_addr	PCIe ECAM register address
	mon_cfg_space_bit_mask	mask for write Dword
	mon_cfg_space_dword_data	Dword read from register anded with mask
MON_CONFIG_SPACE_SET_BAR_RO_MAP	mon_cfg_space_bdf_num	Function number. For setting size of BAR only mon_cfg_space_bdf_num [7:0] is used as function number only.
	mon_cfg_space_bar_num	BAR number
	mon_cfg_space_dword_data	Size of BAR. Write 1'b1 for every read-only bit for BAR, i.e. 32'h0000_00FF for 256 byte BAR size.

Table 20-6 svt_pcie_tl_service Fields relavent to Configuration Space Access Service Requests (Continued)

Service Request	Relevant field	Description
MON_CONFIG_SPACE_GET_BAR_RO_MAP	mon_cfg_space_bdf_num	Function number. For setting size of BAR only mon_cfg_space_bdf_num[7:0] is used as function number only.
	mon_cfg_space_bar_num	BAR number
	mon_cfg_space_dword_data	Return the size of the BAR.
MON_CONFIG_SPACE_PRINT		Prints the contents of configuration space in transcript with called.
MON_CONFIG_SPACE_DUMP	mon_cfg_space_dump_filename	File name for dump configuration space content
	mon_cfg_space_dump_file_mode	Indicates the mode in which to open the file for dumping configuration space. The argument can either be "w" for write or "a" for append mode.
MON_CONFIG_SPACE_SET_FIELD	mon_cfg_space_bdf_num	Function number for which the field has to be set.
	mon_cfg_space_fld_id	Field ID
	mon_cfg_space_dword_data	value that needs to be set into the field
MON_CONFIG_SPACE_GET_FIELD	mon_cfg_space_bdf_num	Function number from which the field has to be retrieved.
	mon_cfg_space_fld_id	Field ID
	mon_cfg_space_dword_data	Value of retrieved field value

For the MON_CONFIG_SPACE_SET_FIELD and MON_CONFIG_SPACE_GET_FIELD service requests, the argument mon_cfg_space_fld_id value uniquely identifies the field in the configuration space (for the function number identified by the value in the mon_cfg_space_bdf_num parameter). The mon_cfg_space_fld_id value in conjunction with function number is used internally by the monitor to calculate the register address of the register and bit position of the fields.

20.11.5 Example Programing Monitor Configuration Space

For both the CFG_SPACE_BACKDOOR_UPDATE and CFG_SPACE_ENUMERATION_UPDATE modes, the test bench programs the complete capability list structure of the DUT's configuration space and all the register & fields that differ from default value, through backdoor methods before link activity begins.

For example, consider the case where the passive monitor is to be programed corresponding to capability structures shown in Figure 20-4.

NOTE: To the below display show in Figure 4 was created through MON_CONFIG_SPACE_DUMP service request.

Figure 20-4 Example Capability Structures

```
REGISTER NAME                                   |        ADDRESS    |           VALUE
--------------------------------------------------------------------------------
Function 0 configuration space :
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
PCI 3.0 Compatible Registers :
--------------------------------------------------------------------------------
vendor_device                                   |      0x0000000   |      0x00000000
command_status                                  |      0x0000004   |      0x00100007
rev_id_class_code                               |      0x0000008   |      0x00000000
cache_lat_hdr_bist                              |      0x000000c   |      0x00000000
base_addr_reg_0                                 |      0x0000010   |      0x10000008
base_addr_reg_1                                 |      0x0000014   |      0x20000008
base_addr_reg_2                                 |      0x0000018   |      0x30000003
base_addr_reg_3                                 |      0x000001c   |      0x00000003
base_addr_reg_4                                 |      0x0000020   |      0x40000000
base_addr_reg_5                                 |      0x0000024   |      0x50000001
cardbus_cis_ptr_reg                             |      0x0000028   |      0x00000000
subsystem_id_reg                                |      0x000002c   |      0x00000000
exp_rom_base_addr_reg                           |      0x0000030   |      0x00000000
capability_ptr_reg                              |      0x0000034   |      0x00000040
intr_line_reg                                   |      0x000003c   |      0x00000000
PCI Power Management Capability Set Registers :
--------------------------------------------------------------------------------
pme_cap_reg                                     |      0x0000040   |      0x00025001
pme_cntrl_stat_reg                              |      0x0000044   |      0x00000000
Message Signaled Interrupt Capability Set Registers :
--------------------------------------------------------------------------------
msi_cap_reg                                     |      0x0000050   |      0x00006005
msi_msg_addr_reg_0                              |      0x0000054   |      0x00000000
msi_msg_data_reg                                |      0x0000058   |      0x00000000
PCI Express Capability Set Registers :
--------------------------------------------------------------------------------
pcie_cap_reg                                    |      0x0000080   |      0x00020010
dev_cap_reg                                     |      0x0000084   |      0x00000000
dev_cntrl_stat_reg                              |      0x0000088   |      0x000058b0
link_cap_reg                                    |      0x000008c   |      0x00000011
link_cntrl_stat_reg                             |      0x0000090   |      0x00010000
dev_cap_2_reg                                   |      0x00000a4   |      0x00000000
dev_cntrl_stat_2_reg                            |      0x00000a8   |      0x00000000
link_cap_2_reg                                  |      0x00000ac   |      0x00000002
link_cntrl_stat_2_reg                           |      0x00000b0   |      0x00000001
SVT_PCIE_CONFIG_SPACE_MSIX Capability Set Registers :
--------------------------------------------------------------------------------
msix_control_reg                                |      0x0000060   |      0x00018011
msix_table_bir_reg                              |      0x0000064   |      0x00000000
msix_pba_bir_reg                                |      0x0000068   |      0x00000000
Virtual Channel Extended Capability Set Registers :
--------------------------------------------------------------------------------
virtual_channel_cap_reg                         |      0x0000100   |      0x00010002
port_vc_cap_reg_1                               |      0x0000104   |      0x00000000
port_vc_cap_reg_2                               |      0x0000108   |      0x1c000000
port_vc_cntrl_stat_reg                          |      0x000010c   |      0x00000000
vc_resource_ctrl_reg_0                          |      0x0000114   |      0x800200ff
vc_resource_stat_reg_0                          |      0x0000118   |      0x00000000
--------------------------------------------------------------------------------
```

To create the capability list show in Figure 20-4, the test bench needs only to write into the six registers highlighted in red through a MON_CONFIG_SPACE_WRITE_ADDR service request. In addition, after programing the capability list the test bench will need to set Bus Master Enable, Memory Space Enable and IO Space Enable through either MON_CONFIG_SPACE_WRITE_ADDR or MON_CONFIG_SPACE_SET_FIELD service request.

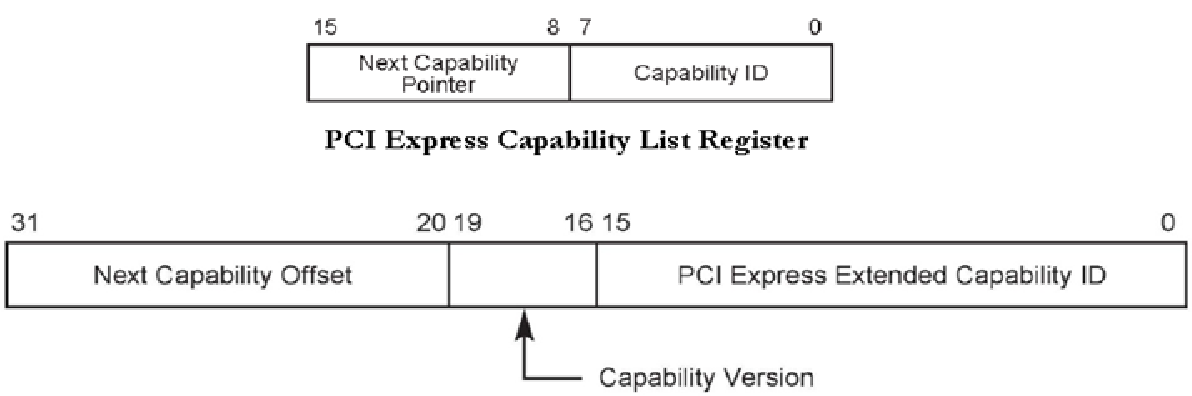
NOTE: Using MON_CONFIG_SPACE_WRITE_ADDR rather than MON_CONFIG_SPACE_SET_FIELD is generally easier when the testbench has a full, address based, copy of the DUT configuration space to work with. It still might be necessary however, to translate the DUT's absolute addresses to those used by the Passive Monitor. The Passive Monitor's Type 0 Header begins at address 0.

Programming capability list linked-list structure (shown in Figure 20-5) means the following fields and registers would be programmed as shown below for all functions:

- Capabilities Pointer in Common Configuration Space Header
  - Capability Pointer Register – offset address 0x34
- Capability ID for PCI and PCI Express capabilities & Next capability pointer.
  - PCI Power Management Capability ID – 0x01, Next capability Pointer – 0x50
  - MSI Capability ID – 0x05, Next Capability Pointer – 0x60
  - MSI-X Capability ID – 0x11, Next Capability Pointer – 0x80
  - PCI Express Capability ID – 0x10, Next Capability Pointer – 0x00 indicating end of PCIe Capability's.
- PCI Express Extended Capability ID and Next Capability Offset for all extended capabilities.
  - Virtual Channel Capability ID – 0x0002, Next Capability Offset – 0x000 indicating end of PCIe extended capabilities.

NOTE: The test bench should take care that the Next Capability pointer is 8-bit field [15:8] and Capability ID is also 8-bit field [7:0] for PCI & PCIe Capabilities. Next Capability Offset is 12 bit field [31:20] and PCIe Extended Capability ID is 16 bit field [15:0]. See the following illustration.

Figure 20-5 Capability ID and Next Capability Pointer/Offset



After programming the capability list, the test bench can alter the setting of specific fields as needed. In the example capability list in the previous, we have set Bus Master Enable; Memory Space Enable and IO Space enable to 1'b1 in Command Register (offset address - 0x4).

20.11.5.1 In CFG_SPACE_BACKDOOR_UPDATE mode

The example test case monitor_configuration_space_backdoor_load_test shows the backdoor configuration space corresponding to the previous figure, and is part of the tb_pcie_svt_uvm_monitor_basic_sys example. The path to the test case is:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/examples/sverilog/
tb_pcie_svt_uvm_monitor_basic_sys/tests/ts.monitor_configuration_space_backdoor_load_test.sv

20.11.5.2 Steps For Initializing Monitor Config Space Through Backdoor Access

This section explains the steps involved in programming the passive monitor's configuration space, as done in the tb_pcie_svt_uvm_monitor_basic_sys example. There are multiple files in this example, refer to

README for example details. Following is the directory structure and file list for tb_pcie_svt_uvm_monitor_basic_sys example.

```
<tb_pcie_svt_uvm_monitor_basic_sys>
      |-> top.sv
      |-> top.pcie_device_pipe_x4.sv
      |-> env
      |     -> pcie_device_basic_env.sv
      |     -> pcie_device_base_test.sv
      |     -> pcie_device_random_traffic_sequence.sv
      |     -> pcie_driver_transaction_directed_sequence.sv
      |     -> pcie_shared_cfg.sv
      |-> hdl_interconnect
      |     -> svt_pcie_device_pipe_x4.sv
      |-> tests
      |     -> ts.base_pipe_test.sv
      |     -> ts.monitor_configuration_space_backdoor_load_test.sv
      |     -> ts.monitor_configuration_space_enumeration_test.sv
      |-> vcs_build_options
      |-> sim_build_options
      |-> sim_run_options
      |-> methodology
      |-> modellist
      |-> Makefile
      |-> README
```

Following are the steps for configuration space updates through the backdoor:

1. "Add Analysis Port of svt_pcie_tl_service type In Environment." on page 413
2. "Add Tasks to Create and Send Service Request to Passive Monitor" on page 413
3. "Set Up Configuration Space Capability List" on page 415
4. "Program BARs" on page 415
5. "Set the Configuration Register Fields" on page 415

20.11.5.2.1 Add Analysis Port of svt_pcie_tl_service type In Environment.

Refer to the testbench file tb_pcie_svt_uvm_monitor_basic_sys/env/pcie_device_basic_env.sv

Declare the analysis port:

```
/** Analysis port to pass TL service request to monitor */
`SVT_XVM(analysis_port)#(svt_pcie_tl_service) endpoint_passive_tl_service_port;
```

Define the analysis ports in build_phase:

```
endpoint_passive_tl_service_port=new("endpoint_passive_tl_service_port",this);
```

Connect TL service analysis port to endpoint monitor in connect_phase:

```
endpoint_passive_tl_service_port.connect
(endpoint_passive.pcie_agent.tl_mon.tl_service_in_port);
```

20.11.5.2.2 Add Tasks to Create and Send Service Request to Passive Monitor

Add tasks to create and send service requests to the Passive Monitor, preferably in the base test so that it will be accessible to all the test cases extended from the base test. In this example, the tasks for creating and sending configuration space service requests are part of backdoor test, instead of base test in order to keep the example simple.

List of tasks for creating and sending service request related to configuration space backdoor to endpoint monitor in ts.monitor_configuration_space_backdoor_load_test.sv are:

```
send_cfg_space_wr_addr_tl_service_to_ep_mon
send_cfg_space_rd_addr_tl_service_to_ep_mon
send_cfg_space_set_fld_tl_service_to_ep_mon
send_cfg_space_get_fld_tl_service_to_ep_mon
send_cfg_space_set_bar_ro_map_service_to_ep_mon
send_cfg_space_get_bar_ro_map_service_to_ep_mon
send_cfg_space_print_service_to_ep_mon
send_cfg_space_dump_service_tp_ep_mon
```

Below implementation of task for sending MON_CONFIG_SPACE_WRITE_ADDR service request as an sample. For other task implementation, refer to the file ts.monitor_configuration_space_backdoor_load_test.sv in the example directory.

```
//------------------------------------------------------------------------------
// Method for sending svt_pcie_tl_service::MON_CONFIG_SPACE_WRITE_ADDR to endpoint
// monitor for writing to monitor's configuration space register
//------------------------------------------------------------------------------
virtual task send_cfg_space_wr_addr_tl_service_to_ep_mon(bit [27:0] cfg_space_ecam_addr,
                                                          bit [31:0] cfg_space_bit_mask,
                                                          bit [31:0] cfg_space_dword_data);
  svt_pcie_tl_service  tl_service     = new();

  tl_service.service_type              = svt_pcie_tl_service::MON_CONFIG_SPACE_WRITE_ADDR;
  tl_service.mon_cfg_space_ecam_addr   = cfg_space_ecam_addr;
  tl_service.mon_cfg_space_bit_mask    = cfg_space_bit_mask;
  tl_service.mon_cfg_space_dword_data = cfg_space_dword_data;
  env.endpoint_passive_tl_service_port.write(tl_service);
  tl_service.end_event.wait_on();
endtask
```

```
// BAR 0 - 1MB prefetchable 32 bit Memory Bar
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_BASE_ADDR + `SVT_PCIE_CONFIG_SPACE_BAR_0_OFFSET , 32'hffff_ffff, 32'h1000_0008);
send_cfg_space_set_bar_ro_map_service_to_ep_mon(0,0,32'h000F_FFFF);
```

20.11.5.2.3 Set Up Configuration Space Capability List

Set up configuration space capability list corresponding to the following code.

```
// Initialize the capability set header registers and enabling the capability list
// in the Status Register.
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_CAP_STAT_ADDR, 32'hffff_ffff, 32'h0010_0000);

// Set the capability pointer to 40h.
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_CAP_PTR_ADDR, 32'hffff_ffff, 32'h0000_0040);

//-----------------------------------------------------------------------------
// PCI Capability Sets
//-----------------------------------------------------------------------------
// Set PME capability register to address 40h with next capability set pointer pointing to address 'h50
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_BASE_ADDR + 28'h000_0040, 32'hffff_ffff, 32'h0002_5001);

// Set MSI capability register to address 'h50 with next capability set pointer pointing to address 60h
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_BASE_ADDR + 28'h000_0050, 32'hffff_ffff, 32'h0000_6005);

// Set MSI-X capability register to 60h with next capability set pointer pointing to address 80h
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_BASE_ADDR + 28'h000_0060, 32'hffff_ffff, 32'h0001_8011);

// Set PCIE capability register to 80h, next pointer pointing to null
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_BASE_ADDR + 28'h000_0080, 32'hffff_ffff, 32'h0002_0010);

//-----------------------------------------------------------------------------
// PCI Extended Capability Sets
//-----------------------------------------------------------------------------
// Set Virtual Channel capability register to 'h100, next pointer pointing to null ('h000)
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_EXT_CAP_PTR_ADDR, 32'hffff_ffff, 32'h0001_0002);
// Add programming of extended VC count .
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_EXT_CAP_PTR_ADDR + 28'h000_0004, 32'hffff_ffff, 32'h0000_0007);
// Set VC ARB TABLE OFFSET to null.
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_EXT_CAP_PTR_ADDR + 28'h000_0008, 32'hffff_ffff, 32'h0000_0000);
```

20.11.5.2.4 Program BARs

For programming BARs through backdoor, the test bench programs the BAR type, starting address of the BAR and BAR size. The BAR starting address and BAR type can be programed together through MON_CONFIG_SPACE_WRITE_ADDR service request. The size of the BAR is programed with a MON_CONFIG_SPACE_SET_BAR_RO_MAP service request.

For example, let us consider for Function 0, BAR0 is 1 MB Prefetchable 32-bit Memory BAR with starting address of 32'h1000_0000.

The BAR type is the last nibble of BAR register, and for Prefetchable 32-bit Memory BAR it will be 4'h8. Combining BAR type with the starting address will give us the DWord value for MON_CONFIG_SPACE_WRITE_ADDR service request, i.e. 32'h1000_0008

For addressing 1 MB you need 20 bit's, i.e. 32'h000F_FFF.

```
// BAR 0 - 1MB prefetchable 32 bit Memory Bar
send_cfg_space_wr_addr_tl_service_to_ep_mon(`SVT_PCIE_CONFIG_SPACE_FUNC_0_BASE_ADDR + `SVT_PCIE_CONFIG_SPACE_BAR_0_OFFSET , 32'hffff_ffff, 32'h1000_0008);
send_cfg_space_set_bar_ro_map_service_to_ep_mon(0,0,32'h000F_FFFF);
```

20.11.5.2.5 Set the Configuration Register Fields

After setting up the capability list and BAR, the testbench can alter some fields as needed if the test requires that these fields have values which are different from the default value in the PCIe Base Specification. In this example, we are setting set Bus Master Enable; Memory Space Enable and IO Space enable to 1'b1 in Command Register (offset address - 0x4).

To set or retrieve fields from monitor's configuration space MON_CONFIG_SPACE_SET_FIELD and MON_CONFIG_SPACE_GET_FIELD service request are used.

```
send_cfg_space_set_fld_tl_service_to_ep_mon(0, `SVT_PCIE_CMD_REG_IO_SPACE_EN_FLD, 1'b1);
send_cfg_space_set_fld_tl_service_to_ep_mon(0, `SVT_PCIE_CMD_REG_MEM_SPACE_EN_FLD, 1'b1);
send_cfg_space_set_fld_tl_service_to_ep_mon(0, `SVT_PCIE_CMD_REG_BUS_MASTER_EN_FLD, 1'b1);
```

For setting or getting a configuration space field value from test bench, the field ID value for the respective file should be known. Field ID is a value uniquely identifying the Fields per function. The complete list of macro's define for all the recognized configuration space register fields can be found in the following include file used by the Passive Monitor:

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/sverilog/include/
svt_pcie_config_space_fld_id_defines.svi

20.11.6 In CFG_SPACE_ENUMERATION_UPDATE Mode

The example test case monitor_configuration_space_enumeration_test shows the programing of monitor's configuration space in enumeration mode corresponding to capability list in Figure 4 as part of the tb_pcie_svt_uvm_monitor_basic_sys example. The path to the test case is

$DESIGNWARE_HOME/vip/svt/pcie_svt/latest/examples/sverilog/
tb_pcie_svt_uvm_monitor_basic_sys/tests/ts. monitor_configuration_space_enumeration_test.sv

As stated above, in enumeration mode, the test bench must program the capability list before link-up. The test bench can either program BARs and registers (and register fields) through backdoor access or through CFG Read and Write transactions as part of the enumeration process. The Passive Monitor will update the configuration space register with the data captured from bus after each successful CFG transaction.

20.12 Equalization Support

PCIe SVT passive monitor supports equalization at 8G as well as at 16G speeds. The Equalization configuration attributes are values provided by the test bench to enable the PCIe SVT Passive Monitor to check the equalization progress and compliance to the specification. The Equalization Status Attributes can be accessed by the test bench for specific checking on various aspects of the equalization procedure.

20.12.1 Equalization Configuration Attributes

The following configuration attributes found in the svt_pcie_pl_configuration class are used by the PCIe SVT Passive Monitor for tracking equalization states and equalization evaluation cycles and evaluating rules (protocol checks) specific to equalization at the 8G and 16G speeds. Please refer to HTML reference guide for detailed description of these properties.

Table 20-7 Attributes Common for 8G and 16G Equalization

No.	Attribute	Description
1	highest_enabled_equalization_phase	Enables VIP to go through respective equalization phases. Passive VIP's LTSSM does not transition to equalization state when this attribute is set to zero.
2	upstream_lanes_recovery_eq_phase0_timeout_ns	Represents LTSSM's timeout in ns for upstream lanes in Recovery.Equalization.0 state.
3	upstream_lanes_recovery_eq_phase1_timeout_ns	Represents LTSSM's timeout in ns for upstream lanes in Recovery.Equalization.1 state.
4	upstream_lanes_recovery_eq_phase2_timeout_ns	Represents LTSSM's timeout in ns for upstream lanes in Recovery.Equalization.2 state.
5	upstream_lanes_recovery_eq_phase3_timeout_ns	Represents LTSSM's timeout in ns for upstream lanes in Recovery.Equalization.3 state.
6	downstream_lanes_recovery_eq_phase1_timeout_ns	Represents LTSSM's timeout in ns for downstream lanes in Recovery.Equalization.1 state.
7	downstream_lanes_recovery_eq_phase2_timeout_ns	Represents LTSSM's timeout in ns for downstream lanes in Recovery.Equalization.2 state.
8	downstream_lanes_recovery_eq_phase3_timeout_ns	Represents LTSSM's timeout in ns for downstream lanes in Recovery.Equalization.3 state.
9	recovery_eq_to_speed_tolerance_timeout_ns	Represents tolerance timeout in ns for Recovery.Equalization to Recovery.Speed timeout transition of LTSSM.
10	get_local_preset_coefficients_timeout_ns	Represents time in ns at which PIPE MAC times out waiting for PIPE PHY to respond to preset to coefficients mapping request.
11	min_rx_eq_eval_delay	Represents time in ns required by PHY to respond to RxEqEval abort situation by asserting PhyStatus.
12	max_rx_eq_eval_delay	In MPIPE mode i.e. when monitored device is MAC PIPE it represents the maximum amount of time in ns MAC takes to assert RxEqEval for current evaluation cycle when accepted by link partner. In SPIPE mode i.e. when monitored device is PHY PIPE it represents the maximum amount of time in ns PHY takes to respond to RxEqEval by PhyStatus assertion.
13	enable_rxeqeval_default_settings_vector	Enables RxEqEval assertion checking for default settings in various equalization phases.
14	ltssm_recovery_equalization_rx_count	Represents number of Training Sets LTSSM must see on RX path prior to exiting current equalization phase.
15	ltssm_recovery_equalization_tx_count	Represents number of Training Sets LTSSM must see on TX path prior to exiting current equalization phase.

The attributes min_rx_eq_eval_delay and max_rx_eq_eval_delay are used differently by the active PCIe SVT VIP. The active VIP in SPIPE mode responds to RxEqEval assertion by asserting PhyStatus after delay of N ns, where N is a random number within min_rx_eq_eval_delay and max_rx_eq_eval_delay boundaries.

Table 20-8 Attributes for 8G Equalization

No.	Attribute	Description
1	preset_to_coefficients_mapping_entry_valid	Bit mapped enable for each entry in preset_to_coefficients_mapping_table
2	preset_to_coefficients_mapping_table	Represents monitored device's link partner's preset to coefficients mapping table for 8G equalization.
3	expected_preset_to_coefficients_mapping_entry_enable	Bit mapped enable for each entry in expected_preset_to_coefficients_mapping_table
4	expected_preset_to_coefficients_mapping_table	Represents monitored device's preset to coefficients mapping table for 8G equalization.

For an active PCIe SVT VIP the attribute 'preset_to_coefficients_mapping_table' represents the VIP's own preset to coefficients mappings for 8G equalization and the attribute 'expected_preset_to_coefficients_mapping_table' represents the DUT's preset to coefficients mappings for 8G equalization.

Table 20-9 Attributes for 16G Equalization

No	Attribute	Description
1	preset_to_coefficients_mapping_entry_valid_16g	Bit mapped enable for each entry in preset_to_coefficients_mapping_table_16g
2	preset_to_coefficients_mapping_table_16g	Represents monitored device's link partner's preset to coefficients mapping table for 16G equalization.
3	expected_preset_to_coefficients_mapping_entry_enable_16g	Bit mapped enable for each entry in expected_preset_to_coefficients_mapping_tabl e_16g
4	expected_preset_to_coefficients_mapping_table_16g	Represents monitored device's preset to coefficients mapping table for 16G equalization.

For an active PCIe SVT VIP attribute 'preset_to_coefficients_mapping_table_16g' represents VIP's own preset to coefficients mappings for 16G equalization and an attribute 'expected_preset_to_coefficients_mapping_table_16g' represents DUT's preset to coefficients mappings for 16G equalization.

20.12.2 Equalization Status Attributes

The PCIe SVT Passive Monitor gathers various equalization attributes of evaluation cycles in svt_pcie_pl_status class. The svt_pcie_pl_status class in turn uses svt_pcie_eq_status to represent equalization specific attributes which in turn uses values in the svt_pcie_eq_eval_cycle class to gather information regarding equalization evaluation cycle. The following table lists the status attributes made available. Refer to the HTML reference guide for a detailed description of these properties.

Table 20-10 svt_pcie_pl_status Attributes Common for 8G and 16G Equalization

No	Attribute	Description
1	start_equalization_w_preset	Represents PCIe specifications variable start_equalization_w_reset

Table 20-11 svt_pcie_pl_status attributes for 8G Equalization

No	Attribute	Description
1	equalization_done_8gt_data_rate	Represents PCIe specifications variable equalization_done_8gt_data_rate
2	equalization_complete	Represents PCIe specifications variable 'Equalization Complete'
3	equalization_phase_1_successful	Represents PCIe specifications variable 'Equalization Phase 1 Successful'
4	equalization_phase_2_successful	Represents PCIe specifications variable 'Equalization Phase 2 Successful'
5	equalization_phase_3_successful	Represents PCIe specifications variable 'Equalization Phase 3 Successful'
6	eq_status	Represents various equalization specific properties captured from TS1s in different phases of 8G equalization. Refer to HTML reference guide on details on 'svt_pcie_eq_status' and 'svt_pcie_eq_eval_cycle' class attributes.

Table 20-12 svt_pcie_pl_status attributes for 16G equalization

No	Attribute	Description
1	equalization_done_16gt_data_rate	Represents PCIe specifications variable equalization_done_16gt_data_rate
2	equalization_16g_complete	Represents PCIe specifications variable 'Equalization 16G Complete'
3	equalization_16g_phase_1_successful	Represents PCIe specifications variable 'Equalization 16G Phase 1 Successful'
4	equalization_16g_phase_2_successful	Represents PCIe specifications variable 'Equalization 16G Phase 2 Successful'
5	equalization_16g_phase_3_successful	Represents PCIe specifications variable 'Equalization 16G Phase 3 Successful'
6	eq_16g_status	Represents various equalization specific properties captured from TS1s in different phases of 16G equalization. Refer to HTML reference guide on details on 'svt_pcie_eq_status' and 'svt_pcie_eq_eval_cycle' class attributes.

