Verification Continuum™

VC Verification IP PCIe VMM User Guide

Version R-2021.03, March 2021

SYNOPSYS®

Contents

Preface

About This Manual

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

Note From the R-2021.03 release onwards, the documentation updates for the guides that are based on the SystemVerilog VMM designs will be suspended. For more information, see the UVM guides for the current updates on this VIP. Contact Synopsys support for any queries or clarifications.

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, "Creating the PLI Object in 32-bit and 64-bit Simulation Modes", describes how to build and compile the message log and PLI files.
- Chapter 4, "General Concepts", introduces the PCIe VIP within the VMM environment and describes the data objects and s that comprise the VIP.
- Chapter 5, "Verification Features", describes the verification features supported by PCIe VIP such as, Verification Planner and Protocol Analyzer.
- Chapter 6, "PCIe Verification Topologies", describes various ways the PCIe VIP can be connected with other s.
- Chapter 7, "Using the VC PCIe Verification IP", shows how to install and run a getting started example.
- Chapter 8, "Using the Transaction Layer", describes features of the Transaction Layer and how to use them.
- Chapter 10, "Functional Coverage", describes how to enable and use the functional coverage features.
- Chapter 11, "Using Callbacks", describes how to use callbacks with the PCIe SVT VMM VIP.
- Chapter 12, "Partition Compile and Precompiled IP", describes the PC and PIP features in Verification Compiler to optimize the compilation performance.
- Chapter 13, "Integrated Planning for VC VIP Coverage Analysis", describes how to enable and use the functional coverage features.

- Appendix A, “Protocol Checks”, outlines the process for working through and reporting VC PCIe issues.
- Appendix B, “PCIe Compile-time Parameters”, contains a table of parameters that can only be set before compilation, not at runtime.
- Appendix C, “Verilog Task/Parameter to SVT Class Mapping”, contains tables that show the correspondence between SVT classes and Verilog tasks or parameters..
- Appendix D, “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 D, “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 D, “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:

    https://www.synopsys.com/support/global-support-centers.html

Synopsys Statement on Inclusivity and Diversity

Synopsys is committed to creating an inclusive environment where every employee, customer, and partner feels welcomed. We are reviewing and removing exclusionary language from our products and supporting customer-facing collateral. Our effort also includes internal initiatives to remove biased language from our engineering and working environment, including terms that are embedded in our software and IPs. At the same time, we are working to ensure that our web content and software applications are usable to people of varying abilities. You may still find examples of non-inclusive language in our software or documentation as our IPs implement industry-standard specifications that are currently under review to remove exclusionary language.

1 Introduction

This chapter gives a basic introduction, overview and features of the Verification Compiler (VC) PCIe VMM Verification IP.

1.1 Introduction

The Synopsys 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 Verification Methodology Manual (VMM). 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 VMM.

1.3 References

For more information on PCIe Verification IP, refer to the Class Reference for Synopsys VC 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_vmm_class_reference/html/index.html

1.4 Product Overview

The PCIe VMM VIP is a suite of VMM-based verification transactors and groups that are compatible for use with SystemVerilog-Compliant testbenches. The Synopsys PCIe VIP suite simulates PCIe transactions using an active group as defined by the PCIe specification.

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

1.5 Features Supported

PCIe SVT VMM VIP currently supports the following verification functions:

- Functional coverage
- Protocol checking
- Protocol Analyzer
- Control on delays and timeouts
- Gen1, Gen2 and Gen3
- Requester, driver and target applications
- Built-in completer memory
- Transaction and Symbol logging
- Link reconfiguration
- Shadow memory with application-level scoreboard. Note: Shadow memory is limited to default behavior; settings are not changeable.
- Analysis ports and channels for connecting the group to the scoreboard, or any other component
- Error injections via Factory, callback, or scenario.

1.6 Features Not Supported Yet

- Gen4
- Passive Monitor
- MPCIe

1.6.1 Methodology Features

PCIe VIP currently supports the following methodology functions:

- VIP organized as a set of groups and applications
- Analysis ports and channels for connecting the group to the scoreboard, or any other transactors
- Error injections via Factory, callback, or scenario.

2 Installation and Setup

This section leads you through installing and setting up the VC VIP for 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:

1. Verifying the Hardware Requirements
2. Verifying Software Requirements
3. Preparing for Installation
4. Downloading and Installing
5. What's Next?

This chapter contains the following additional topics:

❖ Licensing Information
❖ Environment Variable and Path Settings
❖ Determining Your Model Version
❖ Integrating a VC VIP into Your Testbench
❖ Including and Importing Model Files into Your Testbench
❖ Compile-time and Runtime Options

Note If you encounter any problems with installing the VC PCIe, see *Customer Support*.

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 VC for PCIe VIP, check the support matrix for "SVT-based" VIP in the following document:

Support Matrix for SVT-Based Synopsys VC for 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.

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 VC 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 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 VC for 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, see *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
❖ Environment Variable and Path Settings
❖ Determining Your Model Version
❖ Integrating a VC VIP into Your Testbench

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 and Gen4.

- VIP-PCIE-SVT
- VIP-SOC-LIBRARY-SVT
- VIP-PCIE-G3-OPT-SVT (required only for Gen3 support)
- VIP-PCIE-G4-SVT (required only for Gen4 support)

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
- When G4 is being used, the VIP will consume the VIP-PCIE-G3-OPT-SVT and the VIP-PCIE-G4-SVT license keys

Only one license is consumed per simulation, regardless of how many VIP models are instantiated in the design.

The license check is in order and feature names as per the following steps:

1. check VIP-<suite>-SVT

2. check VIP-LIBRARY-SVT + DesignWare-Regression

Note: "+" means "AND" and all those features are required.

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.

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 VC VIP suites contain various transactors such as transceivers, masters, slaves, and monitors depending on the protocol. The setup process gathers together all the required transactor and group 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 VC PCIe VIP contains the following components:

- pcie_system_svt

  This is the name used for the entire set of sub-models.
- pcie_device_group_svt
- pcie_global_shadow_svt
- pcie_cfg_database_svt
- pcie_driver_app_svt
- pcie_io_target_svt
- pcie_group_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 , or the entire set of s 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
```

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_group_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 s are “top level” and will setup the entire suite. You have the choice to set up the entire suite, or just one 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 AXI 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 VMM example:
% $DESIGNWARE_HOME/bin/dw_vip_setup -path design_dir -e
      pcie_svt/tb_pcie_svt_vmm_intermediate_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.

2.9.2 Updating an Existing Model

To add an update to an existing model, perform the following steps:

1. Set the $DESIGNWARE_HOME environment variable to the latest version.
2. Navigate to the directory where the existing *design_dir* is present (and not into the *design_dir*).
3. Update the directory with the latest release by passing the model name, `"pcie_device_agent_svt"` to `-add` option while running the `dw_vip_setup` application.

   Example:

   ```
   $DESIGNWARE_HOME/bin/dw_vip_setup -path <existing_design_dir_name> -add
   pcie_device_agent_svt
   ```

Note In case of any backward incompatible changes in the VIP, you need to update the *design_dir* by reinstalling the example in same location and/or update the testbench files accordingly by referring to the backward incompatible changes listed in the *Release Notes*.

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. Following is a code list of the includes and imports for PCIe:

```
In the program block
// Load the PCIE VMM code
`include "svt_pcie.vmm.pkg"

program pcie_svt_basic;

  // Import the SVT library
```

```
  import svt_vmm_pkg::*;

  // Import the PCIE VIP
  import svt_pcie_vmm_pkg::*;

  vmm_log log = new("program", "pcie_svt_intermediate");

  // Load the environment files
  `include "pcie_device_intermediate_env.sv"

  // Test list
  `include "ts.base_pipe_test.sv"
  `include "ts.base_serdes_test.sv"
  `include "ts.base_pma_test.sv"
  `include "ts.directed_pipe_test.sv"
  `include "ts.directed_serdes_test.sv"
  `include "ts.directed_pma_test.sv"

  //Declare the environment
  pcie_device_intermediate_env env;

  initial begin
    //Construct the environment
    env = new("env");

    vmm_simulation::list();

    //Run tests specified at runtime using the +vmm_test
    vmm_simulation::run_tests();
  end
endprogram

In top file
/** Include the program block file
`include "pcie_svt_vmm_intermediate_sys_tb.sv"

/** Defines required for PCIE SVC */
`define PCIESVC_MEM_PATH                     test_top.mem0
`define EXPERTIO_PCIESVC_GLOBAL_SHADOW_PATH  test_top.global_shadow0
`define SVC_RANDOM_SEED_SCOPE                test_top.global_random_seed

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

  /** Signal to generate the reset */
  bit reset;

  /** PCIE Device instantiation model showing interface / wiring between 2 PCIE Devices
*/
  `include "svt_pcie_device_pipe_x4.sv"

  /** Global random seed */
  int unsigned global_random_seed;
```

```
    // Instantiate global memory seen by everything
    svc_mem #( .DISPLAY_NAME( "mem0." ) ) mem0();

    // Instantiate Global Memory Shadow */
     pciesvc_global_shadow #( .DISPLAY_NAME( "global_shadow0." ) ) global_shadow0();

    /** Set up the Global random seed */
    `ifndef SVC_RANDOM_BY_THREAD
      initial
        begin : L_RandSeed
    `ifdef SVC_RANDOM_SEED
          global_random_seed = `SVC_RANDOM_SEED ;
    `else
          global_random_seed = 0;
    `endif
      end
    `endif

  //Setup the reset.
    initial begin
      $timeformat(-12,2, " ps", 8);
      #5;
      test_top.reset = 1;
      #200;
      test_top.reset = 0;
    end
```

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>

For example VCS:

```
+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

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

vcs_run_options

Note: “scenario” is the vmm test name you pass to VCS

3 Creating the PLI Object in 32-bit and 64-bit Simulation Modes

This chapter contains the following topics:

- ❖ Compiling the Messaging PLI for the PCIe Model
- ❖ Compiling the msglog.o and PLI Files

3.1 Compiling the Messaging PLI for the PCIe Model

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

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

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

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

The purpose of the msglog PLI is to provide a message logging facility in the underlying VC SVT model and to provide a mechanism for all s in the model to use a common logging facility. The msglog PLI tasks and functions are written in C and they are for logging various levels of messages. These tasks and functions should be either included in a build by dynamically linking a library that contains the compiled object file msglog.o, or explicitly statically linked to msglog.o.