21 Support for CCIX

This chapter describes the CCIX features available with the Synopsys PCIe SVT Verification IP.

This chapter discusses the following topics:

21.1 Version Support

The Synopsys PCIe SVT VIP supports the *CCIX Transport, Release 2 Draft specification, March 2017*. Functionality and controls will be updated as the specification changes.

21.2 Supported Interfaces

CCIX support is currently available for both serial and PIPE (4.4/4.4.1) interfaces using the PCIe SVT Unified model.

21.2.1 Updates for PIPE interface

For CCIX ESM support, `svt_pcie_pipe_if` interface is updated with the following:

- Widened PclkRate signal from 3bits to 4bits to accommodate 625 MHz, 781.25 MHz, 1250 MHz, 1562.5 MHz, 2500 MHz, 3125 MHz PCLK frequencies.

- ‘b1000 : 625 MHz
- ‘b1001 : 781.25 MHz
- ‘b1010 : 1250 MHz
- ‘b1011 : 1562.5 MHz
- ‘b1100 : 2500 MHz
- ‘b1101 : 3125 MHz

21.3 Supported Features

The Synopsys PCIe SVT VIP supports the following features:

- Extended Speed Mode (ESM)
- Optimized TLP Format

Note
These are independently enabled.

21.4 Support for CCIX ESM Operation

21.4.1 Enabling ESM Operation

To enable CCIX ESM operation, compile the model with the following macro defined:
`SVT_PCIE_CCIX_ESM_MODE_SUPPORTED`

Also, set the following configuration attribute to 1:

`svt_pcie_pl_configuration::ccix_esm_mode_supported`

21.4.2 Configuring ESM Operation

21.4.2.1 Interface-Independent Configuration

The model provides the following configuration attributes in `svt_pcie_pl_configuration` to support the equivalent fields in the noted ESM registers:

- CCIX Transport Capabilities Register
  - `ccix_esm_phy_reach_len_capability`
  - `ccix_esm_recal_reqd_after_data_rate_update`
  - `ccix_esm_calibration_time`
  - `ccix_esm_quick_eq_timeout`
    - `ccix_esm_quick_eq_select_1_timeout_ns`
    - `ccix_esm_quick_eq_select_2_timeout_ns`
    - `ccix_esm_quick_eq_select_3_timeout_ns`
    - `ccix_esm_quick_eq_select_4_timeout_ns`
    - `ccix_esm_quick_eq_select_5_timeout_ns`
- ESM Mandatory Data Rate Capability Register
  - `ccix_mandatory_data_rates`
- ESM Control Register

- `ccix_esm_extended_eq_ph2_timeout_sel_default`
  - `ccix_esm_extended_eq_ph2_select_0_timeout_ns`
  - `ccix_esm_extended_eq_ph2_select_1_timeout_ns`
  - `ccix_esm_extended_eq_ph2_select_2_timeout_ns`
  - `ccix_esm_extended_eq_ph2_select_3_timeout_ns`
  - `ccix_esm_extended_eq_ph2_select_4_timeout_ns`
  - `ccix_esm_extended_eq_ph2_select_5_timeout_ns`
- `ccix_esm_extended_eq_ph3_timeout_sel_default`
  - `ccix_esm_extended_eq_ph3_select_0_timeout_ns`
  - `ccix_esm_extended_eq_ph3_select_1_timeout_ns`
  - `ccix_esm_extended_eq_ph3_select_2_timeout_ns`
  - `ccix_esm_extended_eq_ph3_select_3_timeout_ns`
  - `ccix_esm_extended_eq_ph3_select_4_timeout_ns`
  - `ccix_esm_extended_eq_ph3_select_5_timeout_ns`

To support dynamic configuration that is equivalent to writing the ESM Control Register, the model implements the sequence `svt_pcie_pl_service_ccix_configure_esm_sequence`. The sequence is defined in *svt_pcie_pl_service_sequence_collection.sv*. It can be used to set the model's implementation of the fields in the ESM Control Register (model attribute names in parentheses):

- ESM Data Rate 0 (`ccix_esm_data_rate_0`)
- ESM Perform Calibration (`perform_calibration`)
- ESM Data Rate 1 (`ccix_esm_data_rate_1`)
- ESM Enable (`ccix_esm_enable`)
- ESM Extended Equalization Phase2 Timeout (`ccix_esm_extended_equalization_phase2_timeout`)
- ESM Extended Equalization Phase3 Timeout (`ccix_esm_extended_equalization_phase3_timeout`)
- Link Reach Target (`link_reach_target`)
- Quick Equalization Timeout Select (`ccix_quick_eq_timeout_select`)

Note
This sequence should be run on the PHY layer sequencer.

Example 21-1

```
svt_pcie_pl_service_ccix_configure_esm_sequence esm_seq;
`uvm_do_on_with(
    ep_esm_seq,
    p_sequencer.endpoint_virt_seqr.pcie_virt_seqr.pl_seqr,
    { ccix_esm_enable == 1;
      ccix_esm_data_rate_0 == <ccix_esm_data_rate_0>;
      ccix_esm_data_rate_1 == <ccix_esm_data_rate_1>;
      ccix_quick_equalization_timeout_select ==
<ccix_quick_equalization_timeout_select>;
      ccix_esm_extended_equalization_phase2_timeout == <phase2_timeout>;
      ccix_esm_extended_equalization_phase3_timeout == <phase3_timeout>;
    }
                )
```

The model implements calibration only as a timeout of the configured duration, with completion indicated in the status class—triggering calibration has no other effect. When `perform_calibration` is set, the timer is initiated after the model enters L1.

When configured as a root complex, the model initiates link training any time this sequence is executed with `ccix_esm_enable` set to 1. A testbench using this sequence should expect the subsequent transition to Recovery.

The model provides the following CCIX ESM status attributes in `svt_pcie_pl_status`:

- `ccix_esm_current_data_rate`
- `ccix_esm_perform_calibration`
- `ccix_esm_calibration_done`

For more information about CCIX-related attributes, see HTML class reference documentation.

21.4.2.2 Configuration for PIPE Interface

There is currently no industry specification for CCIX use over the PIPE interface. The VIP implements the CCIX interface of the Synopsys DWC Controller. This interface uses MBI to communicate commands and status.

In addition, the VIP adds encodings of `pclk_rate` and `width` in its configuration class (`svt_pcie_pl_configuration)`, to support operation at 20G and 25G rates using the PIPE interface.

For 2.5G, 5G, 8G and 16G rates, the `pclk_rate` and `width` combinations are same as the existing use model. For 20G and 25G rates, the VIP provides following additional attributes.

- `svt_pcie_pl_configuration::pclk_rate[5]` and `svt_pcie_pl_configuration::pclk_rate[6]` for 20G and 25G respectively.

  Allowed enum values for 20G are:

  - `svt_pcie_pl_configuration::PCLK_625_MHZ`
  - `svt_pcie_pl_configuration::PCLK_1250_MHZ`
  - `svt_pcie_pl_configuration::PCLK_2500_MHZ`

  Allowed enum values for 25G are:

  - `svt_pcie_pl_configuration::PCLK_781_25_MHZ`
  - `svt_pcie_pl_configuration::PCLK_1562_5_MHZ`
  - `svt_pcie_pl_configuration::PCLK_3125_MHZ`

- `svt_pcie_pl_configuration::width[5]` and `svt_pcie_pl_configuration::width[6]` for 20G and 25G respectively.

  Allowed enum values for both 20G and 25G are:

  - `svt_pcie_pl_configuration::PIPE_8_BITS`
  - `svt_pcie_pl_configuration::PIPE_16_BITS`
  - `svt_pcie_pl_configuration::PIPE_32_BITS`

When configured for MPIPE operation, the `CCIX_CONFIGURE_ESM` service also initiates Write Committed commands with the configured values to PHY registers (ESM Rate0, ESM Rate1, ESM Control, and ESM Link Reach Target) over MBI. If the VIP is configured as root and ESM Enable bit is set, then VIP verifies that Write ACK is received for the Write Committed command before initiating the link retraining.

When configured for SPIPE operation, if configured as root and the ESM Enable bit is set to 1, then VIP initiates link retraining only after the remote partner has completed the ESM Enable handshake with SPIPE over MBI.

Note
Calibration is not supported in PIPE mode.

21.5 Support for CCIX Optimized TLP Format

21.5.1 Enabling Optimized TLP Format

To enable CCIX Optimized TLP format, set each of the following configuration attributes to 1:

- `svt_pcie_configuration::is_ccix_transaction_layer`
- `svt_pcie_tl_configuration::enable_ccix_optimized_tlp`

21.5.2 Configuring VC for Optimized TLP Traffic

To configure the VC to be used for exchanging Optimized TLPs, set the following configuration attribute:

`svt_pcie_tl_configuration::ccix_vc`

If this attribute is set to 0, the model will use the highest enabled VC for Optimized TLP traffic.

The model does not support the use of VC0 or TC0 for Optimized TLP traffic. Although the specification does not permit mapping multiple TCs to the designated VC for Optimized TLP traffic, the model supports this.

21.5.3 Sending Optimized TLPs

To transmit CCIX Optimized TLPs, use the sequence `svt_pcie_tlp_ccix_optimized_tlp_sequence` defined in *svt_pcie_tlp_sequence_collection.sv* on the TLP sequencer.

Example 21-2

```
svt_pcie_tlp_ccix_optimized_tlp_sequence optimized_tlp_sequence;
uvm_do_on_with(optimized_tlp_sequence,
               vip_seqr.pcie_virt_seqr.tlp_seqr,
               { traffic_class == 3;
           ccix_byte0_bit5_to_byte1_bit7 == 7'h0;
        ccix_byte1_bit3_to_byte3_bit7 == 13'h0;
        length = 1;
    })
```

21.5.4 Receiving Optimized TLPs

CCIX Optimized TLPs received by the transaction layer can be checked via output port or callback. Optimized TLPs are not forwarded to the Target Application and consequently are not accessible by the normal mechanisms for TLPs it receives.

The output port for received TLPs in `svt_pcie_tl` is `rx_tlp_out_port`. This is a blocking put port (in UVM, the type is `uvm_blocking_put_port`) for the `svt_pcie_tlp` type. The following example shows the code that connects the port to a blocking put implementation and prints all received TLPs.

Example 21-3

```
class example_blocking_put_imp extends uvm_component;
```

```
  ...
  //Declare the blocking put implementation
  uvm_blocking_put_imp #(svt_pcie_tlp, example_blocking_put_imp) rx_tlp_in_export;

  function new( string name, uvm_component parent );
    super.new( name, parent );
    rx_tlp_in_export = new("rx_tlp_in_export", this);
  endfunction
  //Implement the put task that will be connected to the vip's output port
  virtual task put( svt_pcie_tlp t );
    //Print all received TLPs
    t.print();
  endtask
endclass

//In test, declare instance of the class with the blocking put implementation
example_blocking_put_imp  rcvd_tlp_port;

//In build_phase, create instance:
rcvd_tlp_port = new( "rcvd_tlp_port", this );

//In connect_phase, connect the vip's put port to the test's put implementation:
<vip instance>.tl.rx_tlp_out_port.connect(rcvd_tlp_port.rx_tlp_in_export);
```

The callback to access TLPs received by the transaction layer is `pre_tlp_out_put` in the `svt_pcie_tl_callbacks` class. For example, this implementation would print all received TLPs.

Example 21-4

```
class example_tl_callback extends svt_pcie_tl_callback;
  ...

  /**
  * Callback issued by the component once the TLP is received completely and
  * prior to putting received TLP on the rx port.
  *
  * @param tl: A reference to the component object issuing this callback.
  * @param tlp: A reference to the svt_pcie_tlp descriptor object of interest.
  * @param drop: A reference bit which indicates to drop the transaction when set to 1.
  */
  virtual function void pre_tlp_out_put(svt_pcie_tl tl, svt_pcie_tlp tlp, ref bit drop);
    //Print all received TLPs
    tlp.print();
  endfunction

endclass //ccix_esm_rc_tl_callback
```

The callback occurs immediately before the transaction is pushed to the output port. If the drop bit is set in the callback, the transaction will not be pushed to the output port, but the model will otherwise be unaffected.

21.5.5 Transaction Logging of Optimized TLPs

Display of optimized TLPs in the transaction log is done automatically when optimized TLPs are enabled. No other enable is required. The TLP Type field is set to `CCIX_OPT`, and only the fields that are defined for Optimized TLPs are specified.

Example 21-5

```
              Start     End     D                                    I           Address          BE|ST  Len
  Reporter    Time      Time    I TLP Type R_ID/Tag|ST  Seq TC T  P D R N   Reg#/MsgRt/Cpl        BC   Idx   Prefix (P) / Header (H) / Data (D)  E    ECRC       LCRC      TX/RX
              (ns)      (ns)    R       DLLP Type       Num VC H  H O O S   HdrFC     DataFC    MCode  DW           (All values in Hex)          P               CRC       Status
------------ --------- --------- - --------------------- ---- -- - -- - - - ------------------- ------- ---- ------------ -------- -------- -------- - ---------- ---------- --------
spd_1        52147.000 52150.000 R CCIX_OPT                 2  5                                          3 (H) 10572303   ----     ----     ----               0x4cda2071
                                                                                                          0 (D) c5efe409 4888b941 72095bfa   ----
```

21.6 Limitations

Following are the limitations of CCIX feature:

- CCIX support is only available when using PCIe SVT Unified model.
- The Driver Application does not support sending CCIX Optimized TLPs. They must be injected into the model via the transaction layer as described in this chapter.
- The Target Application does not receive CCIX Optimized TLPs. A testbench that requires access to received Optimized TLPs must do so via the transaction layer.
- The model does not automatically assign CCIX Optimized TLPs the highest priority over other VC traffic. However, the VC arbitration scheme is configurable and can support this prioritization.
- EQ Bypass is not supported when CCIX ESM is enabled.
- When using the PIPE interface:
  - CCIX ESM with PCLK as PHY input is not supported.
  - Calibration is optional and not supported.
  - Retimer is not supported with CCIX ESM.
  - VIP works only with SNPS DWC Controller 5.20a version and later.

A PCIe PIPE Interface

This chapter provides the detailed list of all the signal widths and associated macros present in PIPE interface `svt_pcie_pipe_if` and PIPE5 interface `svt_pcie_pipe5_if`.

Table A-1 lists the control macros for PIPE interfaces `svt_pcie_pipe_if` and `svt_pcie_pipe5_if`.

Table A-1 Control Macros for PIPE Interfaces

	svt_pcie_pipe_if *		svt_pcie_pipe5_if **	
Control Macros for Signal Widths	Default Value	Additional Supported Values	Default Value	Additional Supported Values
SVT_PCIE_PIPE_DATA_WIDTH	32	64	32	64, 40, 80
SVT_PCIE_PIPE_DATAK_WIDTH	4	None	4	8
SVT_PCIE_PIPE_TXELECIDLE_WIDTH	1	None	4	None
SVT_PCIE_PIPE_SYNCHDR_WIDTH	2	None	2	None
SVT_PCIE_PIPE_RATE_WIDTH	2	3	4	None
SVT_PCIE_PIPE_WIDTH_WIDTH	2	None	2	None
SVT_PCIE_PIPE_PCLKRATE_WIDTH	3	4	5	None
SVT_PCIE_PIPE_MBI_WIDTH	8	None	8	None
SVT_PCIE_PIPE_LOCALPRESETINDEX_WIDTH	5	6	NA	
SVT_PCIE_PIPE_PHYMODE_WIDTH	NA		4	None

* Used when PCIe specification version is 4.0 or earlier and PIPE specification version is 4.4 or earlier or when using custom enhancements.

** Used only when PCIe specification version is 5.0 and later and PIPE specification version is 5.0 and later.

Table A-2 lists the signals for PIPE interfaces `svt_pcie_pipe_if` and `svt_pcie_pipe5_if`.

Table A-2 Signals for PIPE Interfaces

	svt_pcie_pipe_if *		svt_pcie_pipe5_if **	
Signal Name	**Shared/per lane**	**Width**	**Shared/per lane**	**Width**
`clkreq_n`	shared	1	shared	1
`wake_n`	shared	1	shared	1
`reset`	shared	1	shared	1
`max_pclk`	shared	1	shared	1
`pclk`	shared	1	Not available	
`pclk_<ln_num>`	per lane	1	per lane	1
`pipe_reset_n`	shared	1	Not available	
`pipe_reset_n_<ln_num>`	Not available		optional	1
`block_align_control`	shared	1	Not available	
`sris_enable_<ln_num>`	per lane	1	per lane	1
`phy_status_<ln_num>`	per lane	1	per lane	1
`elasticity_buffer_mode`	shared	1	Not available	
`rx_standby_<ln_num>`	per lane	1	per lane	1
`rx_standby_status_<ln_num>`	per lane	1	per lane	1
`pclk_change_ok`	shared	1	Not available	
`pclk_change_ok_<ln_num>`	Not available		per lane	1
`pclk_change_ack`	shared	1	Not available	
`pclk_change_ack_<ln_num>`	Not available		per lane	1
`async_power_change_ack`	shared	1	Not available	
`async_power_change_ack_<ln_num>`	Not available		per lane	1
`txdetectrx_loopback`	shared	1	Not available	
`txdetectrx_loopback_<ln_num>`	Not available		optional	1
`power_down`	shared	4	Not available	
`power_down_<ln_num>`	Not available		per lane	4
`rate`	shared	SVT_PCIE_PIPE_RATE_WIDTH	Not available	

Table A-2 Signals for PIPE Interfaces (Continued)

	svt_pcie_pipe_if *		svt_pcie_pipe5_if **	
Signal Name	**Shared/per lane**	**Width**	**Shared/per lane**	**Width**
`rate_<ln_num>`	Not available		optional	SVT_PCIE_PIPE_RATE_WIDTH
`width`	shared	SVT_PCIE_PIPE_WIDTH_WIDTH	Not available	
`width_<ln_num>`	Not available		optional	SVT_PCIE_PIPE_WIDTH_WIDTH
`pclk_rate`	shared	SVT_PCIE_PIPE_PCLKRATE_WIDTH	Not available	
`pclk_rate_<ln_num>`	Not available		optional	SVT_PCIE_PIPE_PCLKRATE_WIDTH
`data_bus_width`	shared	SVT_PCIE_PIPE_WIDTH_WIDTH	Not available	
`data_bus_width_<ln_num>`	Not available		optional	SVT_PCIE_PIPE_WIDTH_WIDTH
`rx_eidetect_disable`	shared	1	Not available	
`rx_eidetect_disable_<ln_num>`	Not available		optional	1
`tx_commonmode_disable`	shared	1	Not available	
`tx_commonmode_disable_<ln_num>`	Not available		optional	1
`tx_data_<ln_num>`	per lane	SVT_PCIE_PIPE_DATA_WIDTH	per lane	SVT_PCIE_PIPE_DATA_WIDTH
`tx_data_k_<ln_num>`	per lane	SVT_PCIE_PIPE_DATAK_WIDTH	per lane	SVT_PCIE_PIPE_DATAK_WIDTH
`rx_data_<ln_num>`	per lane	SVT_PCIE_PIPE_DATA_WIDTH	per lane	SVT_PCIE_PIPE_DATA_WIDTH
`rx_data_k_<ln_num>`	per lane	SVT_PCIE_PIPE_DATAK_WIDTH	per lane	SVT_PCIE_PIPE_DATAK_WIDTH
`tx_data_valid_<ln_num>`	per lane	1	per lane	1
`tx_start_block_<ln_num>`	per lane	1	per lane	1
`tx_sync_header_<ln_num>`	per lane	SVT_PCIE_PIPE_SYNCHDR_WIDTH	per lane	SVT_PCIE_PIPE_SYNCHDR_WIDTH

Table A-2 Signals for PIPE Interfaces (Continued)

Signal Name	svt_pcie_pipe_if * Shared/per lane	svt_pcie_pipe_if * Width	svt_pcie_pipe5_if ** Shared/per lane	svt_pcie_pipe5_if ** Width
tx_elec_idle_<ln_num>	per lane	SVT_PCIE_PIPE_TXELECIDLE_WIDTH	per lane	SVT_PCIE_PIPE_TXELECIDLE_WIDTH
tx_compliance_<ln_num>	per lane	1	per lane	1
rx_valid_<ln_num>	per lane	1	per lane	1
rx_data_valid_<ln_num>	per lane	1	per lane	1
rx_start_block_<ln_num>	per lane	1	per lane	1
rx_sync_header_<ln_num>	per lane	SVT_PCIE_PIPE_SYNCHDR_WIDTH	per lane	SVT_PCIE_PIPE_SYNCHDR_WIDTH
rx_elec_idle_<ln_num>	per lane	1	per lane	1
rx_status_<ln_num>	per lane	3	per lane	3
rx_polarity_<ln_num>	per lane	1	Not available	
m2p_message_bus_<ln_num>	per lane	SVT_PCIE_PIPE_MBI_WIDTH	per lane	SVT_PCIE_PIPE_MBI_WIDTH
p2m_message_bus_<ln_num>	per lane	SVT_PCIE_PIPE_MBI_WIDTH	per lane	SVT_PCIE_PIPE_MBI_WIDTH
phy_mode_<ln_num>	Not available		per lane	SVT_PCIE_PIPE_PHYMODE_WIDTH
rxclk_<ln_num>	Not available		per lane	1
serdes_arch	Not available		shared	1
refclk_required_n_<ln_num>	Not available		per lane	1
rx_width_<ln_num>	Not available		optional	SVT_PCIE_PIPE_WIDTH_WIDTH
tx_swing	shared	1	Not available	
tx_margin	shared	3	Not available	
lf	Available but not used		Not available	
fs	Available but not used		Not available	
lf_<ln_num>	per lane	6	Not available	
fs_<ln_num>	per lane	6	Not available	

Table A-2 Signals for PIPE Interfaces (Continued)

Signal Name	svt_pcie_pipe_if *		svt_pcie_pipe5_if **	
	Shared/per lane	Width	Shared/per lane	Width
get_local_preset_coefficients_<ln_num>	per lane	1	Not available	
local_tx_coefficients_valid_<ln_num>	per lane	1	Not available	
local_tx_preset_coefficients_<ln_num>	per lane	18	Not available	
local_fs_<ln_num>	per lane	6	Not available	
local_lf_<ln_num>	per lane	6	Not available	
local_preset_index_<ln_num>	per lane	5	Not available	
rx_eq_in_progress_<ln_num>	per lane	1	Not available	
rx_preset_hint_<ln_num>	per lane	3	Not available	
rx_eq_eval_<ln_num>	per lane	1	Not available	
link_evaluation_feedback_figure_merit_<ln_num>	per lane	8	Not available	
link_evaluation_feedback_direction_change_<ln_num>	per lane	6	Not available	
invalid_request_<ln_num>	per lane	1	Not available	
tx_deemph_<ln_num>	per lane	18	Not available	

* Used when PCIe specification version is 4.0 or earlier and PIPE specification version is 4.4 or earlier or when using custom enhancements.
** Used only when PCIe specification version is 5.0 and later and PIPE specification version is 5.0 and later.

Table A-3: VIP Supports the following PCLK rates and data widths for PIPE 4.3, and PIPE 4.4 specifications:

Table A-3 PCLK Rates and Data Widths in PIPE 4.3, 4.4 Specifications

Speed	PCLK	PIPE Data Width
2.5 GT/s	2000 MHz	8 bits*[1]
2.5 GT/s	1000 MHz	8 bits[1]
2.5 GT/s	500 MHz	8 bits[1]
2.5 GT/s	250 MHz	8 bits
2.5 GT/s	1000 MHz	16 bits*[1]
2.5 GT/s	250 MHz	16 bits[1]

Table A-3 PCLK Rates and Data Widths in PIPE 4.3, 4.4 Specifications (Continued)

Speed	PCLK	PIPE Data Width
2.5 GT/s	500 MHz	16 bits[1]
2.5 GT/s	125 MHz	16 bits
2.5 GT/s	1000 MHz	32 bits*[1]
2.5 GT/s	250 MHz	32 bits[1]
2.5 GT/s	62.5 MHz	32 bits
5.0 GT/s	2000 MHz	8 bits*[1]
5.0 GT/s	1000 MHz	8 bits[1]
5.0 GT/s	500 MHz	8 bits
5.0 GT/s	1000 MHz	16 bits*[1]
5.0 GT/s	500 MHz	16 bits[1]
5.0 GT/s	250 MHz	16 bits
5.0 GT/s	1000 MHz	32 bits*[1]
5.0 GT/s	250 MHz	32 bits[1]
5.0 GT/s	125 MHz	32 bits
8.0 GT/s	2000 MHz	8 bits[1]
8.0 GT/s	2000 MHz	16 bits*[1]
8.0 GT/s	1000 MHz	8 bits
8.0 GT/s	1000 MHz	16 bits[1]
8.0 GT/s	500 MHz	16 bits
8.0 GT/s	1000 MHz	32 bits*[1]
8.0 GT/s	500 MHz	32 bits[1]
8.0 GT/s	250 MHz	32 bits
8.0 GT/s	125 MHz	64 bits*
16.0 GT/s	2000 MHz	8 bits
16.0 GT/s	2000 MHz	16 bits*[1]
16.0 GT/s	2000 MHz	32 bits*[1]
16.0 GT/s	1000 MHz	16 bits
16.0 GT/s	1000 MHz	32 bits*[1]
16.0 GT/s	500 MHz	32 bits

Table A-3 PCLK Rates and Data Widths in PIPE 4.3, 4.4 Specifications (Continued)

Speed	PCLK	PIPE Data Width
16.0 GT/s	250 MHz	64 bits*
32.0 GT/s	4000 MHz	8 bits*
32.0 GT/s	2000 MHz	16 bits*
32.0 GT/s	1000 MHz	32 bits*
32.0 GT/s	500 MHz	64 bits*

* Indicates the combination is outside PIPE 4.3 and 4.4 specification. These customized combinations are supported only when `enable_custom_data_throttling_mode` attribute of `svt_pcie_pl_configuration` is set to 1.
[1] Indicates data throttling mode.
Gen5 combinations are supported with PIPE 4.4 only when `enable_gen5_using_pipe_4_4` attribute of `svt_pcie_pl_configuration` is set to 1.

Table A-4: VIP Supports the following PCLK rates and data widths in PIPE traditional architecture mode for PIPE 5.1 specifications:

Table A-4 PCLK Rates and Data Widths in PIPE 5.1 Traditional Architecture Mode

Speed	PCLK	PIPE Data Width
2.5 GT/s	4000 MHz	8 bits[1]
2.5 GT/s	2000 MHz	8 bits[1]
2.5 GT/s	1000 MHz	8 bits[1]
2.5 GT/s	500 MHz	8 bits[1]
2.5 GT/s	250 MHz	8 bits
2.5 GT/s	1000 MHz	16 bits*[1]
2.5 GT/s	250 MHz	16 bits[1]
2.5 GT/s	500 MHz	16 bits[1]
2.5 GT/s	125 MHz	16 bits
2.5 GT/s	1000 MHz	32 bits*[1]
2.5 GT/s	250 MHz	32 bits[1]
2.5 GT/s	62.5 MHz	32 bits
2.5 GT/s	31.25 MHz	64 bits*
5.0 GT/s	4000 MHz	8 bits[1]
5.0 GT/s	2000 MHz	8 bits[1]
5.0 GT/s	1000 MHz	8 bits[1]

Table A-4 PCLK Rates and Data Widths in PIPE 5.1 Traditional Architecture Mode (Continued)

Speed	PCLK	PIPE Data Width
5.0 GT/s	500 MHz	8 bits
5.0 GT/s	1000 MHz	16 bits*[1]
5.0 GT/s	500 MHz	16 bits[1]
5.0 GT/s	250 MHz	16 bits
5.0 GT/s	1000 MHz	32 bits*[1]
5.0 GT/s	250 MHz	32 bits[1]
5.0 GT/s	125 MHz	32 bits
5.0 GT/s	62.5 MHz	64 bits*
8.0 GT/s	4000 MHz	8 bits[1]
8.0 GT/s	2000 MHz	8 bits[1]
8.0 GT/s	2000 MHz	16 bits*[1]
8.0 GT/s	1000 MHz	8 bits
8.0 GT/s	1000 MHz	16 bits[1]
8.0 GT/s	500 MHz	16 bits
8.0 GT/s	1000 MHz	32 bits*[1]
8.0 GT/s	500 MHz	32 bits[1]
8.0 GT/s	250 MHz	32 bits
8.0 GT/s	125 MHz	64 bits*
16.0 GT/s	4000 MHz	8 bits[1]
16.0 GT/s	2000 MHz	8 bits
16.0 GT/s	2000 MHz	16 bits*[1]
16.0 GT/s	2000 MHz	32 bits*[1]
16.0 GT/s	1000 MHz	16 bits
16.0 GT/s	1000 MHz	32 bits*[1]
16.0 GT/s	500 MHz	32 bits
16.0 GT/s	250 MHz	64 bits*
32.0 GT/s	4000 MHz	8 bits
32.0 GT/s	2000 MHz	16 bits
32.0 GT/s	1000 MHz	32 bits

Table A-4 PCLK Rates and Data Widths in PIPE 5.1 Traditional Architecture Mode (Continued)

Speed	PCLK	PIPE Data Width
32.0 GT/s	500 MHz	64 bits*
* Indicates the combination is outside PIPE 5.1 specification. These customized combinations are supported only when `enable_custom_data_throttling_mode` attribute of `svt_pcie_pl_configuration` is set. [1] Indicates data throttling mode.		

Table A-5: VIP Supports following PCLK rates and data widths on the Tx path in PIPE SerDes architecture mode for PIPE 5.1 specification:

Table A-5 PCLK Rates and Data Widths on the Tx Path in PIPE SerDes Architecture Mode

Speed	PCLK	SerDes Data Width
2.5 GT/s	4000 MHz	10 bits[1]
2.5 GT/s	2000 MHz	10 bits[1]
2.5 GT/s	1000 MHz	10 bits[1]
2.5 GT/s	500 MHz	10 bits[1]
2.5 GT/s	250 MHz	10 bits
2.5 GT/s	1000 MHz	20 bits[*1]
2.5 GT/s	250 MHz	20 bits[1]
2.5 GT/s	500 MHz	20 bits[1]
2.5 GT/s	125 MHz	20 bits
2.5 GT/s	1000 MHz	40 bits*[1]
2.5 GT/s	250 MHz	40 bits[1]
2.5 GT/s	62.5 MHz	40 bits
2.5 GT/s	31.25 MHz	80 bits
5.0 GT/s	4000 MHz	10 bits[1]
5.0 GT/s	2000 MHz	10 bits[1]
5.0 GT/s	1000 MHz	10 bits[1]
5.0 GT/s	500 MHz	10 bits
5.0 GT/s	1000 MHz	20 bits*[1]
5.0 GT/s	500 MHz	20 bits[1]
5.0 GT/s	250 MHz	20 bits
5.0 GT/s	1000 MHz	40 bits*[1]

Table A-5 PCLK Rates and Data Widths on the Tx Path in PIPE SerDes Architecture Mode (Continued)

Speed	PCLK	SerDes Data Width
5.0 GT/s	250 MHz	40 bits[1]
5.0 GT/s	125 MHz	40 bits
5.0 GT/s	62.5 MHz	80 bits
8.0 GT/s	4000 MHz	10 bits[1]
8.0 GT/s	2000 MHz	10 bits[1]
8.0 GT/s	2000 MHz	20 bits*[1]
8.0 GT/s	1000 MHz	10 bits
8.0 GT/s	1000 MHz	20 bits[1]
8.0 GT/s	500 MHz	20 bits
8.0 GT/s	1000 MHz	40 bits*[1]
8.0 GT/s	500 MHz	40 bits[1]
8.0 GT/s	250 MHz	40 bits
8.0 GT/s	125 MHz	80 bits
16.0 GT/s	4000 MHz	10 bits[1]
16.0 GT/s	2000 MHz	10 bits
16.0 GT/s	2000 MHz	20 bits*[1]
16.0 GT/s	2000 MHz	40 bits*[1]
16.0 GT/s	1000 MHz	20 bits
16.0 GT/s	1000 MHz	40 bits*[1]
16.0 GT/s	500 MHz	40 bits
16.0 GT/s	250 MHz	80 bits
32.0 GT/s	4000 MHz	10 bits
32.0 GT/s	2000 MHz	20 bits
32.0 GT/s	1000 MHz	40 bits
32.0 GT/s	500 MHz	80 bits

* Indicates the combination is outside PIPE specification. VIP provides customized support for these combinations only when `enable_custom_data_throttling_mode` attribute of `svt_pcie_pl_configuration` is set.
[1] Indicates data throttling mode.

Table A-6: VIP Supports following RxCLK rates and RxWidths on the Rx path in PIPE SerDes architecture mode for PIPE 5.1 specification:

Table A-6 RxCLK rates and RxWidths on the Rx Path in PIPE SerDes Architecture Mode

Speed	RxCLK	RxWidth
2.5 GT/s	250 MHz	10 bits
2.5 GT/s	125 MHz	20 bits
2.5 GT/s	62.5 MHz	40 bits
2.5 GT/s	31.25 MHz	80 bits
5.0 GT/s	500 MHz	10 bits
5.0 GT/s	250 MHz	20 bits
5.0 GT/s	125 MHz	40 bits
5.0 GT/s	62.5 MHz	80 bits
8.0 GT/s	1000 MHz	10 bits
8.0 GT/s	500 MHz	20 bits
8.0 GT/s	250 MHz	40 bits
8.0 GT/s	125 MHz	80 bits
16.0 GT/s	2000 MHz	10 bits
16.0 GT/s	1000 MHz	20 bits
16.0 GT/s	500 MHz	40 bits
16.0 GT/s	250 MHz	80 bits
32.0 GT/s	4000 MHz	10 bits
32.0 GT/s	2000 MHz	20 bits
32.0 GT/s	1000 MHz	40 bits
32.0 GT/s	500 MHz	80 bits

B Legacy Testbench

B.1 Instantiation Models

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, 3 or 4 depending on the model used.

Note Support for legacy instantiation models will be deprecated in the future releases. It is recommended to migrate to Unified VIP single instance model. For more details, see Unified PCIe VIP Component.

Models are briefly described in Table B-1.

Table B-1 Types of Instantiation Models

Model Names	Description
svt_pcie_device_agent_serdes_x16_8g_hdl.sv svt_pcie_device_agent_serdes_x16_hdl.sv svt_pcie_device_agent_serdes_x32_8g_hdl.sv svt_pcie_device_agent_serdes_x32_hdl.sv svt_pcie_device_agent_serdes_x4_8g_hdl.sv svt_pcie_device_agent_serdes_x4_hdl.sv svt_pcie_device_agent_serdes_x8_8g_hdl.sv svt_pcie_device_agent_serdes_x8_hdl.sv.	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).
svt_pcie_device_agent_pma_x16_8g_hdl.sv svt_pcie_device_agent_pma_x16_hdl.sv svt_pcie_device_agent_pma_x32_8g_hdl.sv svt_pcie_device_agent_pma_x32_hdl.sv svt_pcie_device_agent_pma_x4_8g_hdl.sv svt_pcie_device_agent_pma_x4_hdl.sv svt_pcie_device_agent_pma_x8_8g_hdl.sv svt_pcie_device_agent_pma_x8_hdl.sv	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).
svt_pcie_device_agent_mpipe_x16_8g_hdl.sv svt_pcie_device_agent_mpipe_x16_hdl.sv svt_pcie_device_agent_mpipe_x32_8g_hdl.sv svt_pcie_device_agent_mpipe_x32_hdl.sv svt_pcie_device_agent_mpipe_x4_8g_hdl.sv svt_pcie_device_agent_mpipe_x4_hdl.sv svt_pcie_device_agent_mpipe_x8_8g_hdl.sv svt_pcie_device_agent_mpipe_x8_hdl.sv	This is a full device (RC or endpoint) model that has a standard master PIPE interface to a DUT Phy.
svt_pcie_device_agent_spipe_x16_8g_hdl.sv svt_pcie_device_agent_spipe_x16_hdl.sv svt_pcie_device_agent_spipe_x32_8g_hdl.sv svt_pcie_device_agent_spipe_x32_hdl.sv svt_pcie_device_agent_spipe_x4_8g_hdl.sv svt_pcie_device_agent_spipe_x4_hdl.sv svt_pcie_device_agent_spipe_x8_8g_hdl.sv svt_pcie_device_agent_spipe_x8_hdl.sv	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.
svt_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.
svt_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.

Table B-1 Types of Instantiation Models (Continued)

Model Names	Description
svt_phy_serdes_x*n*_model	This is a phy model used only in the compliance environment.

`svt_pcie_device_agent_mpipe_rev4_2_pclk_input_x16_8g_hdl.sv`
`svt_pcie_device_agent_mpipe_rev4_2_pclk_input_x32_8g_hdl.sv`
`svt_pcie_device_agent_mpipe_rev4_2_pclk_input_x4_8g_hdl.sv`
`svt_pcie_device_agent_mpipe_rev4_2_pclk_input_x8_8g_hdl.sv`
	This is a full device model that connects via a MPIPE interface and has support for a maximum of *n* lanes (they do not all have to be used), and where the clk is an input. Following are current instantiation model files in SystemVerilog.

`svt_pcie_device_agent_spipe_rev4_2_pclk_input_x32_8g_hdl.sv`
`svt_pcie_device_agent_spipe_rev4_2_pclk_input_x4_8g_hdl.sv`
`svt_pcie_device_agent_spipe_rev4_2_pclk_input_x8_8g_hdl.sv`
	This is a full device model that connects via a Slave Pipe interface and has support for a maximum of *n* lanes (they do not all have to be used), and where the clk is an input. Following are current instantiation model files in SystemVerilog.

Note

Models supporting 8G have "_8g" appended to the model name. 8G models should not be used without the corresponding Gen 3 license add-on.

B.2 General Steps In Picking an Instantiation Model

The first step in choosing an instantiation model is to answer a series of question/checklists as shown in the following table.

Note that due to the difference in data widths of PIPE and PMA Interfaces at 8/16G, the PCS may detect +/-1 symbol skew even if serdes_locked is asserted at the same time on all lanes."

Table B-2 Instantiation Model Checklist

Type of Interface?	
	SERDES (serial) ** Very slow for simulation
	PMA (10bit)
	PIPE • Master (SVT is MAC / DUT is Phy) • Slave (DUT is MAC / SVT is Phy)
Speed?	
	Gen 1: 2.5G
	Gen 2: 5G
	Gen 3: 8G (requires optional license)

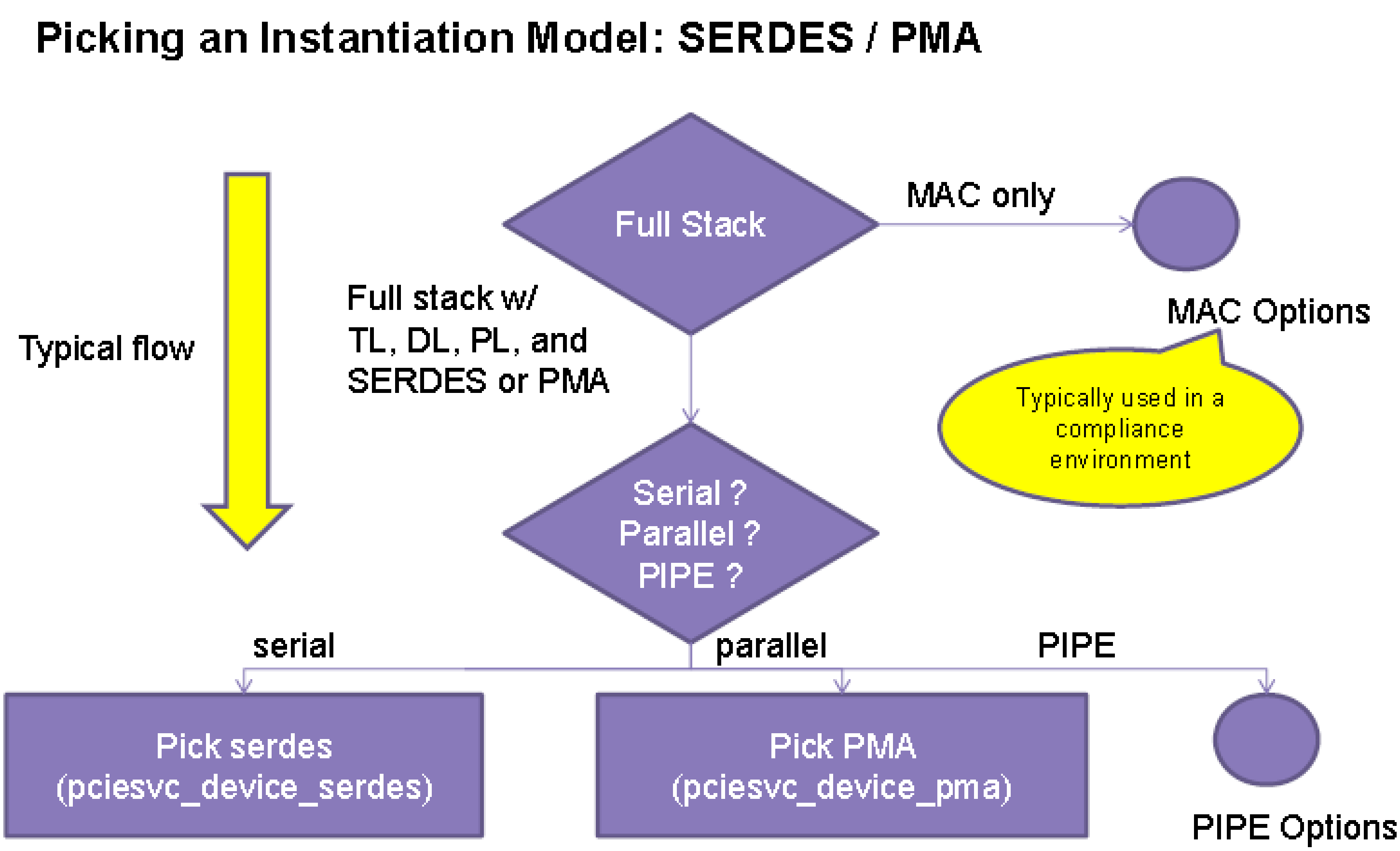
Table B-2 Instantiation Model Checklist (Continued)

Link Width?	
	x4 model (x1, x2, x4)
	x8 model (x1 - x8)
	X32 model (x1 – x32)

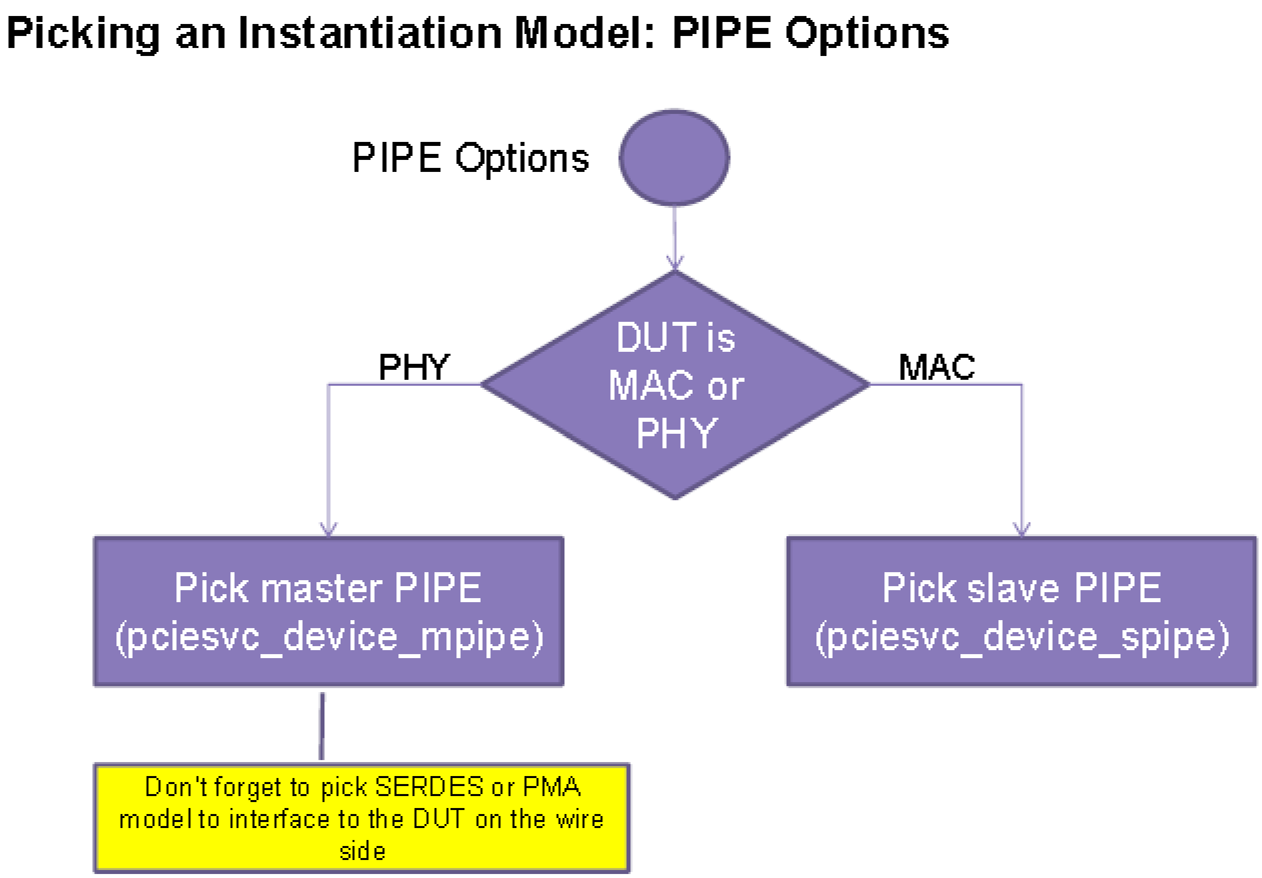
B.2.1 Flowcharts for Choosing an Instantiation Model

Note, more details follow this overview section on each instantiation model.

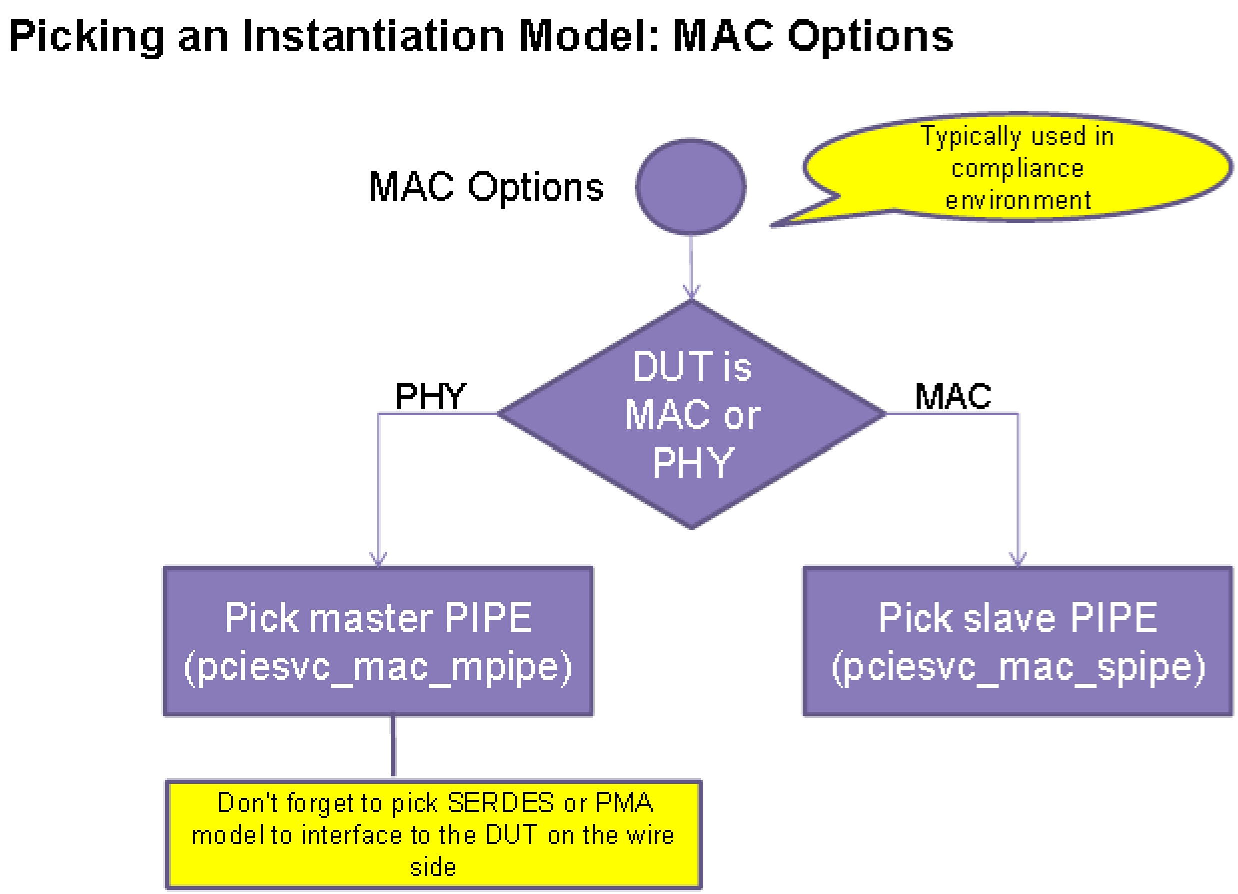
The following flowchart provides guidance on choosing the instantiation model for SERDES/PMA.



The following flow chart provides guidance on choosing your PIPE interface.



Use the following flowchart for guidance on choosing MAC Options:



B.2.2 File Location and includes for Instantiation

The usage flow for instantiation follows:

1. Ensure the include path has the following directories:

   "your_design_path"/src/sverilog/vcs

   "your_design_path"/src/verilog/vcs

   These directories are created by running dw_setup_vip when you install the models into a design directory.
2. Choose an appropriate instantiation model. Refer to Table B-1 on page 441 for a list of instantiation models.
3. Include the SystemVerilog model in your testbench.

The hdl_interconnect directory in our UVM examples show how to instantiate and connect pcie_svt models. The intermediate example has a directory called hdl_interconnect which has a copy of all the instantiation models used in the example. It is located at:

“your_design_path”/examples/sverilog/pcie_svt/tb_pcie_svt_uvm_intermediate_sys/hdl_interconnect

The hdl_interconnect directory in the intermediate example has the following files to use for interconnect:

- svt_pcie_8g_device_pipe_x4.sv
- svt_pcie_8g_device_pma_x4.sv
- svt_pcie_8g_device_serdes_x4.sv
- svt_pcie_device_pipe_x4.sv
- svt_pcie_device_pma_x4.sv
- svt_pcie_device_serdes_x4.sv

The file we are interested in is:

svt_pcie_8g_device_pipe_x4.sv

The path to the svt_pcie_device_agent_spipe_x4_8g_hdl module is in the directory:

"your_design_path"/src/sverilog/vcs.

This directory your_design_path"/src/sverilog/vcs contains all the SystemVerilog instantiation files. Note that there are Verilog instantiation models in the directory `"your_design_path"/src/verilog/vcs.`

While the SystemVerilog instantiation models reference the *.v files, Synopsys recommends only use the *sv instantiation files. The Verilog *.v files are useful for the signal names used in connecting the model.

B.3 SERDES Interface

If the serial interface is used, the SERDES layer will be provided by the VIP. The instantiation model is named svt_pcie_device_agent_serdes_x[4,8,16,32]_[8g]_hdl.

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.

Figure B-1 shows the layers supported by the PCIe VIP when using the serial interface along with the DUT serial interface.

Figure B-1 PCIe VIP serial interface layers

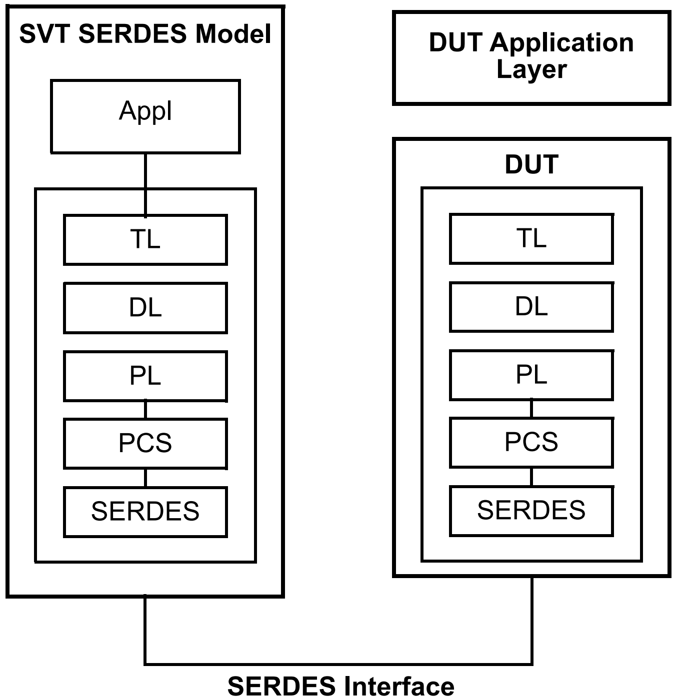


Figure B-2 SERDES module ports

Figure B-2 shows the SERDES module ports.

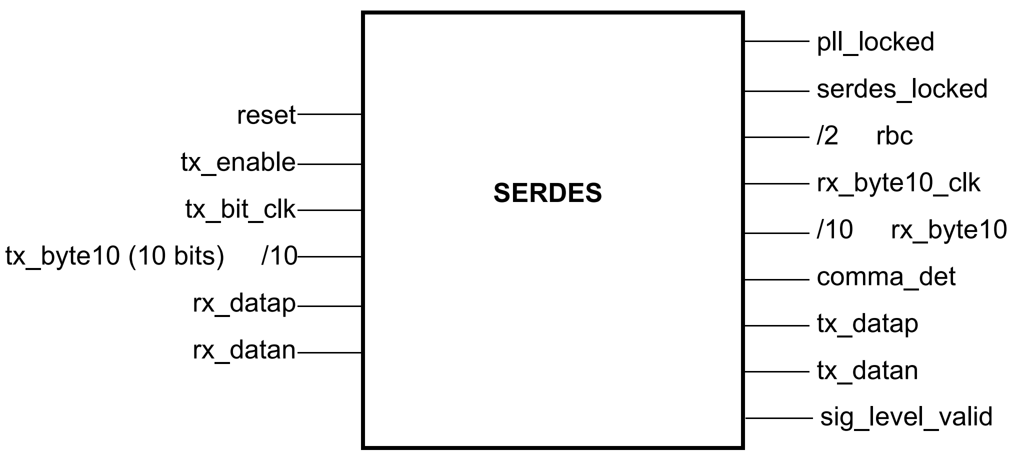


Table B-3 svt_pcie_device_agent_model SERDES ports

Model Interface Type	Port Name	I/O	Description
SERDES Interface.	Reset phy*n*_rx_datap phy*n*_rx_datan phy*n*_tx_datap phy*n*_tx_datan	I I I I	Reset to the VIP model is active high. Reset must be deasserted at time 0 such that a posedge is seen by the VIP. Reset should be asserted at 1ns and held in reset for at least 100 ns. Once deasserted, it should remain deasserted for the remainder of the simulation. Serial datap in to VIP Serial datan in to VIP serial datap out of VIP serial datan out of VIP

B.4 PMA Interface

The PCIe VIP supports a 10-bit parallel interface. The 10-bit quantity produced by the model is the result of 8b/10b encoding as defined by the PCI Express Base Specification. This is the data that would be then serialized and put onto the physical link. The 10-bit interface is fully compliant with the PCI Express protocol standard and can be used to verify PHY layers.