Before you can use the pcie_svt model, you must build the msglog.o object file and the PLI files. The PLI generation is needed for legacy support of the msglog feature and for time 0 messages generated by the underlying Verilog model.

If you run the example scripts included with the installation, those files are generated automatically. You can use either the run script or the Makefile to run the example scripts. Information about building the msglog.o and PLI files is provided in the prescript file.

If you do not run the example scripts, you can use the following procedure to build the msglog.o and PLI files:

Convert the Verilog parameters and the defines that msglog uses from the ${*design-dir*}/src/verilog/vcs/svc_util_parms.v file.You can use the param2def.sh script to do that. The param2def.sh script converts Verilog-style parameters to C-style #defines. Use the following command to run the param2def.sh script:

```
& ${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/bin/param2def.sh < ${design-
dir}/src/verilog/vcs/svc_util_parms.v > svc_util_parms.h
```

3.2 Compiling the msglog.o and PLI Files

The next step is to compile the msglog and the PLI files. You can compile these files on either a 32-bit or 64-bit machine. Please note that cc defaults to 64 bit. The m32 switch forces 32-bit operation on a 64-bit machine. If the compilation is done on a 32-bit machine, the m32 flag is not needed.

1. Set ccflags for the compile depending on the machine type. These flags will match the setting done in the run script from the example testbench.

   For the Linux platform, set the ccflags to the following:

   64 bits: `set ccflags = ""`

   32 bits: `set ccflags = "-m32"`

2. Compile with the correct include files and switches to generate the msglog.o file.

   a. For VCS | VCS MX:

      32 bits:

      ```
      cc -c ${ccflags} -I. -I${VCS_HOME}/include -
      I${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/include -DVCS_VERILOG
      -DUSE_VPI=1 ${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/src/msglog.c
      -o msglog.o
      ```

      64 bits:

      ```
      cc -c ${ccflags} -I. -I${VCS_HOME}/include -
      I${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/include -DVCS_VERILOG
      -DUSE_VPI=1 -DPLI_64_BIT
      ${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/src/msglog.c -o msglog.o
      ```

      NOTE for ccflags_dyn: For the Linux platform, set the ccflags_dyn to the following:

         64 bits: `set ccflags_dyn = "-fPIC"`

         32 bits: `set ccflags_dyn = "-m32 -fPIC"`

   b. For MTI:

      ```
      cc -c ${ccflags_dyn} -I. -I${MTI_HOME}/include
      -I${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/include -DQUESTA
      ${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/src/msglog.c -o msglog.o
      ```

   c. For NC:

      ```
      cc -c ${ccflags_dyn} -I. -I${CDS_INST_DIR}/tools/inca/include
      -I${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/include -DNC_VERILOG
      ${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/src/msglog.c -o msglog.o
      ```

3. Compile the verisuer file, which registers the PLI information with the simulator.

   a. For VCS | VCS MX:

      ```
      ${VCS_HOME}/bin/veriuser_to_pli_tab -include ${VCS_HOME}/include
      ${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/src/veriuser.c > pli.tab || rm -f
      pli.tab
      ```

   b. For MTI:

```
cc -c ${ccflags_dyn} -I. -I${MTI_HOME}/include
-I${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/include -DQUESTA
${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/src/dyn_veriuser.c
-o dyn_veriuser.o
```

   c. For NC:

```
cc -c ${ccflags_dyn} -I. -I${CDS_INST_DIR}/tools/inca/include
-I${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/include -DNC_VERILOG
${DESIGNWARE_HOME}/vip/svt/pcie_svt/latest/C/src/dyn_veriuser.c
-o dyn_veriuser.o
```

   NOTE for ldflags_dyn: For the linux platform, set the ldflags_dyn to the following:

   64 bits: `set ldflags_dyn = "-shared"`

   32 bits: `set ldflags_dyn = "-m32 -shared"`

4. Compile the msglog and dyn_veriuser files to generate the PLI file.

   a. For MTI:

   Once the dyn_veriuser.o file is generated from step 3, run this compile:

```
cc ${ldflags_dyn} -o dyn_mtipli.so msglog.o dyn_veriuser.o
```

   b. For NC:

   Once the dyn_veriuser.o file is generated from step 3, run this compile:

```
cc ${ldflags_dyn} -o dyn_ncvpli.so msglog.o dyn_veriuser.o
```

Once the msglog and PLI information are generated, you can simulate with the model.

4 General Concepts

This chapter describes the usage of the VC for PCIe VIP in a VMM environment, and its user interface.

4.1 Introduction to VMM

VMM 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 VC for PCIe VIP in VMM 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 VMM. For more information:

For the IEEE SystemVerilog standard, see:

- IEEE Standard for SystemVerilog-Unified Hardware Design, Specification, and Verification Language

For essential guides describing VMM as it is represented in SystemVerilog, along with a Class Reference, see:

https://solvnet.synopsys.com/dow_retrieve/latest/VCS/ni/vmm_user_guide.pdf

4.2 VC for PCIe VMM Interface

The VC VMM VIP for PCIe is a suite of advanced verification s and data objects based on SystemVerilog VMM-compliant technology. The VC VMM PCIe VIP is based on the following VMM group architecture and data objects.

The svt_pcie_device_group object defines a VMM group that contains the following vmm transactors for PCIe applications:

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

The svt_pcie_device_group object contains a VMM group named svt_pcie_group. The VC for PCIe VMM subenvironment contains active PCIe applications to send and receive PCIe packets as well as VMM Multi-stream-scenario generators and Scenarios. Your testbench will interact mainly with VMM MS-Scenario generators, which can use your own scenarios.

svt_pcie_group

The svt_pcie_group object defines a VMM group that contain a layered stack of vmm_xactors for the Physical, Link, and Transaction layers of the PCIe protocol. The VC for PCIe VMM group contains active VC for PCIe Physical, Link, and Transaction vmm_xactors, as well as VMM scenario generators and scenarios. Your testbench will interact mainly with VMM scenarios generators which can use your own scenarios.

4.2.1 VMM Transactors of the PCIe Device Subenvironment

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

4.2.2 VMM Transactors of the PCIe Group

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

4.2.3 VMM Multi-stream Scenario Generators and Channels of the PCIe Device Group

- **driver_transaction_ms_scenario_gen[int]** – An object of type vmm_ms_scenario_gen that generates scenarios/transactions of type svt_pcie_driver_app_transaction. A vmm channel instance driver_transaction_chan[0] with the name "driver_transaction_chan0" is registered to driver_transaction_ms_scenario_gen[0] instance of driver_transaction_ms_scenario_gen[int] array

  Generator Access: <env_inst_name>.<device_group_inst_name>.
  driver_transaction_ms_scenario_gen[0]

  Channel Access in ms-scenarios: get_channel("driver_transaction_chan0")
- **driver_svc_ms_scenario_gen[int]** – An object of type vmm_ms_scenario_gen that generates service scenarios/transactions of type svt_pcie_driver_app_service. A vmm channel instance driver_svc_chan[0] with the name "driver_svc_chan0" is registered to driver_svc_ms_scenario_gen[0] instance of driver_svc_ms_scenario_gen[int] array

  Generator Access: <env_inst_name>.<device_group_inst_name>. driver_svc_ms_scenario_gen[0]

  Channel Access in ms-scenarios: get_channel("driver_svc_chan0")

- **requester_svc_ms_scenario_gen** – An object of type vmm_ms_scenario_gen that generates service scenarios/transactions of type svt_pcie_requester_app_service. A channel requester_svc_chan with the name "requester_svc_chan" is registered to requester_svc_ms_scenario_gen

  Generator Access: <env_inst_name>.<device_group_inst_name>. requester_svc_ms_scenario_gen

  Channel Access in ms-scenarios: get_channel("requester_svc_chan")