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. Figure B-3 shows the layers supported by the PCIe VIP when using the PMA along with the DUT PMA interface.

Figure B-3 PCIe VIP PMA layers

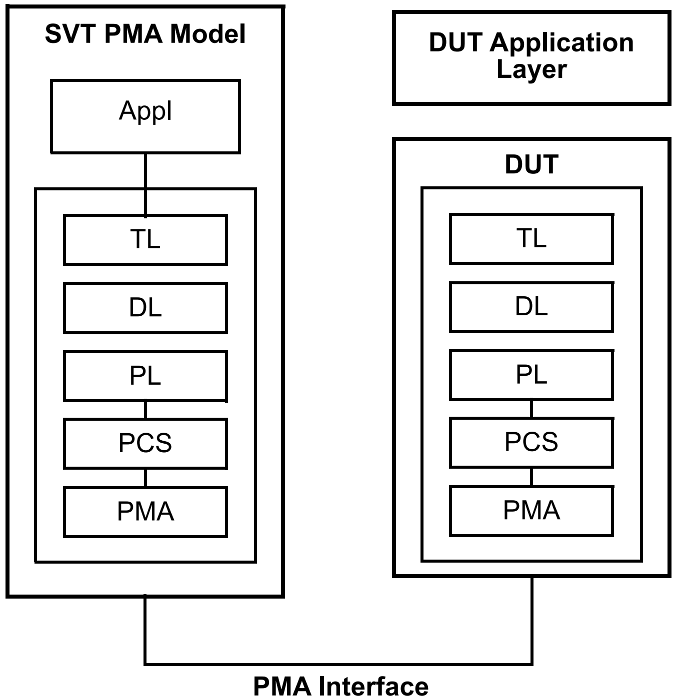
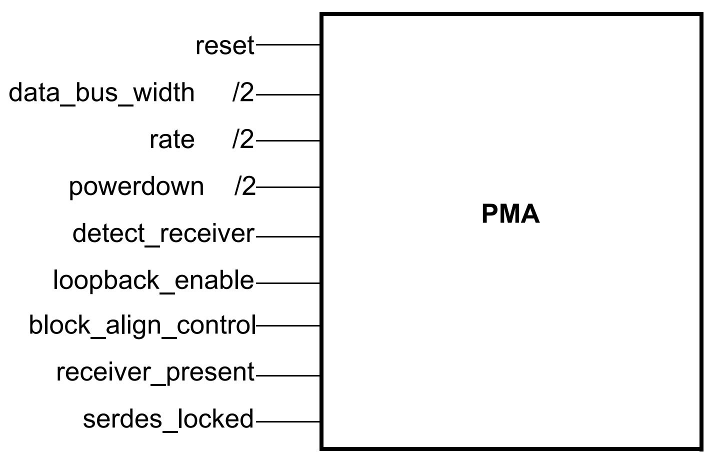


Figure B-4 PMA module ports

The PMA module ports are shown in Figure B-4



B.5 PIPE Interface, Phy VIP and MAC DUT

In addition, the transceiver model supports the PHY Interface for the PCI Express (PIPE) Architecture. This is an Intel specification that defines Media Access, Physical Coding, and Physical Media Attachment sublayers in the PCI Express PHY layer.

The term "PIPE" refers to the parallel interface between the Media Access and Physical Coding sublayers. Note the difference in interface/pin configurations between PIPE as shown in Figure B-2, and 10-bit interfaces. Models configured for a 10-bit interface ignore control pins, and use only 10 pins of the data lines txdata and rxdata. Contact Intel for instructions on obtaining the PHY Interface for the PCI Express TM Architecture specification.

The following versions of the PIPE interface are supported:

2.1

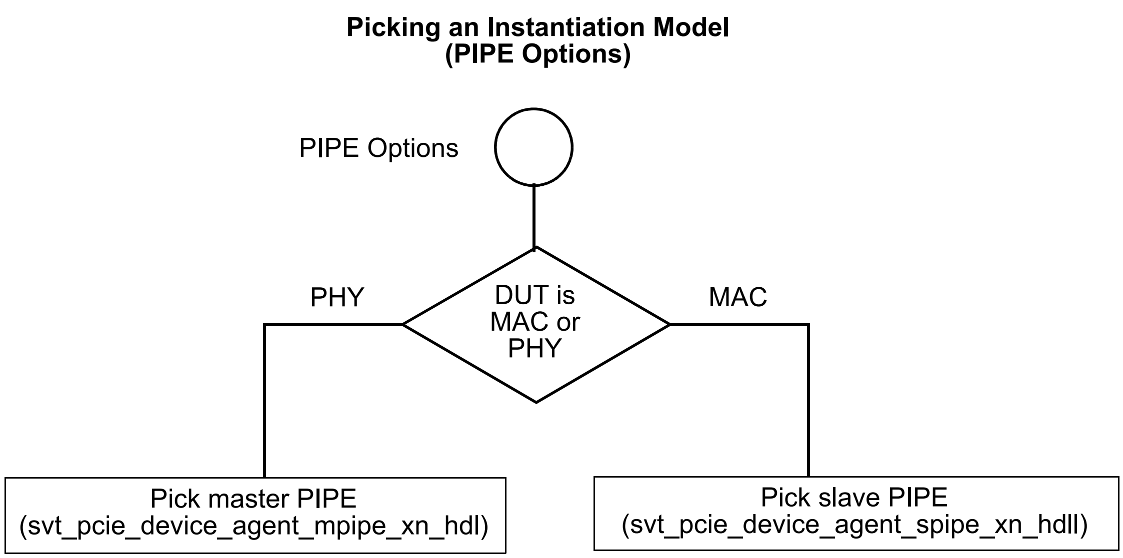
3.0

4.0

The model supports both 8-bit and 16-bit implementations of the PIPE, and can be used to support one of the following PIPE modes (see Figure B-5):

- PHY PIPE (SPIPE) mode: In this mode, the model acts like a PIPE-compliant PHY and supports the complete interface.
- MAC PIPE (MPIPE) mode: In this mode, the model acts like a PIPE-compliant MAC and supports the complete interface.

Figure B-5 Instantiation model PIPE options



You set the PIPE version in the `svt_pcie_device_configuration` class. The settings are:

- For PIPE 2 (Gen2 version of the PCIe spec):

```
svt_pcie_device_configuration::pipe_spec_ver==svt_pcie_device_configuration::
PIPE_SPEC_VER_2
```

- For PIPE 3 or 4 (Gen 3 version of the PCIe spec):

```
==svt_pcie_device_configuration::PIPE_SPEC_VER_4
```

The model may be configured to support the PIPE's interface as either a PIPE compliant PHY (PHY PIPE mode), or as a PIPE compliant MAC (MAC PIPE mode). The PHY PIPE mode and MAC PIPE mode are enabled by selecting the correct instantiation model, either the svt_pcie_device_agent_mpipe or the svt_pcie_device_agent_spipe. This diagram should help to select the correct instantiation model once the DUT configuration is known.

B.5.1 Using ifdefs with PIPE 4.0 To Select Additional Signals

Two "`ifdef"s that are used to by the PIPE 4.0 (and above) instantiation models ("InstantiationModels/svt_pcie_device_agent_*pipe_x*_8g.sv") to select additional interface signals for PIPE 4.2 and PIPE 4.3.. These "`ifdef"s are:

- **`ifdef PCIESVC_PIPE_SPEC_VER_GTR_4_0:** This "define" selects the additional signals associated with PIPE 4.2 versus PIPE 4.0. Signals such as "rx_eq_in_progress_0" (through _n), "lf_0" (through _n), "fs_0" (through _n). It is also used to determine the "PIPE_SPEC_VER" assigned to the "pl0" PHY instance in the "InstantiationModels/svt_pcie_device_agent_mpipe_x*_8g.sv" models.
- **`ifdef PCIESVC_PIPE_SPEC_VER_4_3:** This "define" selects the additional signals associated with PIPE 4.3 versus PIPE 4.2. Signals such as "async_power_change_ack" and "powerdown[3:0] (versus [2:0]). It is also used with "`ifdef PCIESVC_PIPE_SPEC_VER_GTR_4_0" to determine whether "4.2" or 4.3" is assigned to "PIPE_SPEC_VER" that is then assigned to the "pl0" PHY instance in the "InstantiationModels/svt_pcie_device_agent_*pipe_x*_8g.sv" models.
- **`ifdef PCIESVC_PIPE_SPEC_VER_4_4:** This "define" selects the additional signals associated with PIPE 4.4 versus PIPE 4.3. Signals such as "sris_enable", "rx_eidetect_disable", "tx_commonmode_disable" signals and enhanced width of "local_preset_index" signal. It is also used with "`ifdef PCIESVC_PIPE_SPEC_VER_GTR_4_0" and "`ifdef PCIESVC_PIPE_SPEC_VER_4_3" to determine whether "4.3" or 4.4" is assigned to "PIPE_SPEC_VER" that is then assigned to the "pl0" PHY instance in the "InstantiationModels/svt_pcie_device_agent_*pipe_x*_8g.sv" models.

As a result:

- PIPE 4.0 usage would have neither of the above two "`ifdef"s defined.
- PIPE 4.2 usage would have "PCIESVC_PIPE_SPEC_VER_GTR_4_0" defined.
- PIPE 4.3 usage would have "PCIESVC_PIPE_SPEC_VER_GTR_4_0" and "PCIESVC_PIPE_SPEC_VER_4_3" defined.
- PIPE 4.4 usage would have "PCIESVC_PIPE_SPEC_VER_GTR_4_0", "PCIESVC_PIPE_SPEC_VER_4_3" and "PCIESVC_PIPE_SPEC_VER_4_4" defined.

B.5.2 Picking an Instantiation Model (SPIPE Options)

When enabled to act as a PHY, the model supports validation of the MAC interface to a user's DUT. This is shown in Figure B-6. Note on the right side of the diagram that the PCIe transceiver acts not only as the DUT's PHY, but that you can drive the PHY through the model, which also acts as endpoint connected to the PHY from a system point of view.

Figure B-6 PCIe transceiver acting as a DUT's PHY to test the MAC

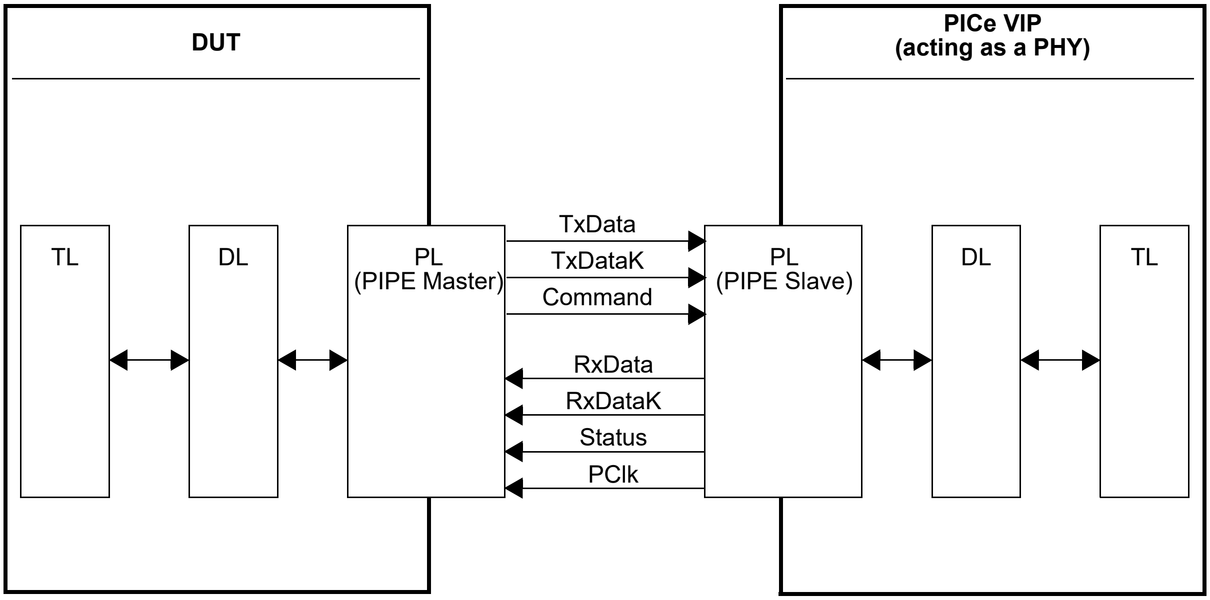

DUT
PICe VIP
(acting as a PHY)
TL
DL
PL
(PIPE Master)
TxData
TxDataK
Command
RxData
RxDataK
Status
PClk
PL
(PIPE Slave)
DL
TL

Table B-4 svt_pcie_device_agent_model SPIPE ports

Model Interface Type	Port Name	I/O	Description
“Slave” PIPE Version 3 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. PIPE 3 models do not have a _8g postfix in their file name.	reset	I	Reset to the VIP model is active high. Reset must be deasserted at time 0 such that a posedge is seen by the VIP. Reset should be asserted at 1ns and held in reset for at least 100 ns. Once deasserted, it should remain deasserted for the remainder of the simulation.
	attached_pipe_reset_*n*	I	PIPE reset signal from the MAC. **Note**: Signal must be de-asserted after reset signal.
	pipe_clk	IO	The pipe_clk signal is an input signal only by using the the Define PCIESVC_PIPE_PCLK_AS_PHY_INPUT. Use the following to make it an input signal: +define+PCIESVC_PIPE_PCLK_AS_PHY_INPUT Otherwise by default the pipe_clk signal is an output of the PHY 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_controll	I	Controls slave block alignment. NOTE: PIPE3 and above.
	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	O	Per lane receive data into the DUT. Supports 8, 16, and 32bit bus widths
	attached_rx_data_k	O	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	attached_rx_status[2:0]	O	Per lane receiver status. See PIPE spec for encodings.
	attached_rx_valid	O	Per lane li nd ica ti on that data coming into the DUT is valid and symbol lock has been achieved.
	attached_rx_elec_idle	I/O	Per lane indication that electrical idle is being received into the DUT

Table B-4 svt_pcie_device_agent_model SPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
“Slave” PIPE Version 3 Interface (continued)	attached_invert_rx_polarity	I	Per lane indication to perform polarity inversion on data into the DUT
	attached_tx_data[31:0]	I	Per lane transmit data out from the DUT.
	attached_tx _data_k[3:0]	I	Per lane transmit_control from the DUT.
	attached_tx_ei_code[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	I	Per lane transmit compliance pattern enable from the DUT
	attached_tx_elect_idle	I	Per lane transmit electrical idle/transmit enable from the DUT Note: Any unused ports must be tied to a 1.
“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	Reset to the VIP model is active high. Reset must be deasserted at time 0 such that a posedge is seen by the VIP. Reset should be asserted at 1ns and held in reset for at least 100 ns. Once deasserted, it should remain deasserted for the remainder of the simulation.
	attached_pipe_reset_n	I	PIPE reset signal from the MAC. **Note**: Signal must be de-asserted after reset signal.
	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: PIPE3 and above.
	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.

Table B-4 svt_pcie_device_agent_model SPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
	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	O	Per lane receive data into the DUT. Supports 8, 16, and 32bit bus widths.
	attached_rx_data_k	O	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	attached_rx_status[2:0]	O	Per lane receiver status. See PIPE spec for encodings.
	attached_rx_valid	O	Per lane indication that data coming into the DUT is valid and symbol lock has been achieved.
	attached_rx_data_valid	O	Per lane indication that data coming into the DUT is valid. Used primarily for rate matching. NOTE: PIPE3 only.
	attached_rx_elec_idle	IO	Per lane indication that electrical idle is being received into the DUT
	attached_rx_start_block	O	Per lane indication that data transmitted by the DUT is the first byte of data in the block. NOTE: PIPE3 only.
	attached_rx_sync_header[1:0]	O	Per lane block sync header. NOTE: PIPE3 only.
	attached_invert_rx_polarity	I	Per lane indication to perform polarity inversion on data into the DUT.
	attached_tx_data[31:0]	I	Per lane transmit data out from the DUT.
	attached_tx _data_k[3:0]	I	Per lane transmit_control from the DUT.
	attached_tx_ei_code[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	I	Per lane transmit compliance pattern enable from the DUT

Table B-4 svt_pcie_device_agent_model SPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
	attached_tx_elect_idle	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	I	Per lane indication that data transmitted to the SVC is valid. Used primarily for rate matching. NOTE: PIPE3 only.
	attached_tx_start_block	O	Per lane indication that data transmitted to the SVC is the first byte of data in the block. NOTE: PIPE3 only.
	attached_tx_sync_header[1:0]		Per lane block sync header. NOTE: PIPE3 only.
	attached_local_tx_preset_coefficients[17:0]		Coefficients returned from a coefficient lookup request.
	attached_link_eval_feedback_figure_of_merit[7:0]	O	Figure of merit value from a link equalization evaluation request.
	attached_link_eval_feedback_direction_change[5:0]	O	Coefficient direction feedback from rx link eval request.
	attached_local_fs[5:0]	O	Provides the FS value for the phy.
	attached_local_lf[5:0]	O	Provides the LF value for the phy.
	attached_local_tx_coefficients_valid	O	Indicates that the values in local_tx_preset_coefficients[] are valid.
	attached_tx_deemph[17:0]	I	Selects transmitter deemphasis.
	attached_local_preset_index[4:0]	I	Index for the local phy preset coefficients requested by the mac.
	attached_rx_preset_hint[2:0]	I	RX preset hint for the receiver.
	attached_get_local_preset_coefficients	I	Request a preset to coefficient mapping lookup.
	attached_rx_eq_eval	I	Request from the mac for the receiver to perform an equalization evaluation.
	attached_invalid_request	I	Indicates the link eval feedback requested was out of range.

B.6 PIPE Interface, MAC VIP and PHY DUT (MPIPE)

When enabled to act as a MAC, the model supports validation of the PIPE or SERDES interface (encoding, serialization, and so on) to a user's PHY. This is shown in Figure B-7.

Figure B-7 PCIe transceiver acting as a DUT's MAC

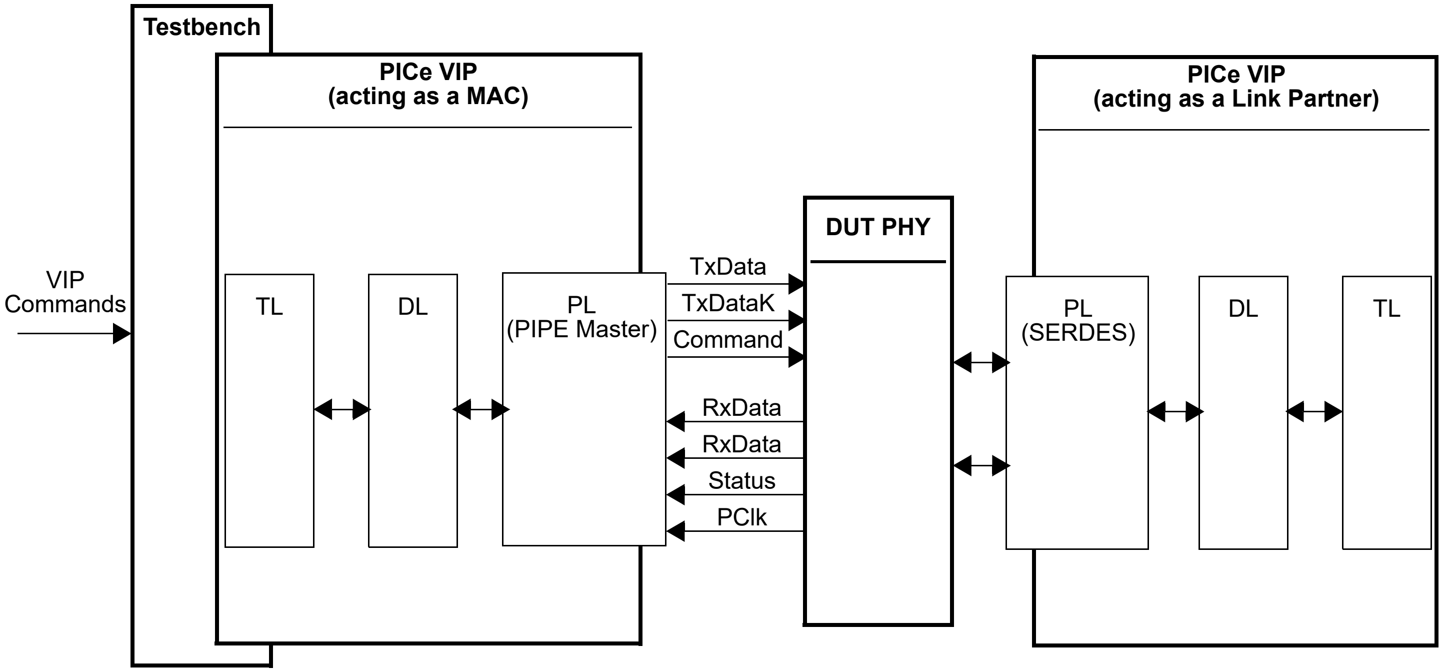

Testbench
PICe VIP
(acting as a MAC)
VIP
Commands
TL
DL
PL
(PIPE Master)
TxData
TxDataK
Command
RxData
RxData
Status
PClk
DUT PHY
PICe VIP
(acting as a Link Partner)
PL
(SERDES)
DL
TL

Table B-5 svt_pcie_device_agent_model MPIPE ports

Model Interface Type	Port Name	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	Reset to the VIP model is active high. Reset must be deasserted at time 0 such that a posedge is seen by the VIP. Reset should be asserted at 1ns and held in reset for at least 100 ns. Once deasserted, it should remain deasserted for the remainder of the simulation.
	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 PIPE2, only bit 0 is used.
	data_bus_width[1:0]	I	Indicates the per lane PIPE data bus width
	rx_data[31:0]	I	Per lane receive data into the VIP. Supports 8, 16, and 32bit bus widths
	rx_data_k[3:0]	I	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	rx_status[2:0]	I	Per lane receiver status. See PIPE spec for encodings. Note: Any unused ports must be tied to a 0.
	rx_valid	I	Per lane indication that data coming into the VIP is valid and symbol lock has been achieved.
	rx_data_valid	I	Per lane indication that data coming into the VIP is valid. Used primarily for rate matching. NOTE: PIPE3 only.

Table B-5 svt_pcie_device_agent_model MPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
	rx_elec_idle	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	O	Per lane indication to perform polarity inversion on data into the VIP
	tx_data[31:0]	O	Per lane transmit data out from the VIP
	tx _data_k[3:0]	O	Per lane transmit_control from the VIP
	tx _ei_code[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	O	Per lane transmit compliance pattern enable from the VIP
	tx_elect_idle	O	Per lane transmit electrical idle/transmit enable from the VIP
"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	Reset to the VIP model is active high. Reset must be deasserted at time 0 such that a posedge is seen by the VIP. Reset should be asserted at 1ns and held in reset for at least 100 ns. Once deasserted, it should remain deasserted for the remainder of the simulation.
	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_control	O	Controls slave block alignment. NOTE: PIPE3 and above.
	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.

Table B-5 svt_pcie_device_agent_model MPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
	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 PIPE2, only bit 0 is used.
	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[31:0]	I	Per lane receive data into the SVC. Supports 8, 16, and 32bit bus widths.
	rx_data_k[3:0]	I	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	rx_status[2:0]	I	Per lane receiver status. See PIPE spec for encodings. Note: Any unused ports must be tied to a 0.
	rx_valid	I	Per lane indication that data coming into the SVC is valid and symbol lock has been achieved.
	rx_data_valid	I	Per lane indication that data coming into the SVC is valid. Used primarily for rate matching.
	rx_elec_idle	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	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	I	Per lane block sync header.
	invert_rx_polarity	O	Per lane indication to perform polarity inversion on data into the SVC.
	tx_data[31:0]	O	Per lane transmit data out from the SVC.
	tx _data_k[3:0]	O	Per lane transmit_control from the SVC
	tx _ei_code[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	O	Per lane transmit compliance pattern enable from the SVC.

Table B-5 svt_pcie_device_agent_model MPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
	tx_elect_idle	O	Per lane transmit electrical idle/transmit enable from the SVC.
	tx_data_valid	O	Per lane indication that data transmitted by the SVC is valid. Used primarily for rate matching.
	tx_start_block	O	Per lane indication that data transmitted by the SVC is the first byte of data in the block.
	tx_sync_header[1:0]	O	Per lane block sync header.
	local_tx_preset_coefficients[17:0]	I	Coefficients returned from a coefficient lookup request.
	link_eval_feedback_figure_of_merit[7:0]	I	Figure of merit value from a link equalization evaluation request.
	link_eval_feedback_direction_change[5:0]	I	Coefficient direction feedback from rx link eval request.
	local_fs[5:0]	I	Provides the FS value for the phy.
	local_lf[5:0]	I	Provides the LF value for the phy.
	local_tx_coefficients_valid	I	Indicates that the values in local_tx_preset_coefficients[] are valid.
	tx_deemph[17:0]	O	Selects transmitter deemphasis.
	local_preset_index[4:0]	O	Index for the local phy preset coefficients requested by the mac.
	rx_preset_hint[2:0]	O	RX preset hint for the receiver.
	get_local_preset_coefficients	O	Request a preset to coefficient mapping lookup.
	rx_eq_eval	O	Request from the mac for the receiver to perform an equalization evaluation.
	invalid_request	O	Indicates the link eval feedback requested was out of range

B.7 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, you should use the device model to take advantage of the Application Layer. MAC models do not include the application layer and its associated scoreboarding.

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

B.7.1 Model Wire Interface Options

Table B-3 defines the port list for each interface configuration of the svt_pcie_device_agent_model instantiation.

Note
All unused input ports must be tied to 0, with the exception of *_elec_idle Any unused *_elec_idle input ports must be tied to 1.

B.8 Configuring the PIPE Data Bus Width

B.8.1 PIPE 2.1/3

In PIPE 2.1 and PIPE 3.0 spec versions, the MAC does not specify the per-lane width and rate of the PIPE clock. For *mpipe* models the width will take the width reflected on the data_bus_signal. See Table 7-3 for information about the data_bus_signal. For *spipe* models, the width must be configured for each supported speed. These settings are made in the svt_pcie_pl_configuration PL configuration class. The settings are:

- pipe_width[0] : Gen 1 data bus width
- pipe_width[1]: Gen 2 data bus width
- pipe_width[2]: Gen 3 data bus width

Supported widths are (pipe_width_enum): PIPE_8_BITS, PIPE_16_BITS, PIPE_32_BITS

B.8.2 PIPE4 and Above

With PIPE 4, the per-lane data bus width and rate of the PIPE clock are determined by the pclk_rate and width PIPE signals. For *mpipe* models the pclk_rate and width signals are set within the PL configuration class: svt_pcie_pl_configuration. The settings are:

- pclk_rate[0]: Gen 1 pclk_rate
- pclk_rate[1]: Gen 2 pclk_rate
- pclk_rate{2]: Gen 3 pclk_rate
- pclk_rate{3]: Gen 4 pclk_rate (applicable for PIPE 4.2 and above)
- pclk_rate{4]: Gen 5 pclk_rate (applicable for PIPE 4.4 and above)

  Options are (pclk_rate_enum): PCLK_31_25_MHZ, PCLK_62_5_MHZ, PCLK_125_MHZ, PCLK_250_MHZ, PCLK_500_MHZ, PCLK_1000_MHZ, PCLK_2000_MHZ, PCLK_4000_MHZ
- pipe_width[0]: Gen 1 data bus width
- pipe_width[1]: Gen 2 data bus width
- pipe_width[2]: Gen 3 data bus width
- pipe_width[3]: Gen 4 data bus width (applicable for PIPE 4.2 and above)
- pipe_width[4]: Gen 5 data bus width (applicable for PIPE 4.4 and above)

  Supported widths are (pipe_width_enum):

  •PIPE_8_BITS

  •PIPE_16_BITS

- PIPE_32_BITS
- PIPE_64_BITS (applicable for 2.5G [for PIPE Specification Version 5.1 and later], 5G [for PIPE Specification Version 5.1 and later], 8G, 16G and 32G link speeds when the macro `SVT_PCIE_PIPE_DATA_WIDTH` is set to 64)

Note
PCIe VIP does not support 12 and 20 symbol size SKIP OS for 64-bit PIPE width at Gen3 and higher data rates.