- **target_svc_ms_scenario_gen[int]–** An object of type vmm_ms_scenario_gen that generates service scenarios/transactions of type svt_pcie_target_app_service. A vmm channel instance target_svc_chan[0] with the name "targetr_svc_chan0" is registered to target_svc_ms_scenario_gen[0] instance of target_svc_ms_scenario_gen[int] array

  Generator Access: <env_inst_name>.<device_group_inst_name>. target_svc_ms_scenario_gen [0]

  Channel Access in ms-scenarios: get_channel("target_svc_chan0")

- **io_target_svc_ms_scenario_gen** – An object of type vmm_ms_scenario_gen that generates service scenarios/transactions of type svt_pcie_io_target_service. A channel io_target_svc_chan with the name "io_target_svc_chan" is registered to io_target_svc_ms_scenario_gen

  Generator Access: <env_inst_name>.<device_group_inst_name>. io_target_svc_ms_scenario_gen

  Channel Access in ms-scenarios: get_channel("io_target_svc_chan")

- **mem_target_svc_ms_scenario_gen** – An object of type vmm_ms_scenario_gen that generates service scenarios/transactions of type svt_pcie_mem_target_service. A channel mem_target_svc_chan with the name "mem_target_svc_chan" is registered to io_target_svc_ms_scenario_gen

  Generator Access: <env_inst_name>.<device_group_inst_name>. mem_target_svc_ms_scenario_gen

  Channel Access: get_channel("mem_target_svc_chan")

- **cfg_database_svc_ms_scenario_gen** – An object of type vmm_ms_scenario_gen that generates service scenarios/transactions of type svt_pcie_cfg_database_service. A channel cfg_database_svc_chan with the name "cfg_database_svc_chan" is registered to cfg_database_svc_ms_scenario_gen

  Generator Access: <env_inst_name>.<device_group_inst_name>. cfg_database_svc_ms_scenario_gen

  Channel Access in ms-scenarios: get_channel("cfg_database_svc_chan")

- **pcie_device_ms_scenario_gen** – Device level multi-stream scenario generator object of type vmm_ms_scenario_gen.

  Any other scenario or multi-stream scenario generators can be registered to this and the streams can be controlled from this device level top ms-scenario generator

  Generator Access: <env_inst_name>.<device_group_inst_name>. pcie_device_ms_scenario_gen

4.2.4 VMM Multi-stream Scenario Generators and Channels of the PCIe Group

- **tlp_ms_scenario_gen** – An object of type vmm_ms_scenario_gen that generates Transaction layer data scenarios/transactions of type svt_pcie_tlp. A channel tlp_chan with the name "tlp_chan" is registered to this generator.

  Generator Access: <env_inst_name>.<device_group_inst_name>. pcie_group.tlp_ms_scenario_gen

  Channel Access in ms-scenarios: get_channel("tlp_chan")

- **tl_svc_ms_scenario_gen** – An object of type vmm_ms_scenario_gen that generates Transaction layer service scenarios/transactions of type svt_pcie_tl_service. A channel tl_svc_chan with the name "tl_svc_chan" is registered to this generator.

  Generator Access: <env_inst_name>.<device_group_inst_name>.
  pcie_group.tl_svc_ms_scenario_gen

  Channel Access in ms-scenarios: get_channel("tl_svc_chan")
- **dllp_ms_scenario_gen** – An object of type vmm_ms_scenario_gen that generates Data Link layer data scenarios/transactions of type svt_pcie_dllp. A channel dllp_chan with the name "dllp_chan" is registered to this generator.

  Generator Access: <env_inst_name>.<device_group_inst_name>.
  pcie_group.dllp_ms_scenario_gen

  Channel Access: get_channel("dllp_chan")
- **dl_svc_ms_scenario_gen** – An object of type vmm_ms_scenario_gen that generates Data Link layer service scenarios/transactions of type svt_pcie_dl_service. A channel dl_svc_chan with the name "dl_svc_chan" is registered to this generator.

  Generator Access: <env_inst_name>.<device_group_inst_name>.pcie_group.
  dl_svc_ms_scenario_gen

  Channel Access in ms-scenarios: get_channel("dl_svc_chan")
- **pl_svc_ms_scenario_gen** – An object of type vmm_ms_scenario_gen that generates Phy layer service scenarios/transactions of type svt_pcie_pl_service. A channel pl_svc_chan with the name "pl_svc_chan" is registered to this generator.

  Generator Access: <env_inst_name>.<device_group_inst_name>.pcie_group.
  pl_svc_ms_scenario_gen

  Channel Access in ms-scenarios: get_channel("pl_svc_chan")

4.2.5 Configuration Data Objects

Configuration data objects are abstracted data objects that represent the content of VC for 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_mpl_phy_configuration

4.2.6 Status Data Objects

Status data objects are abstracted data objects that represent the content of VC for PCIe VIP statistics. 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

4.2.7 VMM Data Objects

The VIP supports extending VMM 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 data 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

4.3 VC for 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
```

4.4 Compliance Patterns

The compliance and modified compliance patterns defined in the PCIE specification contain scenarios 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 Verification Features

This chapter describes the various verification features available with the Synopsys VC PCIe Verification IP.

5.1 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 vmm_test;
…
   virtual function void task gen_config_ph();
      super.gen_config_ph();
      . . .
      endpoint_cfg.port_cfg.enable_transaction_logging = 1;
      endpoint_cfg.port_cfg.transaction_log_filename = "pcie_trans.log";
      . . .
endfunction
```

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

5.1.1 Printing TLP Payload Data to a Transaction Log File

The intermediate example shows you 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.

```
      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.

```
/** 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.

5.2 The Symbol Logger

One log file is created per simulation. All groups share the log file. Each group must be enabled independently as shown in the following example. 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 group set the filename

Example 5-1

```
class example extends vmm_test;
   . . .
   virtual function void task gen_config_ph;
      super.gen_config_ph;
      . . .
      root_cfg.port_cfg.enable_symbol_logging = 1;               // Enable for RC
      endpoint_cfg.prot_cfg.enable_symbol_logging = 1;           // Enable for EP
     root_cfg.port_cfg.transaction_log_filename = "symbol.log"; // Recommended to only
set the filename once
      . . .
   endfunction
endclass
```

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

5.3 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

6 PCIe Verification Topologies

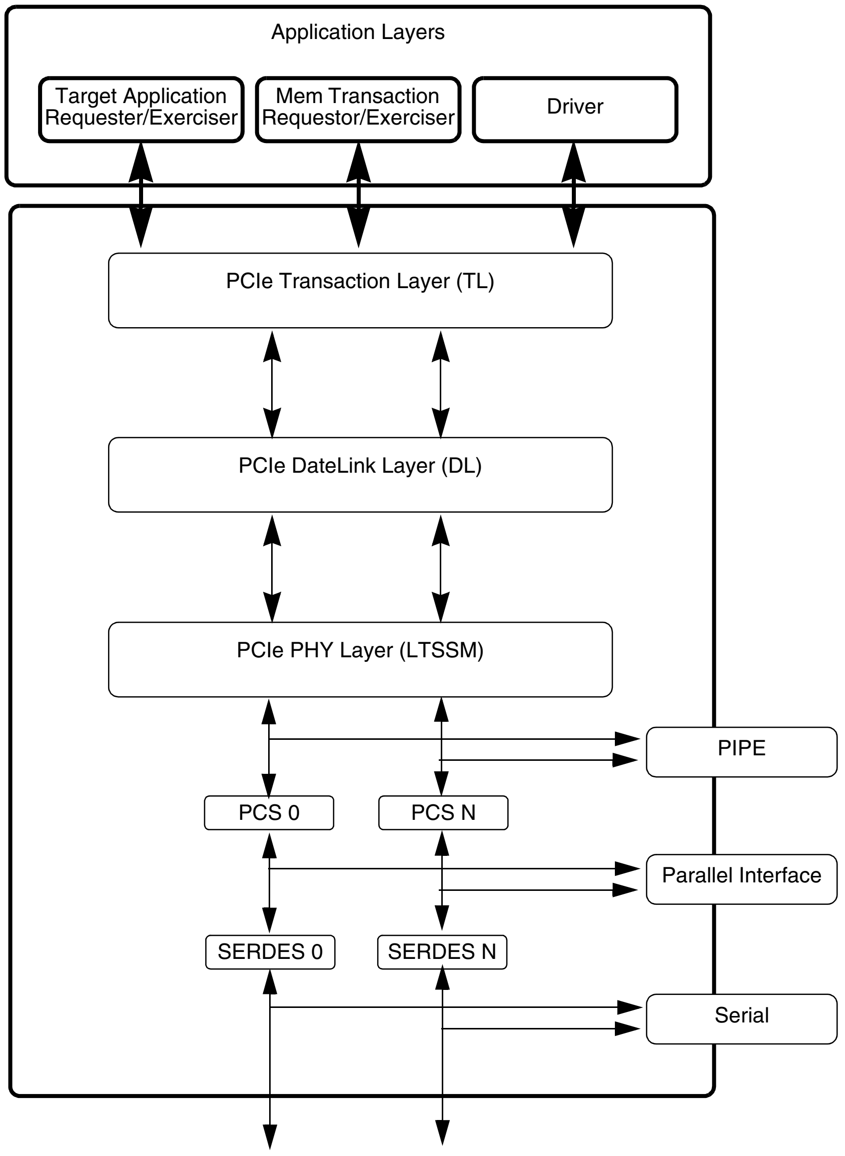
This chapter shows the supported interfaces of the VC for PCIe VIP.

6.1 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 6-1 shows all the supported layers in the VC for 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 6-1 VC for PCIe VIP structure



6.1.1 Choosing an Instantiation Model

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.

Models are briefly described in Table 6-1.

Table 6-1 Descriptions of models

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

Note Models supporting 8G have "_8g" appended to the model name. 8G models should not be used without the corresponding Gen 3 license add-on.

6.2 SERDES Interface

If the serial interface is used, the SERDES layer will be provided by the VIP. The instantiation model is named pciesvc_device_serdes_x[4,8,32]_model_[8g].

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 6-2 shows the layers supported by the VC for PCIe VIP when using the serial interface along with the DUT serial interface.

Figure 6-2 PCIe VIP serial interface layers

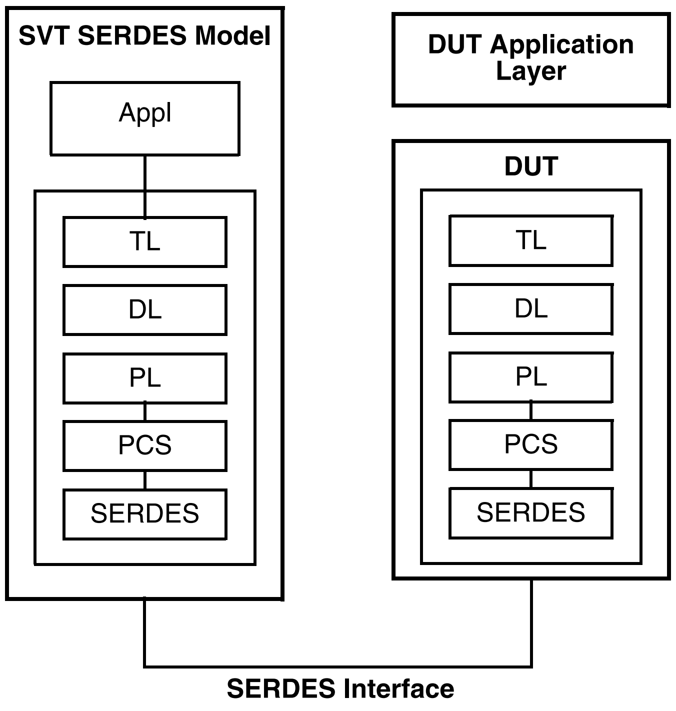


Figure 6-3 shows the SERDES module ports.

Figure 6-3 SERDES module ports

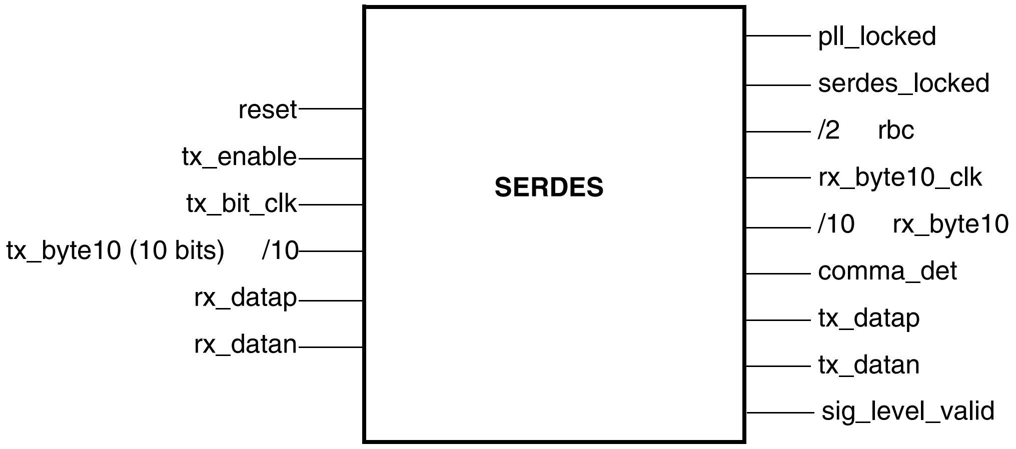


Table 6-2 pciesvc_model SERDES ports

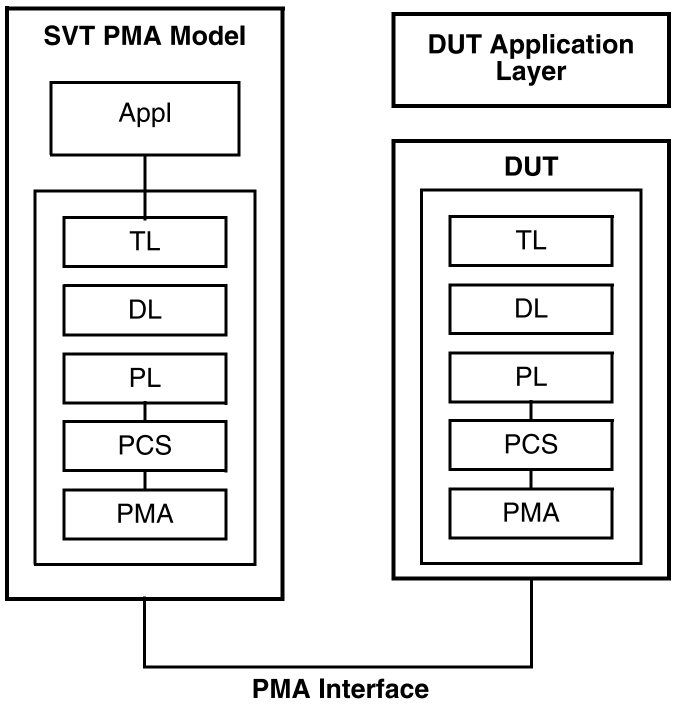
Model Interface Type	Port Name	I/O	Description
SERDES Interface.	Reset	I	Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning. Reset must be deasserted at time 0 such that a posedge is seen by the VIP
	phy*n*_rx_datap	I	Serial datap in to VIP
	phy*n*_rx_datan	I	Serial datan in to VIP
	phy*n*_tx_datap	I	serial datap out of VIP
	phy*n*_tx_datan		serial datan out of VIP

6.3 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.

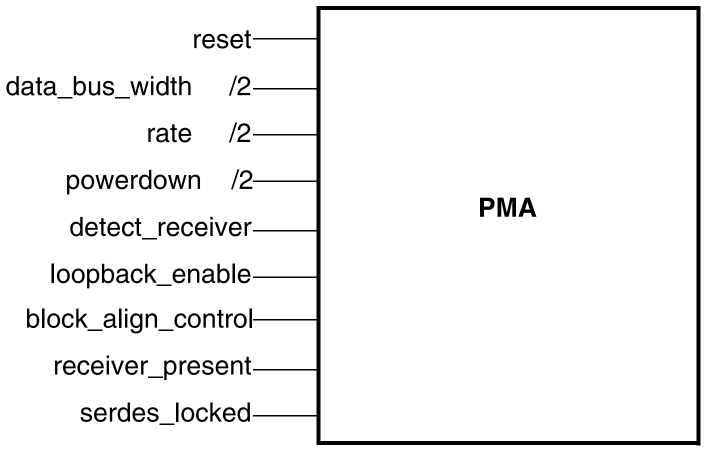
The Physical Coding Sublayer (PCS) checks for commas and performs symbol lock at 2.5GT/s, 5GT/s, and 8GT/s (Optional). It also does 8b/10b encoding at 2.5GT/s and 5GT/s and inserts data sync headers at 8GT/s. The PCS also contains the elastic buffer which inserts/deletes SKP symbols from skip ordered sets. The PCS task pair interface and module inputs/outputs comply with Intel's PIPE interface specification. Figure 6-4 shows the layers supported by the VC for PCIe VIP when using the PMA along with the DUT PMA interface.

Figure 6-4 VC for PCIe VIP PMA layers



The PMA module ports are shown in Figure .