For *spipe* models connect the pclk_rate and width PIPE signals to the MAC.

Note: For SERDES or PMA models, do not set the pclk_rate or pipe_width.

If you want to reconfigure svt_pcie_pl_configuration::pclk_rate or svt_pcie_pl_configuration::pipe_width values, the recommended time is when LTSSM is in Detect.Quiet and pipe_reset_n is asserted.

Following are the two methods to reconfigure pclk_rate or pipe_width:

1. Set svt_pcie_pl_configuration::enable_pipe_reset_n_assertion_in_detect_quiet = 1 and direct LTSSM to Detect.Quiet.
2. Use svt_pcie_pl_service::PIPE_INJECT_RESET to assert Reset#.

B.9 Compile-Time Parameter Settings

Several parameters are set at the model. They typically 'trickle-down' to the individual layers.

Table B-6 lists the Verilog parameters that you set at compile time. They are not runtime changeable. They are not accessible from the UVM environment.

Table B-6 Parameters Set in the Model

Parameter Name	Type	Range	Default Value	Description
NUM_PMA_ INTERFACE_BITS	Integer	10, 16, 32, 64, 128	10	Number of bits on the PMA interface
PCIE_SPEC_VER	Real	1.1, 2.0, 2.1, 3.0	PCIE_SPEC_VER_3_0	See *Include/pciesvc_parms.vp: PCIE_SPEC_VER_** Note: Set this here and not in the individual layers.
HIERARCHY_ NUMBER	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.
DISPLAY_NAME	String		"pcie_svt_device_agent_xx_yy_hdl" (Specific to each model).	String prefixed to messages to display in the output log.

Example B-1 Setting Verilog parameters at compile time

```
// This is a PIPE SLAVE model, so inputs and outputs are reversed from a normal PIPE
pcie_svt_device_agent_spipe_x4_hdl #(.DISPLAY_NAME("root0."),
    .DEVICE_IS_ROOT(1)) root0(.reset
        (reset),
        .pipe_clk
```

```
        (pipe_clk),
        // shared signals
        .attached_pipe_reset_n
        (endpoint0_pipe_reset_n),
        // inputs
        ...
        .attached_tx_compliance_3
        endpoint0_tx_compliance_3),
        .attached_tx_elec_idle_3
        (endpoint0_tx_elec_idle_3
    );
```

B.10 Instantiating 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 doing automatic checking of data, you also need to defparam those shadows respectively.

Each hierarchy number should be unique. It is recommended that you make it match the instantiation number, for ease of reference. The HIEARCHY_NUMBER is used in conjunction with the SHADOW_MEMORY checking. For example, in the above instantiation, add the HIERARCHY_NUMBER as follows:

```
pcie_svt_device_agent_spipe_x4_hdl #(.DISPLAY_NAME("root0."),
.DEVICE_IS_ROOT(1), .HIEARARCHY_NUMBER(1)) root0(.reset
```

B.11 Model Configuration Overview

Configuration tasks are used after reset is deasserted from the pcie_svt_device_agent_model.

Recommended tasks to be set prior to usage are described in Table B-7. The details of the task may be found in the respective section of the documentation.

Table B-7 Configuration tasks to set before usage

Name	Layer
svt_pcie_tl_configuration:init_[cpl/np/p]_[data/hdr]_tx_credits	Transaction Layer
svt_pcie_tl_service_set_vc_en_sequence	Transaction Layer
svt_pcie_tl_service::service_type_enum=SET_TRAFFIC_CLASS_MAP	Transaction Layer
svt_pcie_tl_service::service_type_enum=ADD_MEM_ADDR_APPL_ID_MAP_ENTRY	Transaction Layer
svt_pcie_tl_service::service_type_enum=ADD_IO_ADDR_APPL_ID_MAP_ENTRY	Transaction Layer
svt_pcie_dl_service_set_link_en_sequence	Data Link Layer
svt_pcie_pl_configuration::set_link_speed_values()	Physical Layer
svt_pcie_pl_configuration::set_link_width_values()	Physical Layer

B.12 Turning Off Unused Lanes

When configured to operate in any lower link width configuration, the model asserts the TxCompliance and TxElecidle signals to turn OFF the unused lanes of the link. Use the following member for the turn off feature:

```
rand bit enable_pipe_reset_n_assertion_in_detect_quiet = 0;
```

This attribute controls automatic pipe_resetn assertion in detect.quiet to test the switching to the P1 method for lane turn off feature.

B.13 PIPE CLK as Input to the SPIPE Interface

With the 4.2 version of the PHY PIPE specification, Synopsys added support for using the pipe_clk as an input to the SPIPE model. To use the pipe_clk as an input, do the following.

1. Instantiate the correct model. Use one of these models depending on the number of lanes you require:

   - svt_pcie_device_agent_spipe_rev4_2_pclk_input_x16_8g_hdl.sv

   - svt_pcie_device_agent_spipe_rev4_2_pclk_input_x32_8g_hdl.sv

   - svt_pcie_device_agent_spipe_rev4_2_pclk_input_x4_8g_hdl.sv

   - svt_pcie_device_agent_spipe_rev4_2_pclk_input_x8_8g_hdl.sv

2. Define PCIESVC_PIPE_PCLK_AS_PHY_INPUT. Use:

   +define+PCIESVC_PIPE_PCLK_AS_PHY_INPUT

The previous models can be found in the following installation directory:

/your_DW_install/vip/svt/pcie_svt/latest/verilog/src/vcs

B.13.1 Parameters

The following table describes the PHY parameters used to control the PCLK_Rate and Width remap:

Table B-8 PCLK_Rate and Width Remap Parameters

Parameter Name	Settable VAR*	Default Value	Description
SPIPE_REPLACE_WIDTH_PCLK_RATE_WITH_TABLE_VALUES	YES	0	When set to 1, the SPIPE MAC `PipeSlaveControl` task will drive its `internal_pclk_rate_o[2:0]` and `internal_width_o[1:0]` pins from the `spipe_pclk_rate_2_5g`, ..., `spipe_pclk_rate_32g` and `spipe_width_2_5g`, ..., `spipe_width_32g` variables based on the `rate_i` inputs. When set to 0, the `pclk_rate_i[2:0]` input will be driven on the `internal_pclk_rate_o[2:0]` and the `width_i[1:0]` will be driven on the `internal_width_o[1:0]`output pins (this is the normal case).

B.13.2 Functions

Table B-9 Remap PCLK_Rate and Width Variable Function

Function Name	Arguments	I/O	Description
ConfigureSpipeWidthPclkRate Table Sets the width of per lane data bus and pipe clock rate for a given link speed. This task is valid for SPIPE PIPE 4 (and above) models only. Default operation is 8 bits for all speeds.	pipe_rate_i (2)	I	Data rate that is to be configured. You must use the `PIPE_DATA_RATE_*` parameters defined in *Verilog/Include/pciesvc_parms.vp*.
	pipe_width_i (2)	I	The PIPE width that the VIP is supposed to operate at when at the above speed. You must use the `PIPE_WIDTH_*` parameters defined in *Verilog/Include/pciesvc_parms.vp*. Note: `PIPE_WIDTH_*` is not same as `PIPE_DATA_BUS_WIDTH`. They are different PIPE signals with different values.
	pipe_pclk_ rate_i (3)	I	See Table 3-2 "PCI Express Mode - Possible PCLK rates and data widths" for all legal combinations of Rate, Width and PCLK_Rate.

B.14 PIPE Coefficient Use

Before using the PIPE preset coefficient feature of the model, you must first enable the feature with the following configuration members of the svt_pcie_pl_configuration class. The configuration members are used for both SPIPE and MPIPE.

- **enable_get_local_preset_coefficients**. Before you can use the PIPE GetLocalPresetCoefficients interface of the PHY you must enable those signals using the enable_get_local_preset_coefficients member. If you have a SERDES interface, this parameter has no effect as this interface is only present in a PIPE model (version 4.0 or greater).
  - **MPIPE model:** If the enable_get_local_preset_coefficients is a "1", then it enables the per-Lane GetLocalPresetStateMachine"to drive the per-Lane get_local_preset_coefficients/local_preset_index[4:0]" outputs and interpret the per -Lane local_tx_preset_coefficients[17:0]/local_tx_coefficients_valid inputs. If the enable_get_local_preset_coefficients is a "0", there will be no activity presented on the previous outputs nor any response to the previous inputs.
  - **SPIPE model**. If the enable_get_local_preset_coefficients is a "1", it enables the per-Lane "RespLocalTxPresetCoeffStateMachine" to interpret a get_local_preset_coefficients/local_preset_index[4:0] input request and respond with a per-Lane "local_tx_preset_coefficients[17:0]/local_tx_coefficients_valid" output. If the enable_get_local_preset_coefficients is a "0", there will be no response to any per-Lane get_local_preset_coefficients/local_preset_index[4:0] input request.
- **enable_get_local_preset_coefficients_checking**. When set to a “1”, you enable the model to check the returned preset coefficient values from the PHY against the values in its “preset mapping table”. The preset to coefficients mapping table is used to map received preset requests to coefficients for use in local transmitter settings.

Table B-10 PIPE GetLocalPresetCoefficients Signals

Name	Direction	Active Level	Description
GetLocalPresetCoeffcients	Input	High	A MAC holds this signal high for one PCLK cycle requesting a preset to co-efficient mapping for the preset on LocalPresetIndex[3:0] to coefficients on LocalTxPresetCoefficient[17:0] Maximum Response time of PHY is 128 nSec. Note. A MAC can make this request any time after reset. Note. After a local preset coefficient request a MAC could assert GetLocalPresetCoeffcients again as soon as the next PCLK after LocalTxCoefficientsValid deasserts.
LocalPresetIndex [3:0]	Input	NA	Index for local PHY preset coefficients requested by the MAC The preset index value is encoded as follows: 0000b – Preset P0. 0001b – Preset P1. 0010b – Preset P2. 0011b – Preset P3. 0100b – Preset P4. 0101b – Preset P5. 0110b – Preset P6. 0111b – Preset P7. 1000b – Preset P8. 1001b – Preset P9. 1010b – Preset P10. 1011b – Reserved 1100b – Reserved 1101b – Reserved 1110b – Reserved 1111b – Reserved.
LocalTxCoefficientsValid	Output	High	A PHY holds this signal high for one PCLK cycle to indicate that the LocalTxPresetCoefficients[17:0] bus correctly represents the coefficients values for the preset on the LocalPresetIndex bus.
LocalTxPresetCoefficients[17:0]	Output	NA	These are the coefficients for the preset on the LocalPresetIndex[3:0] after a GetLocalPresetCoeffcients request: [5:0] C-1 [11:6] C0 [17:12] C+1 Valid on assertion of LocalTxCoefficientsValid. The MAC will reflect these coefficient values on the TxDeemph bus when MAC wishes to apply this preset.

You fill in the preset mapping table using PHY layer configuration data member preset_to_coefficients_mapping_table[16]. Its declaration is as follows:

```
rand bit[17:0] preset_to_coefficients_mapping_table[16] = '{ 18'h0c900, 18'h0c900,
18'h0c900, 18'h0c900, 18'h0c900, 18'h0c900, 18'h0c900, 18'h0c900,  18'h0c900,
18'h0c900, 18'h0c900, 18'h0c900, 18'h0c900, 18'h0c900, 18'h0c900, 18'h0c900};
```

When the model is acting as an SPIPE, you must also set the min and max delay value from when the model will respond by asserting values on the local_tx_preset_coefficients_<n> set of signals. The parameters you must set are:

- **min_spipe_preset_coefficients_delay**. Default value is 4 time units. If performing as the PIPE slave for the GetLocalPresetCoefficients interface, this is the minimum number of "PClk cycles that a 'per Lane' "Respond Local Tx Preset Coefficients" state machine will wait in the RESP_LOCAL_TX_COEFF_DELAY state before asserting the "LocalTxCoefficientsValid" and "LocalTxPresetCoefficients[17:0]" PIPE interface signals and proceeding to completion. Selection is random between MIN_SPIPE_PRESET_COEFFICIENTS_DELAY and MAX_SPIPE_PRESET_COEFFICIENTS_DELAY.
- **max_spipe_preset_coefficients_delay**. Default value is 8 time units. This is the "Max" value that is paired with MIN_SPIPE_PRESET_COEFFICIENTS_DELAY above..

Attention The model generates a random value which is between the min and max values set by the previous configuration members.

B.14.1 PHY PIPE GetLocalPresetCoefficients Interface ASCII Signals

The following table shows ASCII signals for the PHY PIPE GetLocalPresetCoefficients interface.

Table B-11 ASCII Signal Names for PHY PIPE GetLocalPresetCoefficients Interface.

Signal Name	Description
ascii_pipe_lane0_get_coeff_state	Current state of Lane 0's GetLocalPresetStateMachine state machine
ascii_pipe_lane1_get_coeff_state	Current state of Lane 1's GetLocalPresetStateMachine state machine
ascii_pipe_lane30_get_coeff_state	Current state of Lane 30's GetLocalPresetStateMachine state machine
ascii_pipe_lane31_get_coeff_state	Current state of Lane 31's GetLocalPresetStateMachine state machine
ascii_pipe_lane0_coeff_response_state	Current state of Lane 0's RespLocalTxPresetCoeffStateMachine state machine
ascii_pipe_lane1_coeff_response_state	Current state of Lane 1's RespLocalTxPresetCoeffStateMachine state machine
ascii_pipe_lane30_coeff_response_state	Current state of Lane 30's RespLocalTxPresetCoeffStateMachine state machine
ascii_pipe_lane31_coeff_response_state	Current state of Lane 31's RespLocalTxPresetCoeffStateMachine state machine

C Functional Coverage

The PCIe VIP provides notification routines which users can utilize for functional coverage. The notifications 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.

C.1 Enabling Functional Coverage

To enable function coverage define the macro 'SVT_PCIE_INCLUDE_AC_COVERAGE' at compile time, and set the following variables in the svt_pcie_configuration class:

```
enable_cov = 6'b111111; // Bitwise enable
```

In the enable_cov variable, bit 0 enables PIPE related functional coverage, and bits 1, 2, and 3 enable functional coverage for the Physical Layer, Data Link Layer, and Transaction Layers respectively.

C.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.

C.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.

C.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
);
```

C.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 C-1. These tasks may also be called through the SystemVerilog API.

Table C-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

Table C-1 Transaction override functions (Continued)

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

C.4 Transaction Layer

All methods and variables are declared in pciesvc_tl_fc_base, which is located 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.

C.4.1 Transaction Layer Functional Coverage

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

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

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

C.4.2 Transaction Layer Callbacks

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

Table C-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 Transaction 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

C.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 coverpoint. However, all TLP fields are updated during the UpdateTxTLP/UpdateRxTLP callbacks so that users can create their own coverage if necessary.

C.5.1 Data Link Layer Functional Coverage

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

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

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

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

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

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

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

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

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

	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
	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

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

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

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

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

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

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

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

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

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	

C.5.2 Link Layer Callbacks

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

Table C-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.vp under RECEIVED_TLP_STATUS*

Table C-5 Data Link Layer callbacks (Continued)

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 C-5 Data Link Layer callbacks (Continued)

	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	2^{nd} dword of message TLP. This would be used for vendor specific messages.
	message_dword3[31:0]	I	3^{rd} 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 C-5 Data Link Layer callbacks (Continued)

	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 C-5 Data Link Layer callbacks (Continued)

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

C.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.

C.6.1 Physical Layer Functional Coverage

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

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

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_complian ce	
	polling_active_to_polling_configura tion	
	polling_active_to_detect_quiet	
	polling_compliance_to_detect_qui et	
	polling_compliance_to_polling_acti ve	
	polling_configuration_to_configurat ion_linkwidth_start	
	polling_configuration_to_detect_qu iet	
	configuration_linkwidth_start_to_di sabled	
	configuration_linkwidth_start_to_lo opback_entry	
	configuration_linkwidth_start_to_co nfiguration_linkwidth_accept	
	configuration_linkwidth_start_to_d etect_quiet	
	configuration_linkwidth_accept_to_ configuation_lanenum_wait	
	configuration_linkwidth_accept_to_ detect_quiet	
	configuration_lanenum_accept_to_ configuration_complete	

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

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

		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_linkwidth_start	
		recovery_rcvrcfg_to_recovery_idle	
		recovery_rcvrcfg_to_configuration_linkwidth_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 C-6 Covergoups, coverpoints and bins in the Physical Layer coverage class (Continued)

		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			

C.6.2 Physical Layer Callbacks

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

Table C-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

C.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.

C.7.1 PIPE Functional Coverage

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

Table C-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

C.7.2 PIPE Interface Callbacks

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

Table C-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 C-9 PIPE callbacks (Continued)

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			

C.8 Mapping Legacy Covergroups to Corresponding New Covergroups

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
`cg_tx_dllp`	`dl_downstream_traveling_dllp_cg` `dl_upstream_traveling_dllp_cg`	Available from O-2018.06-2 release
`cg_rx_dllp`		Available from O-2018.06-2 release
`cg_tx_tlp`	`tl_downstream_traveling_tlp_cg` `tl_upstream_traveling_tlp_cg`	Available from O-2018.06-2 release
`cg_rx_tlp`		Available from O-2018.06-2 release
`cg_tx_tc_vc_mapping`	`tl_downstream_traveling_tlp_cg` `tl_upstream_traveling_tlp_cg`	Available from O-2018.06-2 release TC TLP field is covered as part of TLP coverage in upstream/downstream direction instead of user programed TC-VC mapping.
`cg_rx_tc_vc_mapping`		Available from O-2018.06-2 release TC TLP field is covered as part of TLP coverage in upstream/downstream direction instead of user programed TC-VC mapping.
`cg_tx_tlp_prefix`	`tl_tlp_prefix_cg`	Available from O-2018.06-2 release
`cg_rx_tlp_prefix`		Available from O-2018.06-2 release
`cg_tx_ipg`	`dl_tx_inter_packet_gap_cg`	Available from O-2018.06-3 release
`cg_rx_ipg`	`dl_rx_inter_packet_gap_cg`	Available from O-2018.06-3 release
`cg_max_payload_size`	`tl_downstream_traveling_tlp_cg::length_cp` `tl_upstream_traveling_tlp_cg::length_cp`	Available from O-2018.06-2 release TLP length is covered on RX & TX Path instead of programed Max Payload Size Configuration attribute
`cg_max_payload_size_cross_rate_cross_width`	NA	
`cg_negotiated_data_rate`	`pl_link_speed_cg`	Available from O-2018.06-2 release
`cg_negotiated_link_width`	`pl_link_width_cg`	Available from O-2018.06-2 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
cg_negotiated_data_rate_cross_negotiated_link_width	pl_link_width_and_speed_cg	Available from O-2018.06-3 release
cg_disable_scrambling	pl_ts_os_type_lane*_cg::symbol_5_ts_disable_scrambling_cp	Available from O-2018.06-2 release
cg_disable_scrambling_cross_rate	NA	
cg_ltssm_state_transitions	ltssm_state_transition_cg	Available from O-2018.06-2 release
cg_ltssm_state_transitions_cross_rate_cross_width	ltssm_state_transition_cross_speed_cg	aviable from O-2018.06-3 release Only LTSSM state transtion that varible with speed will be crossed with rate.
cg_tx_l0s_substate_transitions	ltssm_tx_l0s_cg::tx_l0s_state_cp	Available from O-2018.06-2 release
cg_cross_tx_l0s_substate_transitions_cross_rate_cross_width	ltssm_tx_l0s_cg::tx_l0s_state_transition_current_speed_cc	Available from O-2018.06-3 release
cg_rx_l0s_substate_transitions	ltssm_rx_l0s_cg	Available from O-2018.06-2 release
cg_cross_rx_l0s_substate_transitions_cross_rate_cross_width	ltssm_rx_l0s_cg::rx_l0s_state_transition_current_speed_cc	Available from O-2018.06-3 release
cg_ltssm_l0_exit_reasons	ltssm_state_transition_with_reason_cg::L0_STATE_TRANS_CP	Available from O-2018.06-2 release
cg_ltssm_l0_exit_reasons_cross_rate	ltssm_state_transition_with_reason_cg::L0_STATE_TRANS_CURRENT_SPEED_CC	Available from O-2018.06-3 release
cg_rx_advertised_n_fts	pl_ts_os_type_lane*_cg::symbol_3_n_fts_cp pl_ts_os_advertised_n_fts_cross_rate_lane*_cg	Available from O-2018.06-3 release
cg_tx_advertised_n_fts	pl_ts_os_type_lane*_cg::symbol_3_n_fts_cp pl_ts_os_advertised_n_fts_cross_rate_lane*_cg	Available from O-2018.06-3 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
cg_skp_tx_between_fts	pl_skp_tx_between_fts_cg	Available from O-2018.06-3 release
cg_skp_rx_between_fts	pl_skp_rx_between_fts_cg	Available from O-2018.06-3 release
cg_num_eie_before_fts	pl_num_eie_symbol_before_fts_cg	Available from O-2018.06-3 release
cg_num_eieos_after_fts	pl_num_eieos_after_fts_cg	Available from O-2018.06-3 release
cg_tx_skp_os_interval_in_symbol_times	pl_tx_skp_interval_cg	Available from O-2018.06-3 release
cg_tx_skp_os_interval_in_blocks	pl_tx_skp_interval_cg	Available from O-2018.06-3 release
cg_tx_lane_skew	pl_tx_lane_skew_cg	Available from O-2018.06-3 release
cg_num_txdetectrx_in_detect_active	LTSSM DETECT_ACTIVE_STATE_TRANS_CP::TO_DETECT_QUIET_LTSSM_TRANSITION_RCVR_DET_FAILED bin	Available from O-2018.06-2 release
cg_num_tx_skp_symbols_in_sos_8b10b	pl_tx_skp_os_lane*_cg::skp_8b_10b_length_cp	Available from O-2018.06-2 release
cg_num_tx_skp_symbols_in_sos_128b130b	pl_tx_skp_os_lane*_cg::skp_128b_130b_length_cp	Available from O-2018.06-2 release
cg_num_tx_symbols_in_last_eios_8b10b	pl_tx_eios_lane*_cg::eios_tx_8b10b_len_cp	Available from O-2018.06-2 release
cg_num_tx_symbols_in_last_eios_128b130b	pl_tx_eios_lane*_cg::eios_tx_128b130b_len_cp	Available from O-2018.06-2 release
cg_tx_ui_skew	pl_tx_lane_ui_skew_cg	Available from O-2018.06-3 release
cg_eq_phase2_preset_request_accepted	pl_equalization_phase2_requests_request_lane*_cg	Available from O-2018.06-3 release
cg_eq_phase3_preset_request_accepted	pl_equalization_phase3_requests_request_lane*_cg	Available from O-2018.06-3 release
cg_eq_phase2_coefficient_request_accepted	pl_equalization_phase2_requests_request_lane*_cg	Available from O-2018.06-3 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
cg_eq_phase3_coefficient_request_accepted	pl_equalization_phase3_requests_request_lane*_cg	Available from O-2018.06-3 release
cg_tx_kchar	pl_8B10B_k_code_lane*_cg pl_8B10B_k_code_without_stp_and_sdp_lane*_cg	Available from O-2018.06-2 release
cg_rx_kchar		Available from O-2018.06-2 release
cg_tx_token	pl_128b130b_token_cg	Available from O-2018.06-2 release
cg_rx_token		Available from O-2018.06-2 release
cg_tx_eqts_dc_balance	pl_ts_os_type_lane*_cg::symbol_14_to_symbol_15_dc_balance_cp	Available from O-2018.06-2 release
cg_rx_eqts_dc_balance		Available from O-2018.06-2 release
cg_link_error_type	Partially covered by pipe_status_if_rx_status_lane*_cg	Available from O-2018.06-2 release
cg_link_error_type_cross_rate		Available from O-2018.06-2 release
Not Available	pl_tx_lane_sub_ui_skew_cg	Available from O-2018.06-3 release
Not Available	tl_upstream_traveling_message_cg	Available from O-2018.06-2 release
Not Available	tl_downstream_traveling_message_cg	Available from O-2018.06-2 release
Not Available	dl_dlcmsm_cg	Available from O-2018.06-2 release
Not Available	ltssm_state_rc_cg / ltssm_state_ep_cg	Available from O-2018.06-2 release
Not Available	ltssm_state_transition_with_reason_cg ltssm_state_transition_with_reason_rc_cg / ltssm_state_transition_with_reason_ep_cg	Available from O-2018.06-2 release
Not Available	ltssm_tx_low_power_state_l0s_cg	Available from O-2018.06-2 release
Not Available	ltssm_rx_low_power_state_l0s_cg	Available from O-2018.06-2 release
Not Available	ltssm_low_power_state_l1_cg	Available from O-2018.06-2 release
Not Available	ltssm_low_power_state_l2_rc_cg / ltssm_low_power_state_l2_ep_cg	Available from O-2018.06-2 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
Not Available	ltssm_initial_bring_up_2_5_gts_cg ltssm_initial_bring_up_5_gts_cg ltssm_initial_bring_up_8_gts_rc_cg / ltssm_initial_bring_up_8_gts_ep_cg ltssm_initial_bring_up_16_gts_rc_cg / ltssm_initial_bring_up_16_gts_ep_cg	Available from O-2018.06-2 release
Not Available	ltssm_recovery_rc_cg / ltssm_recovery_ep_cg	Available from O-2018.06-2 release
Not Available	ltssm_polling_compliance_cg	Available from O-2018.06-2 release
Not Available	ltssm_disabled_cg	Available from O-2018.06-2 release
Not Available	ltssm_hot_reset_cg	Available from O-2018.06-2 release
Not Available	ltssm_loopback_cg	Available from O-2018.06-2 release
Not Available	ltssm_idle_to_rlock_cg	Available from O-2018.06-2 release
Not Available	ltssm_speed_negotiation_rc_cg/ltssm_speed_negotiation_ep_cg	Available from O-2018.06-2 release
Not Available	pl_speed_change_max_5_0gts_cg pl_speed_change_max_8_0gts_cg pl_speed_change_max_16_0gts_cg	Available from O-2018.06-2 release
Not Available	pl_link_width_change_x2_cg pl_link_width_change_x4_cg pl_link_width_change_x8_cg pl_link_width_change_x12_cg pl_link_width_change_x16_cg pl_link_width_change_x32_cg	Available from O-2018.06-2 release
Not Available	pl_os_type_lane*_cg - pl_os_type_lane32_cg	Available from O-2018.06-2 release
Not Available	pl_ts_os_type_lane32_cg - pl_ts_os_type_lane32_cg	Available from O-2018.06-2 release
Not Available	pl_8B10B_data_symbol_with_nrd_lane*_cg - pl_8B10B_data_symbol_with_nrd_lane32_cg	Available from O-2018.06-2 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
Not Available	pl_8B10B_data_symbol_with_prd_lane*_cg - pl_8B10B_data_symbol_with_prd_lane32_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_sris_enable_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_elasticity_buff_mode_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_tx_detect_rx_loopback_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_tx_elec_idle_lane*_cg pipe_cmd_if_tx_elec_idle_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_tx_compliance_lane*_cg pipe_cmd_if_tx_compliance_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_tx_compliance_and_tx_elec_idle_lane*_cg pipe_cmd_if_tx_compliance_and_tx_elec_idle_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_rx_polarity_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_reset_n_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_power_down_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_phy_specific_power_down_lane*_4_0_cg pipe_cmd_if_phy_specific_power_down_lane*_4_3_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_control_signal_decode_lane*_cg pipe_cmd_if_control_signal_decode_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_rate_lane*_2_0_cg pipe_cmd_if_rate_lane*_4_0_cg pipe_cmd_if_rate_lane*_4_2_cg pipe_cmd_if_rate_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_width_lane*_cg	Available from O-2018.06-2 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
Not Available	pipe_cmd_if_pclk_rate_lane*_4_0_cg pipe_cmd_if_pclk_rate_lane*_4_3_cg pipe_cmd_if_pclk_rate_lane*_4_4_cg pipe_cmd_if_pclk_rate_lane*_5_1_cg pipe_cmd_if_pclk_rate_serdes_arch_lane*_5_1_cg	Available from O-2018.06-2 release Available from O-2018.06-2 release Available from O-2018.06-2 release Available from O-2018.12-3 release Available from O-2018.12-3 release
Not Available	pipe_cmd_if_rate_width_and_pclk_rate_lane*_4_0_cg pipe_cmd_if_rate_width_and_pclk_rate_lane*_4_3_cg pipe_cmd_if_rate_width_and_pclk_rate_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_local_tx_coefficients_valid_lane*_4_0_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_local_tx_preset_coefficients_lane*_4_0_cg pipe_cmd_if_local_tx_preset_coefficients_lane*_4_2_cg pipe_cmd_if_local_tx_preset_coefficients_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_tx_deemph_lane*_4_0_cg pipe_cmd_if_tx_deemph_lane*_4_2_cg pipe_cmd_if_tx_deemph_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_rx_preset_hint_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_local_fs_lane*_4_0_cg pipe_cmd_if_local_fs_lane*_4_2_cg pipe_cmd_if_local_fs_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_local_lf_lane*_4_0_cg pipe_cmd_if_local_lf_lane*_4_2_cg pipe_cmd_if_local_lf_lane*_4_4_cg	Available from O-2018.06-2 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
Not Available	pipe_cmd_if_local_preset_index_lane*_4_0_cg pipe_cmd_if_local_tx_preset_coefficients_lane*_4_2_cg pipe_cmd_if_local_preset_index_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_get_local_preset_coefficients_lane*_4_0_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_fs_lane*_4_0_cg pipe_cmd_if_fs_lane*_4_2_cg pipe_cmd_if_fs_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_lf_lane*_4_0_cg pipe_cmd_if_lf_lane*_4_2_cg pipe_cmd_if_lf_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_rx_eq_eval_lane*_4_0_cg pipe_cmd_if_rx_eq_eval_lane*_4_2_cg pipe_cmd_if_rx_eq_eval_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_rx_eq_in_progress_lane*_4_2_cg pipe_cmd_if_rx_eq_in_progress_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_link_eval_feedback_figure_of_merit_lane*_4_0_cg pipe_cmd_if_link_eval_feedback_figure_of_merit_lane*_4_2_cg pipe_cmd_if_link_eval_feedback_figure_of_merit_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_link_eval_feedback_direction_change_lane*_4_0_cg pipe_cmd_if_link_eval_feedback_direction_change_lane*_4_2_cg pipe_cmd_if_link_eval_feedback_direction_change_lane*_4_4_cg	Available from O-2018.06-2 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
Not Available	pipe_cmd_if_invalid_request_lane*_4_0_cg pipe_cmd_if_invalid_request_lane*_4_2_cg pipe_cmd_if_invalid_request_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_tx_margin_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_tx_swing_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_rx_standby_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_rx_standby_status_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_status_if_phy_status_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_status_if_rx_status_lane*_cg pipe_status_if_rx_status_serdes_arch_lane*_cg	Available from O-2018.06-2 release Available from O-2018.12-3 release
Not Available	pipe_status_if_pclk_change_ok_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_status_if_pclk_change_ack_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_data_bus_width_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_rxelecidle_disable_lane*_cg pipe_cmd_if_rxelecidle_disable_lane*_5_1_cg	Available from O-2018.06-2 release Available from O-2018.09-3 release
Not Available	pipe_cmd_if_txcommonmode_disable_lane*_cg pipe_cmd_if_txcommonmode_disable_lane*_5_1_cg	Available from O-2018.06-2 release Available from O-2018.09-3 release
Not Available	pipe_cmd_if_ref_clk_required_n_lane*_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_if_serdes_arch_lane*_5_1_cg	Available from O-2018.12-3 release
Not Available	pipe_cmd_if_rx_width_lane*_5_1_cg	Available from O-2018.12-3 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
Not Available	pipe_mbi_m2p_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_mbi_p2m_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_status_if_async_power_change_ack_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_if_tx_block_align_control_lane*_4_0_cg pipe_cmd_if_tx_block_align_control_lane*_4_2_cg pipe_cmd_if_tx_block_align_control_lane*_4_4_cg	Available from O-2018.06-2 release Available from O-2018.06-2 release Available from O-2018.09-3 release
Not Available	pipe_data_if_tx_data_k_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_data_if_rx_data_k_lane*_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_mbi_mac_reg_rx_margin_status_0_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_mbi_mac_reg_rx_margin_status_1_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_mbi_mac_reg_rx_margin_status_2_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_mbi_mac_reg_elastic_buffer_status_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_mbi_mac_reg_elastic_buffer_location_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_mac_reg_rx_link_eval_status_0_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_mac_reg_rx_link_eval_status_1_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_mac_reg_rx_status_4_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_mac_reg_rx_status_5_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_mac_reg_tx_status_0_1_2_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_mac_reg_tx_status_3_lane*_5_1_cg	Available from O-2018.09-3 release

Table C-10 Mapping Legacy Covergroups to Corresponding New Covergroups

Legacy Covergroup	Mapped New Unified Covergroup	Comments Mapped New Unified Covergroup
Not Available	pipe_cmd_mbi_mac_reg_tx_status_4_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_mac_reg_tx_status_5_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_mac_reg_tx_status_6_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_phy_reg_rx_margin_control_0_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_mbi_phy_reg_rx_margin_control_1_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_mbi_phy_reg_elastic_buffer_control_lane*_4_4_cg	Available from O-2018.06-2 release
Not Available	pipe_cmd_mbi_phy_reg_rx_control_0_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_phy_reg_rx_control_3_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_phy_reg_elastic_buffer_location_update_frequency_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_phy_reg_rx_control_4_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_phy_reg_tx_control_2_3_4_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_phy_reg_tx_control_5_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_phy_reg_tx_control_6_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_phy_reg_tx_control_7_lane*_5_1_cg	Available from O-2018.09-3 release
Not Available	pipe_cmd_mbi_phy_reg_tx_control_8_lane*_5_1_cg	Available from O-2018.09-3 release

D Partition Compile and Precompiled IP

In design verification, every compilation and recompilation of the design and testbench contributes to the overall project schedule. A typical System-on-Chip (SoC) design may have one or more VIPs where changes are performed in the design or the testbench outside of VIPs. During the development cycle and the debug cycle, the complete design along with the VIP is recompiled, which leads to increased compilation time.

Verification Compiler offers the integration of VIPs with Partition Compile (PC) and Precompiled IP (PIP) flows. This integration offers a scalable compilation strategy that minimizes the VIP recompilations, and thus improves the compilation performance. This further reduces the overall time to market of a product during the development cycle and improves the productivity during the debug cycle.

The PC and PIP features in Verification Compiler provide the following solutions to optimize the compilation performance:

- The partition compile flow creates partitions (of module, testbench or package) for the design and recompiles only the changed or the modified partitions during the incremental compile.
- The PIP flow allows you to compile a self-contained functional unit separately in a design and a testbench. A shared object file and a debug database are generated for a self-contained functional unit. All of the generated shared object files and debug databases are integrated in the integration step to generate a simv executable. Only the required PIPs are recompiled with incremental changes in design or testbench.

For more information on the partition compile and Precompiled IP flows, see the VCS/VCS MX LCA Features Guide.

D.1 Implementing Partition Compile in Testbench

To effectively utilize the partition compile technology, PCIe VIP internally creates multiple sub-packages under existing top package, `svt_pcie_uvm_pkg`. The created multiple sub packages are compiled in parallel to improve the overall VIP compilation time. The compile time improvement depends on the number of cores selected for parallel execution. PCIe VIP sub-packages are mainly based on PCIe protocol layers like Transport, Data Link and Physical Layers along with separate package for coverage at each layer.

The partition compile feature is disabled by default. To enable the partition compile mode, you must perform the following functions:

- Include the file, `import_pcie_svt_uvm_pkgs.svi` to import all individual sub-packages.
- Set the compile macro, `SVT_PCIE_OPTIMIZED_COMPILE` in the compilation command.

D.1.1 High Level Architecture

The PCIe VIP creates multiple sub-packages for parallel compilation under existing PCIe UVM Package, `svt_pcie_uvm_pkg`. Following is the list of multiple sub-packages required to enable parallel compilation:

- `svt_pcie_common_uvm_pkg`
- `svt_pcie_tl_uvm_pkg` (TL Layer Tx/Rx)
- `svt_pcie_dl_uvm_pkg` (DL Layer TX/Rx)
- `svt_pcie_pl_pkg_common_uvm_pkg` (PL Layer Common)
- `svt_pcie_pl_uvm_pkg` (PL Layer Tx/Rx)
- `svt_pcie_tl_coverage_uvm_pkg` (TL Coverage)
- `svt_pcie_dl_coverage_uvm_pkg` (DL Coverage)
- `svt_pcie_pl_coverage_uvm_pkg` (PL Coverage)
- `svt_pcie_sequence_sequencer_collection_uvm_pkg` (Sequence Collection)

D.1.2 Guidelines for Partition Compile Usages

PCIe VIP support for partition compile feature is mostly backward compatible with the default VIP flow. Following are the scenarios where changes may be required to support partition compile flow:

1. **Avoid explicit file import from package.**

   A conflict can occur in scenarios, where class data type is referenced along with the existing PCIe package name using scope resolution. For example,

   ```
   import svt_pcie_uvm_pkg::svt_pcie_configuration;
   ```

   This existing class may have been moved to a new sub-package to support multiple partitions. Therefore, it is required to either change the package of explicit import of class

   ```
   `ifdef SVT_PCIE_OPTIMIZED_COMPILE
   import svt_pcie_common_uvm_pkg::svt_pcie_configuration;
   `elseif
   import svt_pcie_uvm_pkg::svt_pcie_configuration
   `endif
   ```

   Or, export the class file in the top package itself

   ```
   export svt_pcie_common_uvm_pkg::svt_pcie_configuration;
   ```

2. **Avoid accessing data class members using package name.**

   Example:

   ```
   svt_pcie_uvm_pkg::svt_pcie_target_app_configuration::UNINIT_MEM_READ_RESP_BAAD;
   ```

   This can cause an error because files might have been moved to a new package. If such access have been used at multiple places, then export the file in PCIe VIP top package, `svt_pcie_uvm_pkg`.

   Example:

   ```
   export svt_pcie_common_uvm_pkg::svt_pcie_target_app_configuration;
   ```

D.2 Use Model

You can use the three new simulation targets in the Makefiles of the VIP UVM examples to run the examples in the partition compile or the precompiled IP flow. In addition, Makefiles allow you to run the examples in back-to-back VIP configurations. The VIP UVM examples are located in the following directory:

$VC_HOME/examples/vl/vip/svt/vip_title/sverilog

For example,

/project/vc_install/examples/vl/vip/svt/pcie_svt/sverilog

Each VIP UVM example includes a configuration file called as the pc.optcfg file. This configuration file contains predefined partitions or precompiled IPs for the SystemVerilog packages that are used by VIP. The predefined partitions are created using the following heuristics:

- Separate partitions are created for packages that are common to multiple VIPs.
- The VIP level partitions are defined in a way that all of the partitions are compiled in the similar duration of time. This enables the optimal use of parallel compile with the `-fastpartcomp` option.

You can modify the pc.optcfg configuration file to include additional testbench or DUT level partitions.

D.2.1 Parallel Partition Compile

You must perform these steps to enable the parallel partition compilation flow with SVT PCIe VIP:

1. Provide the VCS compile argument:
   ```
   +define+SVT_PCIE_OPTIMIZED_COMPILE
   ```
2. Enable the partition compile flow using VCS partition compile options:
   ```
   -partcomp -fastpartcomp=j<N> +optconfigfile+pc.optcfg
   ```
3. Import all the packages or sub-packages at user testbench Top file. It is recommended to include methodology specific import file, say for example,
   ```
   `include "import_pcie_svt_uvm_pkgs.svi"
   ```

Note **Use auto partition if you have less number of cores.** When the number of available cores is less than 4, use the auto partitioning—that is, do not provide the `pc.optcfg` file with new packages because the partition compile on less cores with many partition degrades the compilation performance.

D.3 The "vcspcvlog" Simulator Target in Makefiles

The vcspcvlog simulator target in the Makefiles of the VIP UVM examples enables compilation of the examples in the two-step partition compile flow. The following partition compile options are used:

```
-partcomp +optconfigfile+pc.optcfg -fastpartcomp=j4 -lca
+define+SVT_PCIE_OPTIMIZED_COMPILE
```

One partition is created for each line specified in the pc.optcfg configuration file. The -fastpartcomp=j4 option enables parallel compilation of partitions on different cores of a multi-core machine. You can incorporate the partition compile options listed above into your existing vcs command line.

In the partition compile flow, changes in the testbench, VIP, or DUT source code trigger recompilation in only the required partitions. You must ensure that the Verification Compiler compilation database is not deleted between successive recompilations.

D.4 The “vcsmxpcvlog” Simulator Target in Makefiles

The vcsmxpcvlog simulator target in the Makefiles of the VIP UVM examples enables compilation of the examples in the three-step partition compile flow. The following partition compile options are used:

```
-partcomp +optconfigfile+pc.optcfg -fastpartcomp=j4 -lca
+define+SVT_PCIE_OPTIMIZED_COMPILE
```

There is no change in the vlogan commands. One partition is created for each line specified in the pc.optcfg configuration file. The -fastpartcomp=j4 option enables parallel compilation of partitions on different cores of a multi-core machine. You can incorporate the partition compile options listed above into your existing vcs command line.

In the partition compile flow, changes in the testbench, VIP, or DUT source code trigger recompilation only in the required partitions. You must ensure that the Verification Compiler compilation database is not deleted between successive recompilations.

D.5 The “vcsmxpipvlog” Simulator Target in Makefiles

The vcsmxpipvlog simulator target in the Makefiles of the VIP UVM examples enables compilation of the examples in the PIP flow. There is no change in the vlogan commands. One PIP compilation command with the -genip option is created for each line specified in the pc.optcfg configuration file. The -integ option is used in the integration step to generate the simv executable.

In the PIP flow, changes in the testbench, VIP, or DUT source code trigger recompilation in only the required PIPs. You must ensure that the Verification Compiler compilation database is not deleted between successive recompilations.

D.6 Precompiled IP Implementation in Testbenches with Verification IPs

You can use the Makefiles in the VIP UVM examples as a template to set up the partition compile or PIP flow in your design and verification environment by performing the following steps:

- Modify the pc.optcfg configuration file to include the user-defined partitions. The recommendations are as follows:
  - Create four to eight overall partitions (DUT and VIP combined).
  - Some VIP packages may include separate packages for transmitter and receiver VIPs. If only a transmitter or a receiver VIP is required, then the unused package can be removed from the configuration file.
  - Continue to use separate partitions for common packages, such as uvm_pkg and svt_uvm_pkg, as defined in the VIP configuration file.
- Incorporate the partition compile or precompiled IP command line options documented in previous sections or issued by the Makefile targets into the vcs command lines.

For more information on partition compile and precompiled IP options, such as, -sharedlib and -pcmakeprof, see the VCS/VCS MX LCA Features Guide.

D.7 Example

The following are the steps to integrate VIPs into the partition compile and PIP flows:

1. Once you set the VC_HOME variable, the VC_VIP_HOME variable is automatically set to the following location:

   $VC_HOME/vl

2. Check the available VIP examples using the following command:

   ```
   $VC_VIP_HOME/bin/dw_vip_setup -i home
   ```

3. Install the example.

   For example, to install the PCIe UVM Unified Example, use the following command:

   ```
   $VC_VIP_HOME/bin/dw_vip_setup -e pcie_svt/tb_pcie_svt_uvm_unified_vip_sys -svtb