Figure 6-5 PMA Module Ports



6.4 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 6-3, 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

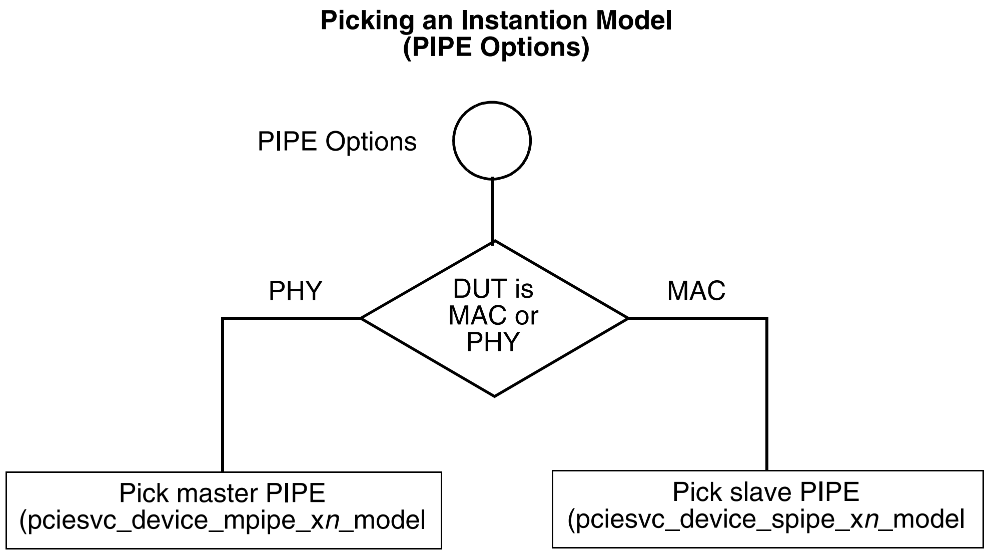
4.0

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 6-6):

❖ 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 6-6 Instantiation model PIPE options


Picking an Instantion Model
(PIPE Options)
PIPE Options
PHY
DUT is MAC or PHY
MAC
Pick master PIPE
(pciesvc_device_mpipe_xn_model
Pick slave PIPE
(pciesvc_device_spipe_xn_model

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 pciesvc_device_mpipe or the pciesvc_device_spipe. This diagram should help to select the correct instantiation model once the DUT configuration is known.

6.4.1 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 6-7. 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 6-7 PCIe transceiver acting as a DUT's PHY to test the MAC

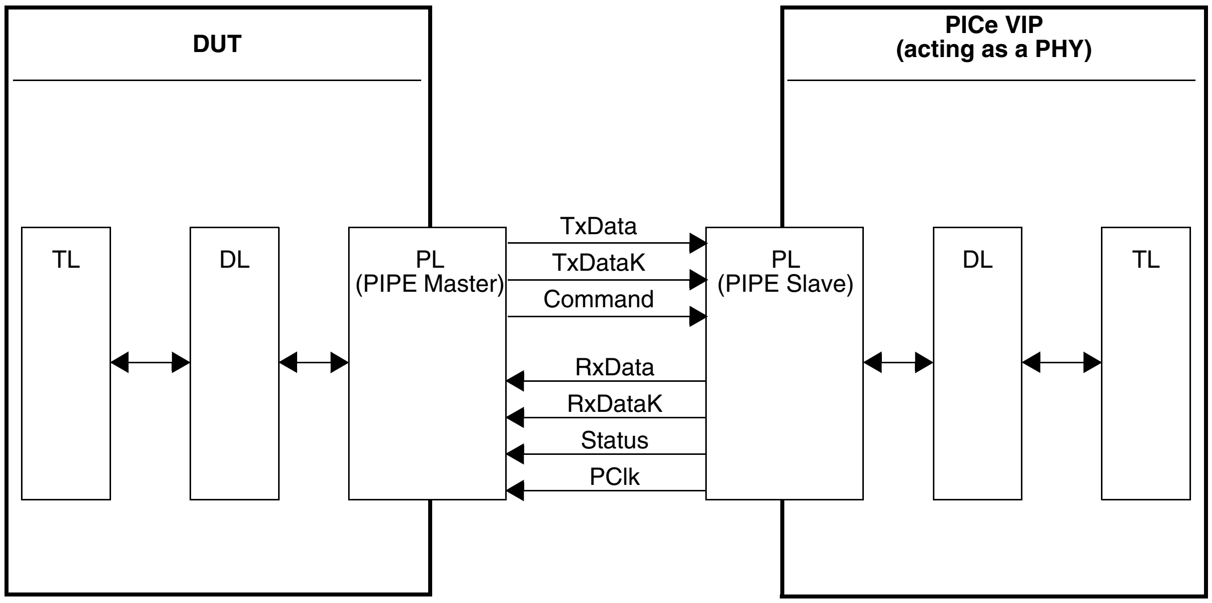


Table 6-3 pciesvc_model SPIPE ports

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	Active high master reset. Must only be asserted for 100ns ONCE per simulation at the beginning. Reset must be deasserted at time 0 such that a posedge is seen by the VIP.
	attached_pipe_reset_*n*	I	PIPE reset signal from the mac.
	pipe_clk	O	PIPE clock generated by the PIPE slave for use by the DUT mac.
	attached_powerdown[1:0]	I	Command from the DUT MAC to change power state of the phy.
	attached_rate[1:0]	I	Link signaling rate control from the DUT MAC. NOTE: Port width is 1 bit for PIPE2 and 2 bits for PIPE3.
	attached_txdetectrx	I	Command from the DUT mac to detect receiver.
	attached_block_align_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_*n*	O	Per lane receive data into the DUT. Supports 8, 16, and 32bit bus widths
	attached_rx_data_k_*n*	O	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	attached_rx_status_*n*[2:0]	O	Per lane receiver status. See PIPE spec for encodings.
	attached_rx_valid_*n*	O	Per lane li nd ica ti on that data coming into the DUT is valid and symbol lock has been achieved.
	attached_rx_elec_idle_*n*	I/O	Per lane indication that electrical idle is being received into the DUT

Table 6-3 pciesvc_model SPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
"Slave" PIPE Version 3 Interface (continued)	attached_invert_rx_polarity_*n*	I	Per lane indication to perform polarity inversion on data into the DUT
	attached_tx_data_*n*[31:0]	I	Per lane transmit data out from the DUT.
	attached_tx _data_k_*n*[3:0]	I	Per lane transmit_control from the DUT.
	attached_tx_ei_code_*n*[31:0]	I	Per lane error injection code. Indicates to the phy that an EI was either injected, or phy needs to inject the EI indicated by the code. For VIP only. No connection to DUT.
	attached_tx_compliance_*n*	I	Per lane transmit compliance pattern enable from the DUT
	attached_tx_elect_idle_*n*	I	Per lane transmit electrical idle/transmit enable from the DUT Note: Any unused ports must be tied to a 1.
"Slave" PIPE Version 4 interface The signals prefixed "attached_" are relative to the DUT and are designed to connect up in a slave like manner to a MAC DUT. The 8G models require a PIPE4 interface, hence all models in the InstantionModels directory with the _8g suffix use the PIPE4.	reset	I	Active high master reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
	attached_pipe_reset_*n*	I	PIPE reset signal from the mac.
	pipe_clk	O	PIPE clock generated by the PIPE slave for use by the DUT mac.
	max_pclk	O	Maximum pclk rate for a given speed.
	attached_powerdown[2:0]	I	Command from the DUT MAC to change power state of the phy.
	attached_rate[1:0]	I	Link signaling rate control from the DUT MAC. NOTE: port width is 1 bit for PIPE2 and 2 bits for PIPE3.
	attached_pclk_rate[2:0]	I	PCLK rate requested by the mac.
	attached_txdetectrx	I	Command from the DUT mac to detect receiver.
	attached_block_align_control	I	Controls slave block alignment. NOTE: 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.
	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.

Table 6-3 pciesvc_model SPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
	attached_rx_standby[31:0]	I	Controls whether the phy RX is active when in L0 or L0s.
	attached_phy_status[31:0]	O	Phy response to rate or power change. Bit 0 only used for PIPE2.
	attached_rx_standby_status[31:0]	O	Reflects the active/standby state of the rx receiver.
	attached_data_bus_width[1:0]	O	Defines the bus width of per lane data.
	attached_rx_data_*n*[	O	Per lane receive data into the DUT. Supports 8, 16, and 32bit bus widths.
	attached_rx_data_k_*n*[	O	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	attached_rx_status_*n*[[2:0]	O	Per lane receiver status. See PIPE spec for encodings.
	attached_rx_valid_*n*[	O	Per lane indication that data coming into the DUT is valid and symbol lock has been achieved.
	attached_rx_data_valid_*n*[	O	Per lane indication that data coming into the DUT is valid. Used primarily for rate matching. NOTE: PIPE3 only.
	attached_rx_elec_idle_*n*	IO	Per lane indication that electrical idle is being received into the DUT
	attached_rx_start_block_*n*[	O	Per lane indication that data transmitted by the DUT is the first byte of data in the block. NOTE: PIPE3 only.
	attached_rx_sync_header_*n*[[1:0]	O	Per lane block sync header. NOTE: PIPE3 only.
	attached_invert_rx_polarity_*n*[	I	Per lane indication to perform polarity inversion on data into the DUT.
	attached_tx_data_*n*[[31:0]	I	Per lane transmit data out from the DUT.
	attached_tx _data_k_*n*[[3:0]	I	Per lane transmit_control from the DUT.
	attached_tx_ei_code_*n*[[31:0]	I	Per lane error injection code. Indicates to the phy that an EI was either injected, or phy needs to inject the EI indicated by the code. For SVC only. No connection to DUT.
	attached_tx_compliance_*n*[	I	Per lane transmit compliance pattern enable from the DUT
	attached_tx_elect_idle_*n*[	I	Per lane transmit electrical idle/transmit enable from the DUT.Note: Any unused ports must be tied to a 1.

Table 6-3 pciesvc_model SPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
	attached_tx_data_valid_*n*[	I	Per lane indication that data transmitted to the SVC is valid. Used primarily for rate matching. NOTE: PIPE3 only.
	attached_tx_start_block_*n*[	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_*n*[[1:0]		Per lane block sync header. NOTE: PIPE3 only.
	attached_local_tx_preset_coefficients*n* [[17:0]		Coefficients returned from a coefficient lookup request.
	attached_link_eval_feedback_figure_of _merit*n*[7:0]	O	Figure of merit value from a link equalization evaluation request.
	attached_link_eval_feedback_direction _change*n*[5:0]	O	Coefficient direction feedback from rx link eval request.
	attached_local_fs*n*[5:0]	O	Provides the FS value for the phy.
	attached_local_lf*n*[5:0]	O	Provides the LF value for the phy.
	attached_local_tx_coefficients_valid*n*	O	Indicates that the values in local_tx_preset_coefficients[] are valid.
	attached_tx_deemph*n*[17:0]	I	Selects transmitter deemphasis.
	attached_local_preset_index*n*[3:0]	I	Index for the local phy preset coefficients requested by the mac.
	attached_rx_preset_hint*n*[2:0]	I	RX preset hint for the receiver.
	attached_get_local_preset_coefficients *n*	I	Request a preset to coefficient mapping lookup.
	attached_rx_eq_eval*n*	I	Request from the mac for the receiver to perform an equalization evaluation.
	attached_invalid_request*n*	I	Indicates the link eval feedback requested was out of range.

6.5 PIPE Interface, MAC VIP and PHY DUT (MPIPE)

When enabled to act as a MAC, tthe model supports validation of the PIPE or SERDES interface (encoding, serialization, and so on) to a user's PHY. This is shown in Figure 6-8.

Figure 6-8 PCIe transceiver acting as a DUT’s MAC

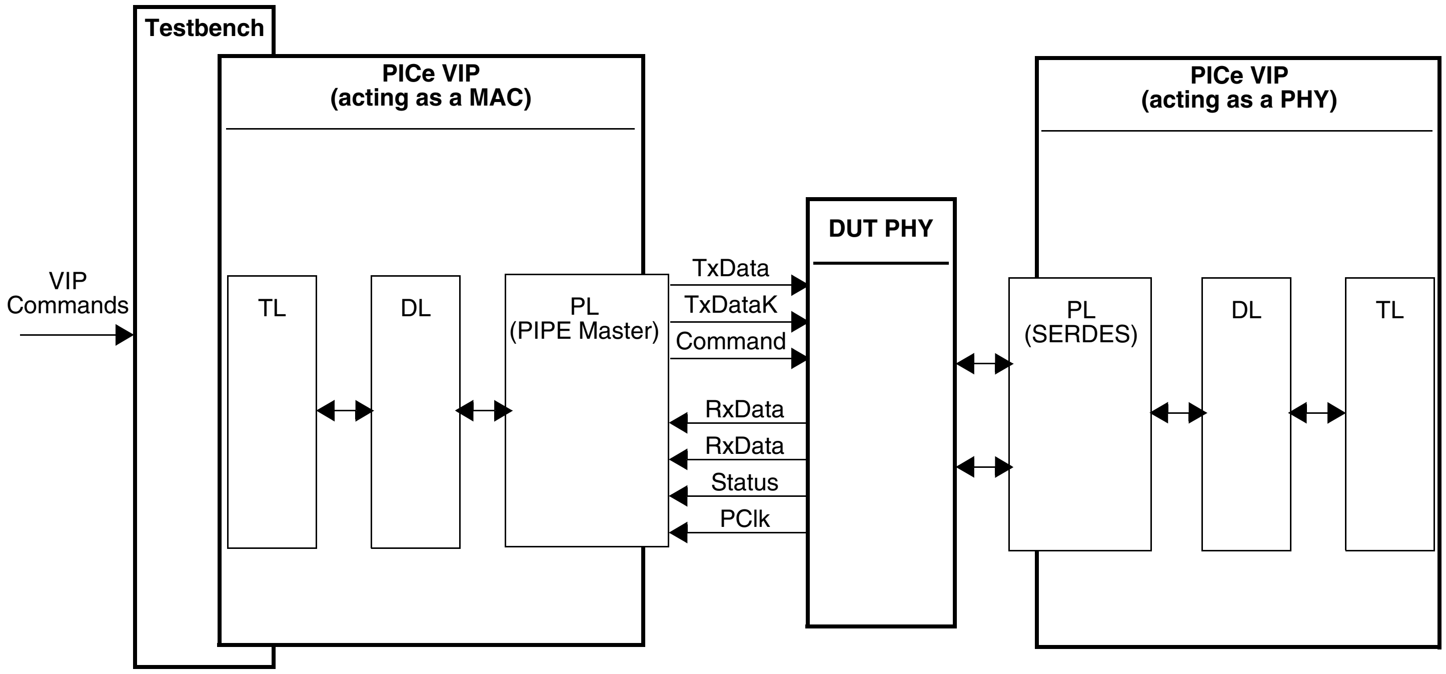


Table 6-4 pciesvc_model MPIPE ports

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	Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning. Reset must be deasserted at time 0 such that a posedge is seen by the VIP.
	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_*n*[31:0]	I	Per lane receive data into the VIP. Supports 8, 16, and 32bit bus widths
	rx_data_k_*n*[3:0]	I	Per lane control indication. Supports 8, 16, and 32 bit bus widths.
	rx_status_*n*[2:0]	I	Per lane receiver status. See PIPE spec for encodings. Note: Any unused ports must be tied to a 0.
	rx_valid_*n*	I	Per lane indication that data coming into the VIP is valid and symbol lock has been achieved.
	rx_data_valid_*n*	I	Per lane indication that data coming into the VIP is valid. Used primarily for rate matching. NOTE: PIPE3 only.

Table 6-4 pciesvc_model MPIPE ports (Continued)

Model Interface Type	Port Name	I/O	Description
	rx_elec_idle_*n*	I	Per lane indication that electrical idle is being received into the VIP Note: Any unused ports must be tied to a 1.
	invert_rx_polarity_*n*	O	Per lane indication to perform polarity inversion on data into the VIP
	tx_data_*n*[31:0]	O	Per lane transmit data out from the VIP
	tx _data_k_*n*[3:0]	O	Per lane transmit_control from the VIP
	tx _ei_code_*n*[31:0]	O	Per lane error injection code. Indicates to the phy that an EI was either injected, or phy needs to inject the EI indicated by the code. For VIP only. No connection to DUT.
	tx_compliance_*n*	O	Per lane transmit compliance pattern enable from the VIP
	tx_elect_idle_*n*	O	Per lane transmit electrical idle/transmit enable from the VIP
"Master" PIPE Version 4 Interface This type of interface is a standard PIPE interface to a phy. The 8G models require a PIPE4 interface, hence all models in the InstantionModels directory with the _8g suffix use the PIPE4.	reset	I	Active high reset. Must only be asserted for 100ns ONCE per simulation at the beginning.
	pipe_reset_*n*	O	PIPE reset signal to phy
	pipe_clk	I	PIPE clock from phy
	powerdown[2:0]	O	Command to change power state of the phy No vendor specific power levels are supported.
	rate[1:0]	O	Link signaling rate control
	pclk_rate[2:0]	O	Specifies the value of the PIPE clock.
	txdetectrx	O	Command to detect receiver
	block_align_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.
	width[1:0]	O	Outputs the width of the per-lane data bus the mac is currently using.

Table 6-4 pciesvc_model MPIPE ports (Continued)

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

Table 6-4 pciesvc_model MPIPE ports (Continued)

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

6.6 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.

6.6.1 Model Wire Interface Options

Table 6-2 defines the port list for each interface configuration of the pciesvc_model instantiation.

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

6.7 Configuring the PIPE Data Bus Width

6.7.1 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

6.7.2 PIPE4 and PIPE 4.2

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

  Options are (pclk_rate_enum) : PCLK_67_5_MHZ, PCLK_125_MHZ, PCLK_250_MHZ, PCLK_500_MHZ, PCLK_1000_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

  Supported widths are (pipe_width_enum): PIPE_8_BITS, PIPE_16_BITS, PIPE_32_BITS

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.

6.8 Compile-Time Parameter Settings

Several parameters are set at the model. They typically ‘trickle-down’ to the individual layers.

Table 6-5 lists the Verilog parameters that you set at compile time. They are not runtime changeable. They are not accessible from the VMM environment.

Table 6-5 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.v: 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.
ENABLE_SHADOW_ MEMORY_CHECKING	Integer	0-1	1	If set, applications will check memory reads against the shadow memory. Make sure this is not enabled if the shadow memory is not instantiated.
DISPLAY_NAME	String		"pciesvc_xx_yy_model_zz." (Specific to each model).	String prefixed to messages to display in the output log.

Example 6-1 Setting Verilog parameters at compile time

```
// This is a PIPE SLAVE model, so inputs and outputs are reversed from a normal PIPE
pciesvc_device_spipe_x4_model #(.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
   );
```

6.9 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:

```
pciesvc_device_spipe_x4_model #(.DISPLAY_NAME("root0."),
.DEVICE_IS_ROOT(1), .HIEARARCHY_NUMBER(1)) root0(.reset
```

...