   cd examples/sverilog/pcie_svt/tb_pcie_svt_uvm_unified_vip_sys
   ```

4. Run the tests present in the tests directory in the example.

   For example, to run the ts.base_test.sv test in the VCS two-step flow with partition compile, use the following command:

   ```
   gmake base_test USE_SIMULATOR=vcspcvlog
   ```

   To run the ts.base_test.sv test in the VCS UUM flow with partition compile, use the following command:

   ```
   gmake base_test USE_SIMULATOR=vcsmxpcvlog
   ```

   To run the ts.base_test.sv test in the VCS UUM flow with precompiled IP, use the following command:

   ```
   gmake base_test USE_SIMULATOR=vcsmxpipvlog
   ```

5. To modify or change the partitions, you must change the pc.optcfg file for the example.

E Protocol Checks

A number of automatic protocol checks are built into the PCIe VIP, to test for compliance with the PCIe specification. The HTML class reference documentation includes the protocol checks of the following groups:

Active component protocol check groups:

- `ACTIVE_PL_LANE_ENDEC`
- `ACTIVE_PL_LANE_OS`
- `ACTIVE_PL_LANE_PCS`
- `ACTIVE_PL_PIE8`
- `ACTIVE_PL_PIPE`
- `ACTIVE_PL_LANE_SERDES`
- `ACTIVE_PL`
- `ACTIVE_DL`
- `ACTIVE_TL`
- `ACTIVE_TARGET_APP`
- `ACTIVE_REQUESTER_APP`
- `ACTIVE_DRIVER_APP`

Passive component protocol check groups:

- `PASSIVE_PL_PIPE`
- `PASSIVE_PL`
- `PASSIVE_DL`
- `PASSIVE_TL`

For check description and PCIe specification version, see “Protocol Checks” tab in the HTML class reference documentation available at the following locations:

- PCIe SVT

  $DESIGNWARE_HOME/vip/svt/pcie_svt/latest/doc/pcie_svt_uvm_class_reference/html/

  protocolChecks.html

F 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.

F.1 Supported Interface Signals

Table F-1 lists the PIE-8 signals implemented to support equalization functionality.

Table F-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 F-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 F-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

F.2 Configuration Parameters

The following table Table F-3 lists configuration members for setting the PIE8 Interface in the class svt_pcie_pl_configuration. For additional information for the PHY layer configuration consult the HTML Reference documentation.

Table F-3 PIE-8 Parameters

Configuration Name	Description
pie8_mode_en	
	Enable PIE-8 mode (PHY Interface Extensions Supporting 8GT/s PCIe): Enables the PIE-8 mode state machines (either as MAC master1dor PHY Slave based on is_pie8_mode_master = SVT_PCIE_IS_PIE8_MODE_MASTER_DEFAULT;)
is_pie8_mode_master	
	Indicates whether the component is PIE8 master or PIE8 slave. When set to 1, acts as PIE8 master. When set to 0, acts as PIE8 slave.

Table F-3 PIE-8 Parameters (Continued)

Configuration Name	Description
pie8_phy_delay_to_tx_cmd_out_min	
	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 signal and proceeding toward completion.
pie8_phy_delay_to_tx_cmd_out_max	
	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 signal and proceeding toward completion.
pie8_enable_equalization_checks	
	When set to a 1, it enables the PIE-8 checks in the PIE8 PHY state machine. The checks performed are enabled individually as defined by pie8_enable_equalization_checks = SVT_PCIE_PIE8_ENABLE_EQUALIZATION_CHECKS_DEFAULT;.
pie8_equalization_check_vector	
	Enable PIE-8 checks when pie8_enable_equalization_checks is set to 1.
pie8_max_mac_wait_delay_to_phy_dataen_timeout	
	The maximum time (in ns) that a lane\u2019s Pie8MacStateMachine will wait in its TX_WAIT_RX_PHY_RESP state for the PHYDataEn signal to be received.

F.3 Status Class PIE8 Members

The following tableTable F-4 lists all the status members for the PIE8 interface in the class svt_pcie_pl_status.

Table F-4 PIE8 Members in Class svt_pcie_pl_status

PIE8 Member	Description
last_pie8_mac_state	Last state of the PIE-8 MAC master state machine. See list below for states.
last_pie8_phy_state	Last state of the PIE-8 PHY slave state machine. See list below for states
pie8_mac_state	Current state of the PIE-8 MAC master state machine. See list below for states.
pie8_phy_state	Current state of the PIE-8 PHY slave state machine. See list below for states.

These are the possible PIE8 PHY states:

- PHY_RX_IDLE
- PHY_RX_DATA

- PHY_RX_WAIT_TX_PHY_RESP
- PHY_TX_CMD_OUT
- PHY_TX_DATA
- PHY_WAIT_EVAL_RESP
- PHY_WAIT_MAC_DATA_EN_DROP

These are the possible PIE8 MAC states

- MAC_TX_IDLE
- MAC_TX_CMD_OUT
- MAC_TX_DATA(`SVT_PCIE_STATE_PIE8_MAC_TX_DATA)
- MAC_TX_WAIT_RX_PHY_RESP
- MAC_RX_DATA
- MAC_WAIT_PHY_DATA_EN_DROP

F.4 PHY PIE-8 ASCII Signals

The following table lists the ASCII signals on the PIE-8 PHY.

Table F-5 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

F.5 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 F-6 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.

Table F-6 MAC Internal PIE-8 "per Lane" Signals

Signal Name	Description
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 F-7 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
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.

F.6 PIE-8 Protocol Check "MSGCODEs"

The following table list the "MSGCODEs" associated with PIE-8 protocol and data checking.

Table F-8 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	

Table F-8 PIE-8 MSGCODES (Continued)

Signal Name	Description
	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.
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	

Table F-8 PIE-8 MSGCODES (Continued)

Signal Name	Description
	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.

G PCIe Compile-Time Parameters

Parameters that must be set before compilation are listed in the following sections:

G.1 Model Parameters

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

Table G-1 Parameters set inthe model

Parameter Name	Type	Range	Default Value	Description
DEVICE_IS_ROOT	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	Integer	10, 16, 32, 64, 128	10	Number of bits on the PMA interface

Table G-1 Parameters set inthe model (Continued)

PCIE_SPEC_VER	Real	1.1, 2.0, 2.1, 3.0	PCIE_SPEC_VER_3_0	See Include/pciesvc_parms.vp: PCIE_SPEC_VER_* Note: Please set this here, not in the individual layers.
HIERARCHY_ NUMBER	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.
DISPLAY_NAME	String		"svt_pcie_device_agent_xx_yy_hdl_model_zz." (Specific to each model).	String prefixed to messages to display in the output log.

G.2 Driver Application Parameters

Driver parameters are listed in Table G-2.

Table G-2 Driver parameters

Parameter Name	Type	Range	Default	Description
MAX_NUM_TAGS				
	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.
CMB_TABLE_SIZE				
	Integer	16-256	256	Size of the command management block, which is used to track pending and outstanding transactions.
DISPLAY_NAME				
	String		"pciesvc_driver"	Default display name for the driver.

G.3 Requester Parameters

Requester parameters are listed in Table G-3.

Table G-3 Requester parameters

Parameter Name	Type	Range	Default	Description
MAX_NUM_TAGS				
	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				
	Integer	16-4096	10	Number of unique randomization segments – each of which is a min/max memory address range.
CMB_TABLE_SIZE				

Table G-3 Requester parameters (Continued)

	Integer	16-256	64 entries	Size of the command management block table, which is used to track pending and outstanding transactions.
DISCARD_COMPLETIONS				
	Integer	01	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.
DISPLAY_NAME				
	String		pciesvc_ requester	Default prefix in $msglog calls.

G.4 Completion Target Parameters

Completion target parameters are listed in Table G-4.

Table G-4 Completion target parameters

Parameter Name	Type	Range	Default	Description
MEM_WRITE_NOTIFICATION_FIFO_SIZE				
	Integer	16-4096	64 entries	Number of queued memory-write notifications that can be queued up.
CMB_TABLE_SIZE				
	Integer	16-256	64 entries	Size of the command management block table, which is used to track pending and outstanding transactions.
DISPLAY_NAME				
	String		“pciesvc_ target”	Default prefix in $msglog calls.
PERCENTAGE_CORRUPT_TLP_DIGEST				
	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.

G.5 Memory Target Parameters

Memory target parameters are listed in Table G-5.

Table G-5 Memory target parameters

Parameter Name	Type	Range	Default	Description
PAGE_SIZE_IN_DWORDS				
	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				
	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				
	Integer	1024-16k	4096	Number of the 64 or 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, Small or Dword uses a single page entry.
NUM_32_BIT_PAGES				
	Integer	1024-16k	4096	
NUM_ATTR_TBL_ENTRIES				
	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				
	String		"memory_target0."	Default prefix in $msglog calls.

G.6 Transaction Layer Parameters

Transaction Layer parameters are listed in Table G-7.

Table G-6 Transaction Layer parameters

Parameter Name	Type	Range	Default Value	Description
DEFAULT_ROUTE_AT_APPL_ID				
	Integer	0 - large value	0	Default Application ID to route Address Translation requests to.
NUM_APPL_ID				
	Integer	8-128	8	Max number of unique Application IDs. IDs assigned to applications must be less than this value.

Table G-6 Transaction Layer parameters (Continued)

RID_APPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique RID to Appl_id map entries.
RID_MSGCODE_APPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique {RID,msgcode} to Appl_id map entries.
MEM_ADDR_ADDPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique Mem Address to Appl_id map entries.
IO_ADDR_ADDPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique I/O Address to Appl_id map entries.
AT_ADDR_ADDPLID_TABLE_SIZE				
	Integer	4-4096	64	Number of unique AT Address to Appl_id map entries.
IS_TX_DOWNSTREAM				
	Integer	0-1	0	Stack direction. Used for TLP header checking/routing.
IS_SWITCH				
	Integer	0-1	0	Is this stack part of a switch? Used for TLP header checking/routing.

G.7 Data Link Layer Parameters

Data Link Layer parameters are listed in Table G-7.

Table G-7 Data Link Layer parameters

Parameter Name	Type	Range	Default	Description
MAX_NUM_RETRY_BUFFER_DWORDS				
	Integer	16-2^16	4096	Maximum number of dwords the retry buffer can hold before it backpressures the Transaction Layer.

G.8 Physical Layer Parameters

G.8.1 General Parameters

General Physical Layer parameters are listed in Table G-8

Table G-8 Physical Layer general parameters

Parameter Name	Type	Range	Default	Description
DISPLAY_NAME				
	String		pciesvc_pl0	Default display name for the Physical Layer.

G.8.2 Physical Layer LTSSM-specific Parameters

Physical Layer LTSSM-specific parameters are listed in Table G-9.

Table G-9 LTSSM-specific parameters

Parameter Name	Type	Range	Default	Description
IS_TX_DOWNSTREAM				
	Integer	0-1	0	Indicates whether the transmitter is downstream or not. This affects link training behavior.
IS_PIPE_MASTER				
	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.

G.8.3 Equalization Parameters

Equalization parameters are listed in Table G-10.

Table G-10 Equalization parameters

Parameter Name	Type	Range	Default	Description
FULL_SWING				
			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				
			?'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				
			3	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				
			5	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				
			6	Default postcursor coefficient value that is loaded into all preset settings. Users should use the task SetEQPreset2CoeffTable to change preset to coefficient mappings.

G.9 Physical Coding Sublayer (PCS) Parameters

PCS parameters are listed in Table G-11.

Table G-11 PCS parameters

Parameter Name	Type	Range	Default	Description
ENABLE_RX_ELASTIC_BUFFER				
	Integer	0-1	1	Enables the receive elastic buffer. This is not necessary for most simulations.
NUM_PMA_INTERFACE_BITS				
	Integer	10, 16, 32, 64, 128	10	Number of bits on the PMA interface. NOTE: please set this in the model.
DISPLAY_NAME				
	String		pciesvc_pcs	Default display name for msglog statements.

G.10 Serializer/Deserializer (SERDES) Parameters

Parameters that affect the behavior of the SERDES are listed in Table G-12.

Table G-12 SERDES parameters

Parameter Name	Type	Range	Default	Description
COMMA_SYNC_COUNT				
	Integer	0 - large value	0	Number of aligned commas before SERDES lock declared. Not used in PCIE.
BIT_SYNC_COUNT				
	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				
	Real	Any value	0.0001	Tolerance in ns. Equivalent to 100 PPM SAS requirement.
COMMA_P				
	Integer	Any 10-bit value	10’b0011 111010	Definition of positive COMMA character.
COMMA_N				
	Integer	Any 10-bit value	10’b1100 000101	Definition of negative COMMA character.
DISPLAY_NAME				
	String		pciesvc_ serdes.	String prefixed to messages to display in the output log.

H Verilog Task/Parameter to SVT Class Mapping

This appendix provides mapping from SVC Verilog or SVT Verilog tasks and parameters to SVT UVM class members, for users who are migrating from the SVC or SVT Verilog PCIe VIP to the UVM PCIe.

This appendix contains the following sections:

- "Transaction Layer Verilog Tasks and Parameters to UVM Class Members Map" on page 537
- "Data Link Layer Verilog Tasks and Parameters to UVM Class Member Maps" on page 539

H.1 Transaction Layer Verilog Tasks and Parameters to UVM Class Members Map

This section contains the following tables:

- "Transaction Layer Verilog Task to UVM Class Member Map" on page 537
- "Transaction Layer Verilog Parameters to UVM Class Members Map" on page 538

H.1.1 Transaction Layer Verilog Task to UVM Class Member Map

Transaction Layer Verilog tasks are mapped to SVT class members in Table H-1, listed alphabetically by Verilog task.

Table H-1 Map of Transaction Layer Verilog tasks to UVM class members

Verilog Task	UVM Class Member
AddATAddrApplIdMapEntry()	svt_pcie_tl_service::service_type =
AddCfgBDFApplIdMapEntry()	svt_pcie_tl_service::service_type
AddIOAddrApplIdMapEntry()	svt_pcie_tl_service::service_type
AddMemAddrApplIdMapEntry()	svt_pcie_tl_service::service_type
AddRequesterIdApplIdMapEntry()	svt_pcie_tl_service::service_type
AddRIdMsgCodeApplIdMapEntry()	svt_pcie_tl_service::service_type

Table H-1 Map of Transaction Layer Verilog tasks to UVM class members (Continued)

Verilog Task	UVM Class Member
CheckFinalCredits()	svt_pcie_tl_service_check_final_credits_sequence
ClearStats()	svt_pcie_tl_service::service_type
DisplayATAddrApplidMap()	svt_pcie_tl_service::service_type
DisplayCfgBDFApplidMap()	svt_pcie_tl_service::service_type
DisplayIOAddrApplidMap()	svt_pcie_tl_service::service_type
DisplayMemAddrApplidMap()	svt_pcie_tl_service::service_type
DDisplayRidApplidMap()	svt_pcie_tl_service::service_type
DisplayRidMsgCodeApplidMap()	svt_pcie_tl_service::service_type
DisplayStats()	svt_pcie_tl_service::service_type
IsTransactionLayerIdle()	svt_pcie_tl_service::service_type
IsVcInitFinished()	svt_pcie_tl_status::vc_initialized
QueryCreditCounts()	svt_pcie_tl_status::credits_allocated::credit_limit::credits_consumed :: credits_received::init_credits_allocated::init_credit_limit
QueryRxCreditsAvailable()	svt_pcie_tl_status::rx_credits_available[48]
QueryTxCreditsAvailable()	svt_pcie_tl_status::tx_credits_available[48]
SetVcEnable()	svt_pcie_tl_service::service_type

H.1.2 Transaction Layer Verilog Parameters to UVM Class Members Map

Transaction Layer Verilog parameters are mapped to UVM class members in Table H-2, listed alphabetically by Verilog parameter.

Table H-2 Map of Transaction Layer class members to parameters

Verilog Parameter	UVM Class Member
AUTO_ENABLE_VC0_AT_STARTUP	svt_pcie_tl_configuration::auto_enable_vc0_at_startup
CREDIT_STARVATION_TIMEOUT_NS	svt_pcie_tl_configuration::credit_starvation_timeout_ns
DEFAULT_ROUTE_AT_APPL_ID	svt_pcie_tl_configuration::default_route_at_appl_id
DEFAULT_ROUTE_CFG_TYPE0_APPL_ID	svt_pcie_tl_configuration::default_route_cfg_type0_appl_id
DEFAULT_ROUTE_CFG_TYPE1_APPL_ID	svt_pcie_tl_configuration::default_route_cfg_type1_appl_id
DEFAULT_ROUTE_IO_APPL_ID	svt_pcie_tl_configuration::default_route_at_appl_id
DEFAULT_ROUTE_MEM_APPL_ID	svt_pcie_tl_configuration::default_route_mem_appl_id
DEFAULT_ROUTE_MSG_APPL_ID	svt_pcie_tl_configuration::default_route_msg_appl_id
ENABLE_ROUTE_AT_TO_FUNCTION	svt_pcie_tl_configuration::enable_route_at_to_function
ENABLE_ROUTE_CFG_TYPE0_TO_FUNCTION	svt_pcie_tl_configuration::enable_route_cfg_type0_to_function

Table H-2 Map of Transaction Layer class members to parameters (Continued)

Verilog Parameter	UVM Class Member
ENABLE_ROUTE_CFG_TYPE1_TO_FUNCTION	svt_pcie_tl_configuration::enable_route_cfg_type1_to_function
ENABLE_ROUTE_IO_TO_FUNCTION	svt_pcie_tl_configuration::enable_route_io_to_function
ENABLE_ROUTE_MEM_TO_FUNCTION	svt_pcie_tl_configuration::enable_route_mem_to_function
ENABLE_ROUTE_MSG_TO_FUNCTION	svt_pcie_tl_configuration::enable_route_msg_to_function
MAX_VC[0-7]_[P/NP/CPL]_UPDATEFC_DELAY	svt_pcie_tl_configuration::max_vc[0-7]_[p|np|cpl]_updatefc_delay
MIN_VC[0-7]_[P/NP/CPL]_UPDATEFC_DELAY	svt_pcie_tl_configuration::min_vc[0-7]_[p|np|cpl]_updatefc_delay
NUM_VC_[P/NPCPL]_NIT_[HDRDATA]_CREDITS	svt_pcie_tl_configuration::init_[cpl|np|p]_data_tx_credits
NUM_VC_[P/NPCPL]_NIT_[HDRDATA]_CREDITS	svt_pcie_tl_configuration::init_[cpl|np|p]_hdr_tx_credits
REMOTE_EXTENDED_TAG_FIELD_ENABLED	svt_pcie_tl_configuration::remote_extended_tag_field_enabled
REMOTE_MAX_PAYLOAD_SIZE	remote_max_payload_size::remote_max_payload_size
REMOTE_MAX_READ_REQUEST_SIZE	svt_pcie_tl_configuration::remote_max_read_request_size

H.2 Data Link Layer Verilog Tasks and Parameters to UVM Class Member Maps

- "Data Link Layer Verilog Task to UVM Class Member Map"
- "Data Link Layer Verilog Parameter to UVM Class Member Map"

H.2.1 Data Link Layer Verilog Task to UVM Class Member Map

Data Link Layer Verilog tasks are mapped to UVM class members in Table H-3, listed alphabetically by Verilog task.

Table H-3 Map of Data Link Layer Verilog tasks to UVM class members

Verilog Task	UVM Class Member
ClearStats	svt_pcie_dl_service_clr_stats_sequence
DisplayAttachedAckNakLatencyTolerances	svt_pcie_dl_configuration::min_ack_nak_latency
DisplayAttachedReplayTimeoutTolerances	svt_pcie_dl_service::DISPLAY_ATTACHED_REPLAY_TIMER_TOLERANCES
DisplayStats	svt_pcie_dl_service_disp_stats_sequence
InitiateASPMExit	svt_pcie_dl_configuration::min_ack_nak_latency
InitiateASPML0sEntry	svt_pcie_dl_configuration::min_ack_nak_latency
InitiateASPML1Entry	svt_pcie_dl_configuration::min_ack_nak_latency
InitiatePMExit	svt_pcie_dl_configuration::min_ack_nak_latency
InitiatePML1Entry	svt_pcie_dl_configuration::min_ack_nak_latency
InitiatePML23Entry	svt_pcie_dl_configuration::min_ack_nak_latency
IsDataLinkIdle	svt_pcie_dl_service_disp_stats_sequence

Table H-3 Map of Data Link Layer Verilog tasks to UVM class members (Continued)

Verilog Task	UVM Class Member
ReceivedDLLP	svt_pcie_dl::received_tlp_observed_port
ReceiveVendorDLLP	svt_pcie_dllp_vendor_defined_sequence, svt_pcie_dllp_vendor_defined_exception_sequence
SentDLLP	svt_pcie_dl::sent_dllp_observed_port
SentTLP	svt_pcie_dl::sent_tlp_observed_port
SetAttachedAckNakLatencyTolerance	svt_pcie_dl_configuration::attached_ack_nak_latency_tolerance_x1
SetAttachedReplayTimeout	svt_pcie_dl_configuration::attached_replay_timeout
SetAttachedReplayTimeoutTolerance	svt_pcie_dl_configuration:attached_replay_timeout_tolerance_xn where n is 1,2,4,8,12,16,32.
SetLinkEnable()	svt_pcie_dl_service_set_link_en_sequence::enable
SetMaxAckNakLatency	svt_pcie_dl_configuration::max_ack_nak_latency
SetMaxAttachedAckNakLatency	svt_pcie_dl_configuration::max_attached_nak_latency
SetMaxAttachedNakLatency	svt_pcie_dl_configuration::min_ack_nak_latency
SetMinAckNakLatency	svt_pcie_dl_configuration::min_ack_nak_latency
SetMinAttachedAckNakLatency	svt_pcie_dl_configuration::min_ack_nak_latency
SetReplayTimeout	svt_pcie_dl_configuration::replay_timeout
TransmitUserDLLP	svt_pcie_dllp_vendor_defined_sequence, svt_pcie_dllp_vendor_defined_exception_sequence

H.2.2 Data Link Layer Verilog Parameter to UVM Class Member Map

Data Link Layer Verilog parameters are mapped to UVM class members in Table H-4, listed alphabetically by Verilog parameter.

Table H-4 Map of Data Link Layer class members to tasks

Verilog Parameter	UVM Class Member
ASPM_TIMEOUT_CNT_LIMIT	svt_pcie_dl_configuration::aspm_timeout_cnt_limit
ATTACHED_INTERNAL_DELAY_2_5G	svt_pcie_dl_configuration::attached_internal_delay_2_5g
ATTACHED_INTERNAL_DELAY_5G	svt_pcie_dl_configuration::attached_internal_delay_5g
ENABLE_ASPM_L1_1_ENTRY	svt_pcie_dl_configuration
ENABLE_ASPM_L1_2_ENTRY	svt_pcie_dl_configuration
ENABLE_ASPM_L1_ENTRY	svt_pcie_dl_configuration::enable_aspm_l1_entry
ENABLE_EI_TX_TLP_ON_RETRY	svt_pcie_dl_configuration
ENABLE_PM_L1_1_ENTRY	svt_pcie_dl_configuration
ENABLE_PM_L1_2_ENTRY	svt_pcie_dl_configuration
ENABLE_TRANSACTION_LOG	svt_pcie_dl_configuration

Table H-4 Map of Data Link Layer class members to tasks (Continued)

Verilog Parameter	UVM Class Member
ENABLE_TX_TLP_REPORTING	svt_pcie_dl_configuration
INITFC_TIMEOUT_NS	svt_pcie_dl_configuration::min_ack_nak_latency
INITIAL_RECEIVE_SEQUENCE_VALUE	svt_pcie_dl_configuration::initial_receive_sequence_value
INITIAL_TRANSMIT_SEQUENCE_VALUE	svt_pcie_dl_configuration::initial_transmit_sequence_value
InitiateASPMExit	svt_pcie_dl_configuration::min_ack_nak_latency
InitiateASPML0sEntry	svt_pcie_dl_configuration::max_ack_nak_latency
InitiateASPML1Entry	svt_pcie_dl_configuration::min_ack_nak_latency
InitiatePMExit	svt_pcie_dl_configuration::min_ack_nak_latency
InitiatePML1Entry	svt_pcie_dl_configuration::min_ack_nak_latency
InitiatePML23Entry	svt_pcie_dl_configuration::min_ack_nak_latency
INTERNAL_DELAY_2_5G	svt_pcie_dl_configuration::internal_delay_2_5g
INTERNAL_DELAY_5G	svt_pcie_dl_configuration::internal_delay_5g
L0S_IDLE_TIMER_LIMIT_NS	svt_pcie_dl_configuration::l0s_idle_timer_limit_ns
LTR_L1_2_THRESHOLD_SCALE	svt_pcie_dl_configuration
LTR_L1_2_THRESHOLD_VALUE	svt_pcie_dl_configuration
MAX_INITFC_DELAY	svt_pcie_dl_configuration::min_ack_nak_latency
MAX_NAK_LATENCY	svt_pcie_dl_configuration::min_ack_nak_latency
MAX_NUM_REPLAYS	svt_pcie_dl_configuration::min_ack_nak_latency
MAX_PAYLOAD_SIZE	svt_pcie_dl_configuration::min_ack_nak_latency
MAX_TX_IPG	svt_pcie_dl_configuration::min_ack_nak_latency
MAX_TX_NULLIFIED_TLP_LEN	svt_pcie_dl_configuration::max_tx_nullified_tlp_len
MAX_UPDATEFC_DELAY	svt_pcie_dl_configuration::min_ack_nak_latency
MIN_INITFC_DELAY	svt_pcie_dl_configuration::min_ack_nak_latency
MIN_NAK_LATENCY	svt_pcie_dl_configuration::min_ack_nak_latency
MIN_TX_IPG	svt_pcie_dl_configuration::min_ack_nak_latency
MIN_TX_NULLIFIED_TLP_LEN	svt_pcie_dl_configuration::min_tx_nullified_tlp_len
MIN_UPDATEFC_DELAY	svt_pcie_dl_configuration::min_ack_nak_latency
PCIE_SPEC_VER	svt_pcie_dl_configuration::min_ack_nak_latency
PERCENTAGE_TX_TLP_INSTEAD_OF_INITFC2	svt_pcie_dl_configuration::tx_fc_init_completed_percentage
PM_TIMEOUT_CNT_LIMIT	svt_pcie_dl_configuration::pm_timeout_cnt_limit
RECEIVED_DLLP_INTERFACE_MODE	svt_pcie_dl_configuration::received_dllp_interface_mode
RECEIVED_TLP_INTERFACE_MODE	svt_pcie_dl_configuration::received_tlp_interface_mode
SENT_DLLP_INTERFACE_MODE	svt_pcie_dl_configuration::sent_tlp_interface_mode

Table H-4 Map of Data Link Layer class members to tasks (Continued)

Verilog Parameter	UVM Class Member
SENT_TLP_INTERFACE_MODE	svt_pcie_dl_configuration::sent_tlp_interface_mode
TX_NULLIFIED_TLP_HDR0_VALUE	svt_pcie_dl_configuration::tx_nullified_tlp_hdr0_value
TX_NULLIFIED_TLP_HDR1_VALUE	svt_pcie_dl_configuration::tx_nullified_tlp_hdr1_value
TX_NULLIFIED_TLP_HDR2_VALUE	svt_pcie_dl_configuration::tx_nullified_tlp_hdr2_value
TX_NULLIFIED_TLP_HDR3_VALUE	svt_pcie_dl_configuration::tx_nullified_tlp_hdr3_value
UPDATEFC_TIMEOUT_NS	svt_pcie_dl_configuration::min_ack_nak_latency
VC[0-7]_UPDATEFC_INTERVAL_NS	svt_pcie_dl_configuration::vc[0:7]_updatefc_interval_ns

I SolvNet PCIe VIP Articles

The following table lists and links to all the SolvNet articles published on the PCIe VIP. The articles are organized by the following categories:

- "Application Layer" on page 543
- "Transaction Layer" on page 544
- "Data Link Layer" on page 545
- "PHY Layer" on page 546
- "Methodology Testbench and Debug" on page 551
- "Miscellaneous" on page 555

I.1 Application Layer

I.1.1 Driver App

See the following articles:

- VC VIP: Configuring PCIe for Sending Packets Back-to-Back
- VC VIP: Generating MSG TLPs with the PCIe VIP
- PCIe SVT VIP: Memory Read Transactions Causing an Uninitialized Memory Error
- PCIE SVT: Setting the EP Bit in a Driver Application Transaction for MemRd TLPs
- VC VIP: Setting the PCIe VIP to Expect CRS Status on Received Completions
- PCIE SVT VIP : svt_pcie_driver_app_transaction : Config Write
- PCIe SVT svt_pcie_driver_app_transaction transaction_type
- PCIE SVT VIP: svt_pcie_driver_app_transaction, Config Read
- VC VIP: Generating Type0/Type1 Configuration Transactions in PCIe
- VC VIP: Generating Memory Write and Memory Read Transactions in PCIe
- VC VIP: Understanding PCIe EI_CODEs and Their Usage
- PCIe SVC Verilog VIP: Injecting Various Types of ERRORs at the Transactional Level of the Model
- VC VIP: Fields of svt_pcie_driver_app_transaction Class Used in Configuration Transactions in PCIe

- VC VIP: Usage Examples for svt_pcie_driver_app_cfg_request_sequence (PCIe)
- VC VIP: Configuring the PCIe VIP to Not Expect a Completion for a Read Request
- VC VIP: Setting the Correct COMPLETION_TIMEOUT_NS Value to Avoid APPL_DRIVER_COMMAND_TIMEOUT Error in PCIe

I.1.2 Target App

See the following articles:

- PCIE SVT: Create out of order completions (CPL/CPLDs) of TLP packets
- VC VIP: Generating Poisoned Completions in PCIe
- VC VIP: Sending Different Size Completion Packets
- VC VIP: Configuring PCIe to Send Completions on Second RCB
- VC VIP: Generating PCIe Error Poisoned Completions
- VC VIP: Creating Out of Order Completions with the UVM PCIe VIP
- VC VIP: Emulating a PCIe Completion Timeout Error Using the PCIe VIP
- VC VIP: Varying Completion Response Latencies in the PCIe VIP
- VC VIP: Setting Read Completion Boundaries in the PCIe VIP
- VC VIP: Updating/Setting a PCIe TLP Digest Field While Corrupting ECRC
- VC VIP: How to Control the Read Request Completion Latency in PCIe
- VC VIP: Corrupting Completions Using Target Application Callback in PCIe
- VC VIP: Guidelines for Using max_read_cpl_data_size_in_bytes in PCIe Target App Configuration
- VP VIP: Setting Completion Latency Values and Generating Out of Order Completions in PCIe
- VC VIP: Generating and Setting the Percentage of Error Poisoned Completions in PCIe

I.1.3 Requester App

See the following articles:

- The SetRequestID Verilog task of the ExpertIO PCIE SVC

I.2 Transaction Layer

See the following articles:

- VC VIP: Resolving Memory Read Error (MemRd accessed unitialized memory) in PCIe
- VC VIP: Setting the PCIe VIP FC Credit Starvation Timeout
- VC VIP: Retrieving Final Credit Information in PCIe
- VC VIP PCIe: Traffic Class and Virtual Channels
- VC VIP: Application Routing with an Upstream Port in the PCIe VIP
- VC VIP: Updating TLP Reserved Fields During a PCIe CFG Request

- VC VIP: Notes about PCIe VIP Handling of Duplicate TLPs
- VC VIP: Using various EI_CODEs at Transaction Layer level in PCIe
- VC VIP: Creating Spurious Completion TLPs in the PCIe Model
- VC VIP: Displaying TLP Payload in a Transaction Log in PCIe
- VC VIP: Avoid Automatic Corruption of a TLP Packet Generated from PCIe VIP
- VC VIP: Transmitting Nullified TLP Packet From PCIe
- VC VIP: Setting Transaction Ordering Rule in PCIe
- VC VIP: Resolving UVM Error (Received duplicate TLP) in PCIe
- VC VIP: Resolving UVM ERROR (HandleCfgReq: Wrong Bus# 0x00 for Function# 0 - Bus#) in PCIe
- VC VIP: Stop Sending ACK or NAK in Response to TLPs in PCIe
- VC VIP: Resolving Memory Allocation Issues in the PCIe SVT VIP
- VC VIP: Controlling the Number of Completions Sent in Response to a Read Request in the PCIe VIP
- VC VIP: Malformed TLP Error Related to Extended Tag in PCIe
- VC VIP: Retaining ECRC After TLP Corruption in PCIe
- VC VIP: Configuration Error When 10-Bit Tag Is Not Enabled in PCIe
- VC VIP: User-Defined Tag Field of a Memory Read/Write Transaction in PCIe
- VC VIP: Enable PASID for MRd/MWr Packets Driven by PCIe
- VC VIP: Reading From an Unallocated Memory in BuildMemCpl in PCIe
- VC VIP: Probably Cause for the PCIe VIP Not to Release Posted Credits
- VC VIP: Controlling the Vendor ID Field of a Msg TLP in PCIe
- VC VIP: Setting the VIP to Expect Cfg Read to Have a Competion as Config Retry Status
- VC VIP: Preventing the Gerneration of ECRCs in PCIe.
- VC VIP: Understanding How the PCIe Model Manages the Transmission Order of TLPs
- VC VIP: Initializing PCIe Memory Space with Random Data
- VC VIP: Discarding AtomicOPs Transactions When EP DUT Does Not Support AtomicOPs in PCIe
- VC VIP: DUT Response to ECRC Check Failure in PCIe
- VC VIP: Vendor Message With Multiple Data Payload in PCIe
- VC VIP: Maximum Allowable Payload Size in PCIe
- VP VIP: Configuring PCIe VIP to Send Read Completions of Required Size

I.3 Data Link Layer

See the following articles:

- VC VIP: PCIe VIP Retry Support for CFG TLPs
- VC VIP: Using the PCIe VIP to Block INITFCs and Then Sending Out User-Defined and/or Vendor Specific DDLPS
- VC VIP: Programming the PCIe VIP to Send a Single ACK for Multiple TLP Packets

I.4 PHY Layer

Title: PCIe SVC: Negotiating on Lower Link Width than VIP Actually Supports

https://solvnet.synopsys.com/retrieve/2047172.html

Title: VC VIP: Testing the Taking Down of a Link Using the PCIe VIP

https://solvnet.synopsys.com/retrieve/1894725.html

Title: VC VIP: PCIe VIP Issues RxStandby Error Messages in Recovery.Speed

https://solvnet.synopsys.com/retrieve/2302038.html

Title: VC VIP: PCIe VIP Not Sending User TS1s

https://solvnet.synopsys.com/retrieve/1894555.html

Title: VC VIP: Some Issues to Consider When Connecting the PCIe VIP SERDES to the Synopsys PCIe IIP.

https://solvnet.synopsys.com/retrieve/2060917.html

Title: VC VIP PCIe: Lane Reversal Testing

https://solvnet.synopsys.com/retrieve/1831170.html

Title: VC VIP: Corrupting the 'COM' Symbol in a Skip Ordered-Set Using the PCIe SVT VIP

https://solvnet.synopsys.com/retrieve/2024177.html

Title: Handling DUT port reset with the PCIe SVT VIP

https://solvnet.synopsys.com/retrieve/1598388.html

Title: VC VIP PCIe: Polarity Inversion

https://solvnet.synopsys.com/retrieve/1824528.html

Title: VC VIP: Resolving Why the PCIe Model Does Not Enter L1 Through ASPM

https://solvnet.synopsys.com/retrieve/2061376.html

Title: Why is the Discovery pcie_svt VIP stuck in the CONFIGURATION_LINK_WIDTH_START state when link number is not zero?

https://solvnet.synopsys.com/retrieve/1483737.html

Title: VC VIP: Setting Valid FS, LF and Equalization/Coefficient Values in the UVM PCIe VIP

https://solvnet.synopsys.com/retrieve/1894110.html

Title: VC VIP: PCIe UVM Attributes to Change to Support a 500MHZ PCLK Frequency in Gen3

https://solvnet.synopsys.com/retrieve/2359311.html

Title: VC VIP: Active and Passive Components and Scrambling in PCIe

https://solvnet.synopsys.com/retrieve/2432153.html

Title: VC VIP: Enabling the PCIe VIP PHY

https://solvnet.synopsys.com/retrieve/2057812.html

Title: VC VIP: Critical Points about PCIe Link Width Settings

https://solvnet.synopsys.com/retrieve/2057833.html

Title: VC VIP: Setting the Target & Expected Link Speeds in the PCIe VIP

https://solvnet.synopsys.com/retrieve/1904896.html

Title: VC VIP: Bringing Up a Link When lane0 is Disabled in the PCIe VIP

https://solvnet.synopsys.com/retrieve/2058463.html

Title: VC VIP: Example Showing Multiple Iterations on PCIe L1 Sub-States

https://solvnet.synopsys.com/retrieve/2119994.html

Title: VC VIP: BIT_FLIP Error Injection in PCIe

https://solvnet.synopsys.com/retrieve/2464559.html

Title: VC VIP: Checking the Presence of Retimer in PCIe

https://solvnet.synopsys.com/retrieve/2787233.html

Title: VC VIP: Initiating Redo Equalization for RC DUT in PCIe

https://solvnet.synopsys.com/retrieve/2851992.html

Title: VC VIP: Enabling eq_eval During RECOVERY_EQUALIZATION in the PCIe SVT Model

https://solvnet.synopsys.com/retrieve/2343288.html

Title: VC VIP: Enabling PCIe VIP to Detect the Presence of a Retimer

https://solvnet.synopsys.com/retrieve/2787194.html

Title: VC VIP: Passive Monitor Issues Error When DUT Is in Electrical Idle Longer Than 1000 ns in PCIe

https://solvnet.synopsys.com/retrieve/2787173.html

Title: VC VIP: Reconfiguring the Link Width of the PCIe Model During Run Time

https://solvnet.synopsys.com/retrieve/2263409.html

Title: VC VIP: Achieving Non-Zero Error Count in Response to Timing Margining Command in PCIe

https://solvnet.synopsys.com/retrieve/2787050.html

Title: VC VIP: Enabling EIEOS Pattern for 16GT in PCIe
https://solvnet.synopsys.com/retrieve/2528058.html

Title: VC VIP: Lane-to-Lane Skew in Transmit/Receive Path in PCIe
https://solvnet.synopsys.com/retrieve/2662399.html

Title: VC VIP: Avoiding "Malformed OS detected " Errors Using the PCIe Passive Monitor
https://solvnet.synopsys.com/retrieve/2334608.html

Title: VC VIP: Controlling the Number of Garbage Symbols to be Sent After an EIOS Is Transmitted in PCIe
https://solvnet.synopsys.com/retrieve/2528011.html

Title: VC VIP: Updating Symbols During User Defined TS1 OS Using the PCIe VIP
https://solvnet.synopsys.com/retrieve/2343320.html

Title: VC VIP: Software Initiated Gen4 Equalization in PCIe
https://solvnet.synopsys.com/retrieve/2767508.html

Title: VC VIP: Possible Scenario for FIFO Overflow Issues in PCIe
https://solvnet.synopsys.com/retrieve/3010813.html

Title: VC VIP: Disabling Scrambling in the PCIE SVT VIP
https://solvnet.synopsys.com/retrieve/2349545.html

Title: VC VIP: Entering/Exiting Low Power States in PCIe
https://solvnet.synopsys.com/retrieve/2464089.html

Title: VC VIP: Configuring SKP Symbols in a SKP Order Set in the PCIe VIP Model
https://solvnet.synopsys.com/retrieve/2298918.html

Title: VC VIP: Controlling the Number of EIOS Before L1 State in PCIe Gen3
https://solvnet.synopsys.com/retrieve/2788451.html

Title: PCIe UVM VMT: An Example Showing How to Bring the Link to a DISABLE State and Then Back to DETECT
https://solvnet.synopsys.com/retrieve/2046500.html

Title: PCIe VMT VIP: Example for Generating Hot Reset
https://solvnet.synopsys.com/retrieve/2060580.html

Title: PCIE VMT: Performing Speed Changes From Gen1-to-Gen2-to-Gen1 from a RC PCIe VIP
https://solvnet.synopsys.com/retrieve/2060956.html

Title: VC VIP: Available Configuration Settings for Changing Link Width in PCIe
https://solvnet.synopsys.com/retrieve/3089015.html

Title: VC VIP: Bringing up the Link When Lane0 is Disabled in PCIe - Lane Reversal Mode Usage
https://solvnet.synopsys.com/retrieve/3093529.html

Title: VC VIP: Error "TX L0s status after 20000 symbols" in PCIe
https://solvnet.synopsys.com/retrieve/3044900.html

Title: VC VIP: Sending a Specific Number of TS1s in RECOVERY.EQUALIZATION_1 State in PCIe

https://solvnet.synopsys.com/retrieve/2984801.html

Title: VC VIP: Lane Reversal Using Symbol Log in PCIe

https://solvnet.synopsys.com/retrieve/3024336.html

Title: VC VIP: PCLK Frequency Is Not Aligned With PCLK Rate Driven by MAC DUT in SPIPE Interface in PCIe

https://solvnet.synopsys.com/retrieve/2787311.html

Title: VC VIP: Initiating Polling.Compliance by Load Board Pattern in PCIe

https://solvnet.synopsys.com/retrieve/2774336.html

Title: VC VIP: PCIe Gen3 Link Up Failure in PIPE Mode

https://solvnet.synopsys.com/retrieve/2616943.html

Title: VC VIP: Changing link_eval_feedback_figure_of_merit in PCIe

https://solvnet.synopsys.com/retrieve/2802535.html

Title: VC VIP: Checking Compliance Pattern Received by PCIe VIP

https://solvnet.synopsys.com/retrieve/2762224.html

Title: VC VIP: Check Data Link Layer Is Idle Before Initiating Speed Change in PCIe

https://solvnet.synopsys.com/retrieve/2778807.html

Title: VC VIP: Link Up Issue Due to phy_status Not Working Correctly in PCIe

https://solvnet.synopsys.com/retrieve/2492842.html

Title: VC VIP: Directed LTSSM State Transition to Loopback in PCIe

https://solvnet.synopsys.com/retrieve/2641488.html

Title: VC VIP: Valid Preset Value Is Rejected by PCIe

https://solvnet.synopsys.com/retrieve/2617473.html

Title: VC VIP: Using Real Values for LTSSM Timeouts in the PCIe Model

https://solvnet.synopsys.com/retrieve/2317906.html

Title: VC VIP: Controlling the Count of TS1 OS Transmitted in Polling.Active State in PCIe

https://solvnet.synopsys.com/retrieve/2680387.html

Title: VC VIP: Expected UVM_ERROR During Mid-Simulation Reset in PCIe

https://solvnet.synopsys.com/retrieve/2401213.html

Title: VC VIP: Understanding Why VIP Initiated ASPML1 Entry Might Not Be Working (PCIe)

https://solvnet.synopsys.com/retrieve/2317924.html

Title: VC VIP: Passive Monitor Error on DC Common Mode Voltage Requirements on datap and datan Signals in PCIe

https://solvnet.synopsys.com/retrieve/2803236.html

Title: VC VIP: Reconfiguring Link Width by Disabling rx_detection Random Lanes in PCIe

https://solvnet.synopsys.com/retrieve/2581581.html

Title: VC VIP: Negotiated Link Width Does Not Match With the Expected Link Width in PCIe

https://solvnet.synopsys.com/retrieve/2741849.html

Title: VC VIP: Does the PCIe VIP Support D States?
https://solvnet.synopsys.com/retrieve/2326026.html

Title: VC VIP: Using PIPE Signals for Error Scenarios in PCIe
https://solvnet.synopsys.com/retrieve/2674106.html

Title: VC VIP: Skipping Link Training in PCIe
https://solvnet.synopsys.com/retrieve/2439416.html

Title: VC VIP: Controlling the Step Command Behavior in Rx Margining
https://solvnet.synopsys.com/retrieve/2834230.html

Title: VC VIP: Forced Assertion of tx_data_vaild PIPE Signal Using PIPE Data Callback in PCIe
https://solvnet.synopsys.com/retrieve/2798117.html

Title: VC VIP: LTSSM State Changed to Recovery When Training Is Skipped in PCIe
https://solvnet.synopsys.com/retrieve/2621829.html

Title: VC VIP: Assertion of Rx_status in SPIPE Model in PCIe
https://solvnet.synopsys.com/retrieve/2768038.html

Title: VC VIP: Lane Skew Status in PCIe
https://solvnet.synopsys.com/retrieve/2793787.html

Title: VC VIP: Inserting/Deleting a Bit From a Serial Bit Stream Using PL Callback in PCIe
https://solvnet.synopsys.com/retrieve/2798742.html

Title: Configuring the ExpertIO Verilog SVC PCIe VIP for 8-bits PIPE
https://solvnet.synopsys.com/retrieve/2331593.html

Title: VC VIP: Illegal Pipe Interface Request in PCI Express
https://solvnet.synopsys.com/retrieve/2367786.html

Title: VC VIP: Avoiding FIFO Underflow Errors from the PCIe VIP
https://solvnet.synopsys.com/retrieve/2322996.html

Title: VC VIP: Using the PIPE InvalidRequest Signal in the PCIe VIP
https://solvnet.synopsys.com/retrieve/2350924.html

Title: VC VIP: Avoiding FIFO Overflow Errors from the PCIe VIP
https://solvnet.synopsys.com/retrieve/2322974.html

Title: VC VIP: Sending Payload Over PCIe Serial Link (Big Endian or Little Endian)
https://solvnet.synopsys.com/retrieve/2322544.html

Title: VC VIP: Blocking and Transmitting SKP Ordered Sets in the PCIe VIP
https://solvnet.synopsys.com/retrieve/2335767.html

Title: VC VIP: Sampling Order Sets in PCIe Active Component
https://solvnet.synopsys.com/retrieve/2620095.html

Title: VC VIP: Resolving the Decoder Error in PCIe

https://solvnet.synopsys.com/retrieve/2798761.html

Title: VP VIP: Guidelines to Configure recovery_speed_electrical_idle_time_ns Value in PCIe

https://solvnet.synopsys.com/retrieve/3072596.html

I.5 Methodology Testbench and Debug

Title: VC VIP: Native PA FSDB dump for PCIe

https://solvnet.synopsys.com/retrieve/2457446.html

Title: VC VIP: Understanding Why the PCIe VIP Issues Null Object Acess (NOA) Errors After Initial Model Reset

https://solvnet.synopsys.com/retrieve/2061817.html

Title: VC VIP: Writing to PCIe Configuration Space Using Backdoor Methods

https://solvnet.synopsys.com/retrieve/2061679.html

Title: VC VIP: Verifying the DUT Dropped Received Packet With EP=1 in PCIe

https://solvnet.synopsys.com/retrieve/2719555.html

Title: VC VIP: Calculating Delays for Sending Queued Packets on the PCIe VIP Link

https://solvnet.synopsys.com/retrieve/2113953.html

Title: VC VIP: Message Demotion is Not Working as Expected in the Pcie VIP

https://solvnet.synopsys.com/retrieve/1894682.html

Title: VMT PCIe VIP: Installing the VMT-PCIe VIP into a VC Installation

https://solvnet.synopsys.com/retrieve/2113996.html

Title: DesignWare Memory Models Overview (DWMM)

https://solvnet.synopsys.com/retrieve/1549344.html

Title: VC VIP: Configuring the PCIe Transaction Log to Show TLP Payload Data

https://solvnet.synopsys.com/retrieve/2060836.html

Title: VC VIP: Configuring the PCIe SVT VIP in EP Mode to Support More Than One Function

https://solvnet.synopsys.com/retrieve/1894507.html

Title: VC VIP: Resolving "Undefined System Task Call " Error Message in the PCIe VIP

https://solvnet.synopsys.com/retrieve/2359377.html

Title: VC VIP: License Warning for VIP-PCIE-VDB-SVT Key in PCIe

https://solvnet.synopsys.com/retrieve/2735218.html

Title: VC VIP: Viewing PCIe SVT Functional Coverage Information

https://solvnet.synopsys.com/retrieve/2284910.html

Title: PCIe SVT:Setting Up and Showcasing Global Shadow

https://solvnet.synopsys.com/retrieve/1588012.html

Title: VC VIP: Creating an Unprotected Install Version of the PCIe SVT VIP.

https://solvnet.synopsys.com/retrieve/1859683.html

Title: VC VIP: PCIe Configuration and Scoreboarding

https://solvnet.synopsys.com/retrieve/2087662.html

Title: MSI/MSI-X with the PCIe SVT VIP

https://solvnet.synopsys.com/retrieve/1450929.html

Title: VC VIP: Getting a Fatal Error Stating 8G is Not Supported

https://solvnet.synopsys.com/retrieve/2272176.html

Title: VC VIP: Getting a Handle to Packets with Errors off a PCIe TLP Analysis Port

https://solvnet.synopsys.com/retrieve/2240203.html

Title: VC VIP: VCS Compilation Flows with the PCIe VIP

https://solvnet.synopsys.com/retrieve/1914814.html

Title: VC VIP: Understanding PLL Recovery Reset Messages From the PCIe SVT VIP.

https://solvnet.synopsys.com/retrieve/1476941.html

Title: VC VIP: PCIe VIP Encryption Preventing VCS from Collecting Code Coverage

https://solvnet.synopsys.com/retrieve/2266126.html

Title: All VC VIPs: Model Not Installing into DESIGNWARE_HOME

https://solvnet.synopsys.com/retrieve/1920276.html

Title: VC VIP: PCIe VIP Compile and Work Around for VCS's -debug_pp Switch

https://solvnet.synopsys.com/retrieve/2246895.html

Title: VC VIP: User-Defined UVM Testbench Configuration Is Not Used by PCIe

https://solvnet.synopsys.com/retrieve/2515748.html

Title: VC VIP: Displaying All Payload Data in the PCIe VIP Transaction Log File

https://solvnet.synopsys.com/retrieve/1905837.html

Title: What is causing the Null object access [NoA] error in my pcie_svt UVM test bench?

https://solvnet.synopsys.com/retrieve/1432988.html

Title: VC VIP: Resolving SLI UVM ERROR (SLI: Detected older SLISERV version) in PCIe

https://solvnet.synopsys.com/retrieve/2515691.html

Title: VC VIP: Disabling Shadow Memory Checking in the PCIe VIP

https://solvnet.synopsys.com/retrieve/2061397.html

Title: VC VIP: Flagging UVM_ERROR Data Mismatch Errors Using the PCIe VIP

https://solvnet.synopsys.com/retrieve/2061428.html

Title: VC VIP: What is the Purpose of the PCIe VIP's Active/ Passive Setting?

https://solvnet.synopsys.com/retrieve/2057771.html

Title: VC VIP: PCIe Gen4 and PIPE 4.4

https://solvnet.synopsys.com/retrieve/2821992.html

Title: VC VIP: PCIe Gen4 Installation and Documentation Video

https://solvnet.synopsys.com/retrieve/2639274.html

Title: VC VIP: PCIe Gen4 Simplified Replay Timer

https://solvnet.synopsys.com/retrieve/2821766.html

Title: VC VIP: PCIe Gen4 Flow Control Scaling

https://solvnet.synopsys.com/retrieve/2821737.html

Title: VC VIP: Interfacing a PCIe UVM Agent to a DUT via model_instance_scope

https://solvnet.synopsys.com/retrieve/2305315.html

Title: VC VIP: Fixing PCIe Fatal Error -- Exceeding Number of 32-bt Pages

https://solvnet.synopsys.com/retrieve/2057854.html

Title: VC VIP for PCIe : Configuring a Model Video

https://solvnet.synopsys.com/retrieve/2649090.html

Title: PCIe SVT VIP: Detecting if the VIP is Idle

https://solvnet.synopsys.com/retrieve/1624052.html

Title: VC VIP: PCIe Gen4 EIEOS Format Change

https://solvnet.synopsys.com/retrieve/2821795.html

Title: PCIE SVT VIP : DISPLAY_NAME vs. instance name

https://solvnet.synopsys.com/retrieve/1623995.html

Title: VC VIP: Checking Out a Gen4 License in PCIe

https://solvnet.synopsys.com/retrieve/2418353.html

Title: Discovery PCIe SVT: Selecting PIPE clock rate and width

https://solvnet.synopsys.com/retrieve/1440701.html

Title: VC VIP: PCIe Gen4 Multiple Endpoint Mode

https://solvnet.synopsys.com/retrieve/2821953.html

Title: VC VIP: PCIe Gen4 Overview Video

https://solvnet.synopsys.com/retrieve/2639230.html

Title: VC VIP: PCIe Gen4 10-Bit Tags

https://solvnet.synopsys.com/retrieve/2822427.html

Title: VC VIP: Options for Controlling PCIe Logging Verbosity from the Command Line

https://solvnet.synopsys.com/retrieve/2357749.html

Title: VC VIP: PCIe Gen4 Licensing, Compilation, Speed Change Video

https://solvnet.synopsys.com/retrieve/2766596.html

Title: VC VIP : Reducing License Checkout Time for the PCIe VIP

https://solvnet.synopsys.com/retrieve/1827384.html

Title: VC VIP: Using the PCIe SPEC_VER Setting

https://solvnet.synopsys.com/retrieve/2057792.html

Title: PCIe SVT VIP: PIPE 4/4.2 Wiring for Gen1/2/3

https://solvnet.synopsys.com/retrieve/1520787.html

Title: VC VIP: PCIe Gen4 Up and Down Configure

https://solvnet.synopsys.com/retrieve/2821859.html

Title: VC VIP for PCIe : VIP to DUT Interconnect (Unified)

https://solvnet.synopsys.com/retrieve/2673035.html

Title: VC VIP: PCIe Product Training and Support Videos

https://solvnet.synopsys.com/retrieve/2639181.html

Title: VC VIP for PCIe : Logging and Debug Video

https://solvnet.synopsys.com/retrieve/2649134.html

Title: Pcie VMT: UVM examples give compilation errors

https://solvnet.synopsys.com/retrieve/1597817.html

Title: VC VIP: Specifying PCIe Transaction and Symbol Log Name and Path

https://solvnet.synopsys.com/retrieve/2058441.html

Title: VC VIP: Setting the Number of DWORD's Printed in a PCIe VIP Transaction Log

https://solvnet.synopsys.com/retrieve/1904812.html

Title: VC VIP: Creating Filters To Exclude Non-Revelant Coverage Reports in the PCIe VIP

https://solvnet.synopsys.com/retrieve/2369320.html

Title: VC VIP: Basic Compilation Errors During Initial Setup in PCIe

https://solvnet.synopsys.com/retrieve/2852019.html

Title: ExpertIO VIP: Printing Complete Payload in Transaction Log in PCIe

https://solvnet.synopsys.com/retrieve/2506076.html

Title: VC VIP: Resolving Uninitialized Memory Access Error in PCIe

https://solvnet.synopsys.com/retrieve/2796787.html

Title: VC VIP: Addressing Timescale Issues in the PCIe VIP

https://solvnet.synopsys.com/retrieve/2298935.html

Title: VC VIP: Resolving SERDES Locking Error in PCIe

https://solvnet.synopsys.com/retrieve/2470967.html

Title: All VIP: Using DW_LICENSE_OVERRIDE in DesignWare Verification IP

https://solvnet.synopsys.com/retrieve/2125477.html

Title: VC VIP: Avoiding PCIe DRIVER_COMMAND_TIMEOUT Errors (Verilog)

https://solvnet.synopsys.com/retrieve/2349570.html

Title: VC VIP: Disabling/Demoting Protocol Checks Using svt_err_check in PCIe

https://solvnet.synopsys.com/retrieve/2375558.html

Title: VC VIP: Enabling PCIe Gen4 Support in Verilog

https://solvnet.synopsys.com/retrieve/2617050.html

Title: VC VIP: Avoiding SetCPTPointer Related Error Caused When Reset Is Applied to PCIe VIP

https://solvnet.synopsys.com/retrieve/2389912.html

Title: VC VIP: Resolving pli.tab and msglog.o Errors in PCIe

https://solvnet.synopsys.com/retrieve/2334587.html

Title: VC VIP PCIe : Steps for demoting UVM VIP Error messages to Warning messages in an UVM based environment

https://solvnet.synopsys.com/retrieve/3087479.html

Title: VC VIP: PCIe Controls for Scrambling for Active Components

https://solvnet.synopsys.com/retrieve/3077247.html

Title: VC VIP: Resolving PCIe License Issue "UVM_ERROR: Encountered SLI error 'PCIE' followed by UVM_FATAL : License checkout failure. Authorization not granted for suite 'PCIE' "

https://solvnet.synopsys.com/retrieve/3060128.html

Title: VC VIP: Resolving the PCIe Integration Compile Error

https://solvnet.synopsys.com/retrieve/2603743.html

Title: VC VIP: Suppressing Demoted Messages in PCIe Log Files

https://solvnet.synopsys.com/retrieve/2800351.html

Title: VC VIP: Understanding UVM Warning Message Regarding Shadow Memory Configuration in the PCIe VIP

https://solvnet.synopsys.com/retrieve/2337518.html

I.6 Miscellaneous

See the following articles:

- VC VIP: Performing Backdoor Write to Memory in PCIe
- VC VIP: Tuning Presets and Coefficients in Equalization in PCIe
- VC VIP: Preserving the Memory When the PCIe VIP is Reset
- Spread Spectrum Clocking with ExpertIO PCIe SVC VIP
- Configuring the PCIe SVT SerDes VIP model to transmit 0's in its disabled state.
- VC VIP: Keeping the PCIe Model in the L0S State
- VC VIP: Programming a Mid-Sim Reset in PCIe VIP
- VC VIP: Generating Backpressure in PCIe
- VC VIP: Enabling Extended Tag Support in PCIe
- VC VIP: Using Status Classes in PCIe
- VC VIP: PCIE Redo-Equalization in the L0 state
- VC VIP: Controlling pipe_reset_n driven by MAC in PCIe
- VC VIP: Setting the Version of Your PCIe VIP PIPE Interface
- VC VIP: Preloading Memory in the PCIe Model
- VC VIP: Controlling SVC VIP From C Testbench Using DPI in PCIe
- VC VIP: Enabling Spread Spectrum Clocking in PCIe
- VC VIP: Accessing Expansion ROM in PCIe

- VC VIP: PLL Tolerance Adjustment When pll_lock Is Not Achieved in PCIe
- VC VIP: Enabling Transaction Ordering Support in PCIe
- VC VIP: Disabling Expansion ROM in PCIe
- PCIe VMT VIP: Initiating Various Types of Messages Using ASSERT_INTA as an Example
- VC VIP: Steps to Enter into and Exit from ASPM L1_2 sub-state (Low Power State) in PCIe
- VC VIP: Data Monitoring Options in PCIe
- VC VIP: Spread Spectrum Clocking in PCIe
- VC VIP: Resolving the Memory Error (ncelab: *E,CUVNPL: An argument list has been applied to an inappropriate object) in PCIe
- VC VIP: Toolbox Randomization Error in PCIe
- VC VIP: Simulation Run Failure on Questa Simulator in PCIe

J Reporting Problems

J.1 Introduction

This chapter outlines the process for working through and reporting VIP 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. This section outlines the process for working through and reporting problems. It shows how to use Debug Automation to enable all the debug capabilities of any VIP. In addition, the VIP provides a case submittal tool to help you pack and send all pertinent debug information to Synopsys Support.

J.2 Debug Automation

Every Synopsys model contains a feature called "debug automation". It is enabled through *svt_debug_opts* plusarg. The Debug Automation feature allows you to enable all relevant debug information. The following are critical features of debug automation:

- Enabled by the use of a command line run-time plusarg.
- Can be enabled on individual VIP instances or multiple instances using regular expressions.
- Enables debug or verbose message verbosity:
  - The timing window for message verbosity modification can be controlled by supplying `start_time` and `end_time`.
- Enables at one time any, or all, standard debug features of the VIP:
  - Transaction Trace File generation
  - Transaction Reporting enabled in the transcript
  - PA database generation enabled
  - Debug Port enabled
  - Optionally, generates a file name *svt_model_out.fsdb* when Verdi libraries are available

When the Debug feature is enabled, then all VIP instances that are enabled for debug will have their messages routed to a file named *svt_debug.transcript*.

J.3 Enabling and Specifying Debug Automation Features

Debug Automation is enabled through the use of a run-time plusarg named +*svt_debug_opts. T*his plusarg accepts an optional string-based specification to control various aspects Debug Automation. If this command control specification is not supplied, then the feature will default to being enabled on all VIP instances with the default options listed as follows:

Note the following about the plusarg:

- The command control string is a comma separated string that is split into the multiple fields.
- All fields are optional and can be supplied in any order.

The command control string uses the following format (white space is disallowed):