6.10 Model Configuration Overview

Configuration tasks are used after reset is deasserted from the pciesvc_model.

Recommended tasks to be set prior to usage are described in Table 6-6. The details of the task may be found in the respective section of the documentation.

Table 6-6 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_scenario	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_scenario	Data Link Layer
svt_pcie_pl_configuration::set_link_speed_values()	Physical Layer
svt_pcie_pl_configuration::set_link_width_values()	Physical Layer

6.11 Turning Off of 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.

7 Using the VC PCIe Verification IP

7.1 SystemVerilog VMM Example Testbenches

This section describes SystemVerilog VMM example testbenches that show general usage for various applications. A summary of the examples is listed in Table 7-1

Table 7-1 SystemVerilog Example Summary

Example Name	Level	Description
tb_pcie_svt_vmm_intermediate_sys	Intermediate	The example consists of the following: • A top-level module, which includes tests, instantiates interfaces, HDL interconnect wrapper and generates system clock • A program block instantiates and creates PCIe System Env, includes tests and runs the tests • PCIe System Env is configured with one Root Complex and one Endpoint • Shows how to reconfigure the PCIe link for speed and lane width • Coverage generation • Functional coverage is enabled • Protocol Analyzer is enabled • Symbol and transaction log are enabled

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 intermediate example are:

- ts.base_8g_pipe_test.sv
- ts.base_8g_pma_test.sv
- ts.base_8g_serdes_test.sv
- ts.base_pipe_test.sv
- ts.base_pma_test.sv
- ts.base_serdes_test.sv
- ts.directed_pipe_test.sv
- ts.directed_pma_test.sv

- ts.directed_serdes_test.sv
- ts.driver_exceptions_inline_pipe_test.sv
- ts.driver_exceptions_inline_pma_test.sv
- ts.driver_exceptions_inline_serdes_test.sv
- ts.reconfigure_link_pipe_test.sv
- ts.reconfigure_link_pma_test.sv
- ts.reconfigure_link_serdes_test.sv
- ts.symbol_exception_via_callback_pipe_test.sv
- ts.tlp_exception_via_callback_pipe_test.sv

7.2 Installing and Running the Examples

Below are the steps for installing and running example tb_pcie_svt_vmm_intermediate_sys. Similar steps are applicable for other examples:

1. Install the example using the following command line:

   ```
   // Location where the example is to be installed.
   % cd <location>

   // Provide any name of your choice in place of "design_dir"
   % mkdir design_dir

   % $DESIGNWARE_HOME/bin/dw_vip_setup -path ./design_dir  -e
     pcie_svt/tb_pcie_svt_vmm_intermediate_sys -svtb
   ```

   This installs the example under:

   ```
   <design_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_vmm_intermediate_sys
   ```

2. Use either one of the following to run the testbench:

   a. Use the Makefile. Use any of the names documented in the previous section. For example: ts.directed_pipe_test.sv.

      To run the ts.directed_pipe_test.sv for example, do following:

      ```
      gmake USE_SIMULATOR=vcsvlog directed_pipe_test WAVES=1
      ```

      To see more options, invoke "gmake help".

   b. Use the sim script: To run the directed_pipe_test.sv, for example, do following:

      ```
      ./run_pcie_svt_vmm_intermediate_sys -w directed_pipe_test vcsvlog
      ```

      To see more options, invoke "./run_pcie_svt_vmm_intermediate_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_vmm_intermediate_sys/README

or

<design_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_vmm_intermediate_sys/README

7.2.1 Modifications to Run the Intermediate Example in Gen3 Using x8 with the PIPE Interface

Use the following procedure to change the intermediate example files to use x8 and Gen3.

1. Go to<design_dir>/src/verilog/vcs directory.
2. Choose the desired model, for example: pciesvc_device_s(m)pipe_x8_model_8g.v for x8 8G.
3. Go to the <design_dir>/examples/sverilog/pcie_svt/tb_pcie_svt_vmm_intermediate_sys/hdl_interconnect directory.
4. Create a new file svt_pcie_8g_device_pipe_x8.sv and instantiate the MPIPE/SPIPE Verilog modules from step 2 and make connections by taking a reference of the existing x4 pipe connection.

5. Create a new top file top.pcie_8g_device_pipe_x8.sv which will have no change but the `include of the above svt_pcie_8g_device_pipe_x8.sv.
6. Modify the prescript in order to make top.svt_pcie_device_pipe_x8.sv as top_test.sv.

   Alternatively, instead of modifying the prescript, you can overwrite the svt_pcie_8g_device_pipe_x4.sv with instantiation of pciesvc_device_s(m)pipe_x8_model_8g.v and make the requisite connections
7. Modify the configuration attributes for the required configuration.

   For example, the link_width needs to be changed when going from x4 to x8 in the following file:

      env/pcie_shared_cfg.sv

   In the file, change the argument from 4 to 8 for the set_link_width_values() task:

   ```
   cust_cfg.root_cfg.pcie_cfg.pl_cfg.set_link_width_values(8);
   ```

7.3 Resetting the VC PCIe VIP

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 for at least 100 ns. Once deasserted, it should remain deasserted 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 deasserted. 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

7.4 Creating and Using Custom Applications

Custom applications and test scenarios can be developed for the VC for 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_ms_scenario_gen TLP generator using a TLP scenario that generates TLPs from svt_pcie_group::tlp_ms_scenario_gen.

User applications co-exist and run in parallel to the Synopsys-supplied applications. Alternatively, testbenches can emulate user applications using scenarios that generate TLPs with unique application IDs. The scenarios that generate these TLPs run on the svt_pcie_group::tlp_ms_scenario_gen generator.

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.

7.4.1 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 scenario of this type will run on the svc_ms_scenario_gen of the device group, as shown in Example 7-1.

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 7-1

```
svt_pcie_tl_service add_mem_add_req;
bit res;
add_mem_add_req = svt_pcie_pl_service::create_instance(this, "blueprint",
`__FILE__, `__LINE__);
    add_mem_add_req.cfg = pl_cfg;

res = add_mem_add_req.randomize() with {
                   service_type == svt_pcie_tl_service::ADD_MEM_ADDR_APPL_ID_MAP_ENTRY;
                   appl_id == 32'h21;
                   memory_addr == 0;
                   memory_window == 32'h1000;};
out_chan.put(add_mem_add_req);
```

7.4.2 Using Testbench Scenarios to Emulate User Applications

Testbenches can add scenarios that generate TLPs with unique application IDs to emulate user applications. A scenario is created that generates TLPs with unique application IDs and is run on the tlp_ms_scenario_gen of the VC for PCIe group contained in the PCI device group.

```
env.root.tlp_ms_scenario_gen.register_ms_scenario("tlp_directed_scenario",
tlp_directed_scenario);
```

The output channel of this ms_scenario generator is internally connected to the input channel(tlp_chan) of the VIP's transaction layer.

The TLPs are set up with the appropriate application ID and requester ID and pushed into the input channel, as indicated in Example 7-2.

Example 7-2

```
svt_pcie_tlp mem_rd_request;
bit res;

mem_rd_request = svt_pcie_driver_app_transaction::create_instance(this, "blueprint",
`__FILE__, `__LINE__);

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;
out_chan.put(mem_rd_request);
```

7.4.3 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:

```
out_chan.put(mem_rd_request);
root.tl.EVENT_RECEIVED_TLP.wait_trigger();
root.tl.rx_tlp_peek_chan.get(resp);
```

7.5 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 VC for PCIeVC for VIP.

Each device allocates a Configuration Pointer Table which contains pointers to all of the Configuration Blocks allocated (one for each function in the device). Tasks that provide access to the completion target configuration space access are listed in

7.6 Setting up the Configuration Space for Backdoor Access

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.