```
inst:<inst>,type:<string>,feature:<string>,start_time:<longint>,end_time:<longint>,verb
osity:<string>
```

The following table explains each control string:

Table J-1 Control Strings for Debug Automation plusarg

Field	Description
inst	Identifies the VIP instance to apply the debug automation features. Regular expressions can be used to identify multiple VIP instances. If this value is not supplied, and if a type value is not supplied, then the debug automation feature will be enabled on all VIP instances.
type	Identifies a class type to apply the debug automation features. When this value is supplied then debug automation will be enabled for all instances of this class type.
feature	Identifies a sub-feature that can be defined by VIP designers to identify smaller grouping of functionality that is specific to that title. The definition and implementation of this field is left to VIP designers, and by default it has no effect on the debug automation feature. (Specific to VIP titles)
start_time	Identifies when the debug verbosity settings will be applied. The time must be supplied in terms of the timescale that the VIP is compiled. If this value is not supplied, then the verbosity settings will be applied at time zero.
end_time	Identifies when the debug verbosity settings will be removed. The time must be supplied in terms of the timescale that the VIP is compiled. If this value is not supplied, then the debug verbosity remains in effect until the end of the simulation.
verbosity	Message verbosity setting that is applied at the `start_time`. Two values are accepted in all methodologies: DEBUG and VERBOSE. UVM and OVM users can also supply the verbosity that is native to their respective methodologies (UVM_HIGH/UVM_FULL and OVM_HIGH/OVM_FULL). If this value is not supplied then the verbosity defaults to DEBUG/UVM_HIGH/OVM_HIGH. When this feature is enabled, then all VIP instances that are enabled for debug will have their messages routed to a file named `svt_debug.transcript`.

Examples:

Enable on all VIP instances with default options:

```
+svt_debug_opts
```

Enable on all instances:

- containing the string "endpoint" with a verbosity of UVM_HIGH
- starting at time zero (default) until the end of the simulation (default):

```
+svt_debug_opts=inst:/.*endpoint.*/,verbosity:UVM_HIGH
```

Enable on all instances:

- starting at time 1000 until time 1500:

```
+svt_debug_opts=start_time:1000,end_time:1500,verbosity:VERBOSE
```

Enable debug feature on all instances using default options:

- By setting the macro DEBUG_OPTS to 1 in the command line, the debug feature is enabled on all instances using default options. The macro will enable the XMLs and Trace files.

```
gmake <testname> DEBUG_OPTS=1 PA=FSDB
```

Note

The DEBUG_OPTS option is available through the installed VIP examples, but if required, in customer environments, then a similar feature should be added to their environment.

The PA=FSDB option is available in public examples and is required to enable Verdi libraries, and that when this option is used, then the Debug Opts file will record VIP activity to a file named `svt_model_log.fsdb`.

In addition, the SVT Automated Debug feature will enable waveform generation to an FSDB file, if the Verdi libraries are available. When enabled this feature, it should cause the simulator to dump waveform information only for the VIP interfaces.

When this feature is enabled then all VIP instances that have been enabled for debug will have their messages routed to a file named `svt_debug.transcript`.

J.4 Debug Automation Outputs

The Automated Debug feature generates a *svt_debug.out* file. It records important information about the debug feature itself, and data about the environment that the VIPs are operating in. This file records the following information:

- The compiled timeunit for the SVT package
- The compiled timeunit for each SVT VIP package
- Version information for the SVT library
- Version information for each SVT VIP
- Every SVT VIP instance, and whether the VIP instance has been enabled for debug
- For every SVT VIP enabled for debug, a list of configuration properties that have been modified to enable debug will be listed
- A list of all methodology phases will be recorded, along with the start time for each phase

The following are the output files generated:

- *svt_debug.out*: It records important information about the debug feature itself, and data about the environment that the VIPs are operating. One file is optionally created when this feature is enabled, depending on if the Verdi libraries are available.
- *svt_debug.transcript*: Log files generated by the simulation run.
- *transaction_trace*: Log files that records all the different transaction activities generated by VIPs.
- *svt_model_log.fsdb*: Contains PA FSDB information (if the VIP supports this), and which contains other recorded activity. The additional information records signal activity associated with the VIP interface, TLM input (through SIPP ports), other TLM output activity, configurations applied to the VIP, and all callback activity (recorded by before and after callback execution).

J.5 FSDB File Generation

To enable FSDB writing capabilities, the simulator compile-time options and environment must be updated to enable this. The steps to enable this are specific to the simulator being used (the {LINUX/LINUX64} label needs to be replaced based on the platform being used). The ability to write to an FSDB file requires that the user supplies the Verdi dumper libraries when they compile their testbench. If these are not supplied then the VIP will not be enabled to generate the *svt_model_log.fsdb* file.

J.5.1 VCS

The following must be added to the compile-time command:

```
-debug_access
```

For more information on how to set the FSDB dumping libraries, see "Appendix B" section in *Linking Novas Files with Simulators and Enabling FSDB Dumping* guide available at *$VERDI_HOME/doc/linking_dumping.pdf*.

J.5.2 Questa

The following must be added to the compile-time command:

```
+define+SVT_FSDB_ENABLE -pli novas_fli.so
```

J.5.3 Incisive

The following must be added to the compile-time command:

```
+define+SVT_FSDB_ENABLE -access +r
```

J.6 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.

Enable the Debug Opts feature. For more information, see the "Debug Automation" on page 557.

J.7 Sending Debug Information to Synopsys

To help you debug testing issues, follow the given instructions to pack all pertinent debug information into one file which you can send to Synopsys (or to other users in your company):

1. Create a description of the issue under investigation. Include the simulation time and bus cycle of the failure, as well as any error or warning messages that are part of the failure.
2. Create a description of your verification environment. Assemble information about your simulation environment, making sure to include:
   - OS type and version
   - Testbench language (SystemVerilog or Verilog)
   - Simulator and version
   - DUT languages (Verilog)
3. Use the VIP case submittal tool to pack a file with the appropriate debug information. It has the following usage syntax:

```
$DESIGNWARE_HOME/bin/snps_vip_debug [-directory <path>]
```

The tool will generate a "<username>.<uniqid>.svd" file in the current directory. The following files are packed into a single file:

- FSDB
- HISTL
- MISC
- SLID
- SVTO
- SVTX
- TRACE
- VCD
- VPD
- XML

If any one of the above files are present, then the files will be saved in the "<username>.<uniqid>.svd" in the current directory. The simulation transcript file will not be part of this and it will be saved separately.

The -directory switch can be specified to select an alternate source directory.

4. You will be prompted by the case submittal tool with the option to include additional files within the SVD file. The simulation transcript files cannot be automatically identified and it must be provided during this step.

Note

For SVT, you must set the verbosity to UVM_HIGH/OVM_HIGH.

5. The case submittal tool will display options on how to send the file to Synopsys.

J.8 Limitations

Enabling DEBUG or VERBOSE verbosity is an expensive operation, both in terms of runtime and disk space utilization. The following steps can be used to minimize this cost:

- Only enable the VIP instance necessary for debug. By default, the `+svt_debug_opts` command enables Debug Opts on all instances, but the `'inst'` argument can be used to select a specific instance.
- Use the `start_time` and `end_time` arguments to limit the verbosity changes to the specific time window that needs to be debugged.