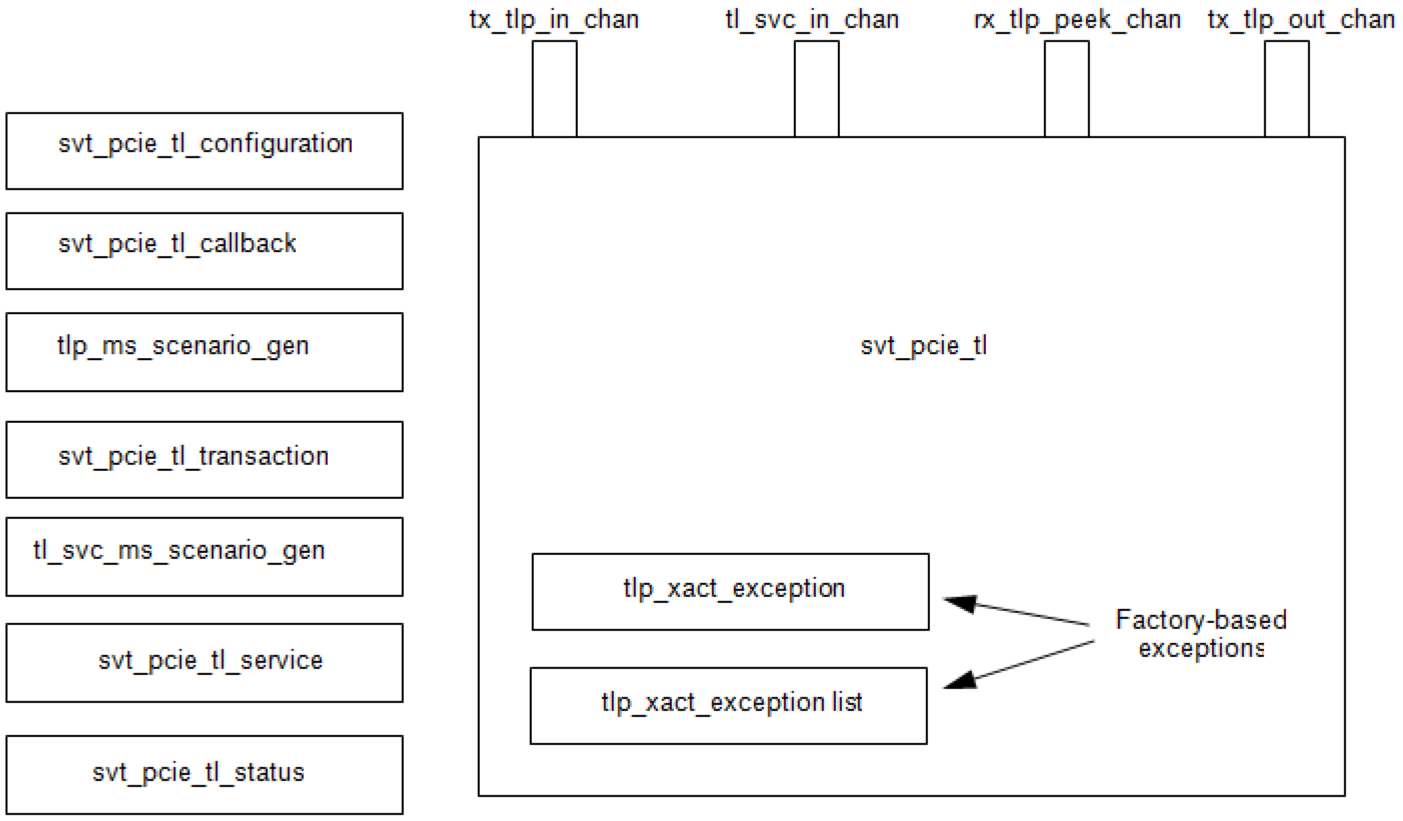
8 Using the Transaction Layer

8.1 Transaction Layer

The Transaction Layer, TL, is implemented as a vmm_xactor, svt_pcie_tl. The TL contains a configuration object, svt_pcie_tl_configuration (see "Transaction Layer Configuration"), a service ms-scenario generator (see "Transaction Layer Generators and Sscenarios"), 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 Exceptions"), as well as a status object "Transaction Layer Status".

A block diagram of the Transaction Layer elements is shown in Figure 8-1.

Figure 8-1 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.

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 scenario to set VC_ENABLE to set up the VCs.

```
task vc_enable_scenario::execute(ref int n);
svt_pcie_tl_service set_vc_en_req;
bit res;
res = set_vc_en_req.randomize() with {
                              service_type == svt_pcie_tl_service::SET_VC_ENABLE;
                              vc_num == 4;
                              vc_enable == 1;};
endtask
env.root.tl_svc_ms_scenario_gen.register_ms_scenario("vc_enable_scenario",
vc_enable_scenario);
```

8.2 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_device_configuration class are listed in Table 8-1.

Table 8-1 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.

Table 8-1 Transaction Layer configuration members (Continued)

svt_pcie_tl_configuration::default_route_mem_appl_id Type: rand int unsigned Default:0 Description: Default application ID to use.
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.

Table 8-1 Transaction Layer configuration members (Continued)

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.
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.

8.2.1 Verilog Configuration Parameters

Not all configuration items are currently controlled from within the VMM interface. At this time the items in Compile-time Verilog Parameters and Runtime-changeable Verilog Parameters are controlled only via Verilog.

8.2.1.1 Compile-time Verilog Parameters

Parameters that are only changeable when instantiating the TL as part of the instantiation model are listed in Table 8-2.

Table 8-2 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.

8.2.1.2 Runtime-changeable Verilog Parameters

Transaction Layer parameters that are changeable at runtime are listed in Table 8-3.

Table 8-3 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.

8.3 Transaction Layer Multi-Stream Scenario Generators and Scenarios

The transaction layer supports both service and transaction scenarios. All TL service scenarios run on the svt_pcie_group::tl_svc_ms_scenario_gen. The TL Service generator is accessed through one of the following paths:

Generator:

instance name of svt_pcie_group.tl_svc_ms_scenario_gen

The driver app service generator control stop_after_n_insts and stop_after_n_scenarios can be set through svt_pcie_configuration class members

```
cust_cfg.root_cfg. pcie_cfg.stop_after_n_tl_service_insts = 6;
cust_cfg.root_cfg. pcie_cfg.stop_after_n_tl_service_scenarios = 1;
```

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 called svt_pcie_tl_service using which service scenarios can be built.

The service, svt_pcie_tl_service, is a vmm_data type class 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 scenarios that contain the base svt_pcie_tl_service that can be used for test development.

svt_pcie_tl_check_final_credits_scenario
fc_scenario;env.root.tl_svc_ms_scenario_gen.register_ms_scenario("fc_scenario",fc_scenario);

Transaction Layer service scenario examples are listed in the following table.

Transaction Layer provides the svt_pcie_tl_service, which is a vmm_data type that can be used to build custom scenarios. Enumerated values of the service type and their associated attributes are listed in the following table.

Table 8-4 Transaction Layer Service Scenarios

TL Service Final Credits Scenario
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.
 	
class pcie_tl_service_check_final_credits_scenario extends vmm_ms_scenario;
virtual task execute(ref int n);
svt_pcie_tl_service check_final_credits_req;
…
check_final_credits_req.randomize() with {service_type == svt_pcie_tl_service::CHECK_FINAL_CREDITS;};
…
endtask
endclass
pcie_tl_service_check_final_credits_scenario fin_crdt_scenario;
env.root.tl_svc_ms_scenario_gen.register_ms_scenario("fin_crdt_scenario",fin_crdt_scenario);

TL Service clear Stats Scenario
Description:
Clears all stats in Transaction Layer
Description:
Displays all stats in Transaction Layer.
 	
execute(ref int n);
svt_pcie_tl_service clear_stats_req;
…
clear_stats_req.randomize() with {service_type == svt_pcie_tl_service::CLEAR_STATS;};
…
endtask
 	
tl_service_clr_stats_scenario clr_stats_scenario;
env.root.tl_svc_ms_scenario_gen.register_ms_scenario("clr_stats_scenario ", clr_stats_scenario);

Table 8-4 Transaction Layer Service Scenarios (Continued)

```
TL service set_vc_en scenario
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.

task vc_enable_scenario::execute(ref int n);
svt_pcie_tl_service set_vc_en_req;
bit res;
res = set_vc_en_req.randomize() with {
                    service_type == svt_pcie_tl_service::SET_VC_ENABLE;
                    vc_num == 4;
                    vc_enable == 1;};
endtask

env.root.tl_svc_ms_scenario_gen.register_ms_scenario("vc_enable_scenario",  vc_enable_scenario);
```

Table 8-5 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.

Table 8-5 Service type enumerated parameters (Continued)

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.
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.v 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.			

Table 8-5 Service type enumerated parameters (Continued)

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.			
	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.

8.4 Transaction Layer Callbacks and Exceptions

The Transaction Layer provides a callback class, svt_pcie_tl_callback.   It is available for observation or data modification via the use of exceptions.

```
class svt_pcie_tl_callback extends svt_xactor_callback;
   extern function void new ();
   extern virtual function void pre_tlp_chan_put ( svt_pcie_tl tl , svt_pcie_tlp tlp ,
      ref bit drop );
   endfunction;
   virtual function void post_chan_get(svt_pcie_tl tl, svt_pcie_tlp tlp,
      ref bit drop);
   endfunction
endclass
```

The pre_tlp_chan_put method is on the RX side only. It is called by the TL whenever the TL receives a transaction from the DL, but before it puts the transaction onto its RX channel. No processing is done to the transaction, although the callback implementation could be used to do this, after the TLP is received completely and prior to putting the received TLP on the rx_tlp_peek_chan.

The post_chan_get() function is on the TX side only.   It's issued by the TL once the TLP is pulled from its TLP input, but before acting on the TLP in any way.

8.4.1 Transaction Layer Exceptions

Exceptions can be applied using the svt_pcie_tl_callback. The svt_pcie_tlp class contains an exception list, svt_pcie_tlp_exception_list. By default the 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.

Reviewing the svt_pcie_tlp_exception_list, you can use the svt_pcie_tlp_exception class to set up a particular exception.

Refer to these classes in the HTML reference to see available exception types.

8.5 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_group.status.tl_status

Refer to the HTML Reference for a full listing of status members.

8.5.1 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 8-1

```
wait(env.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 8-5.

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 8-1.

8.6 Transaction Layer Channels

rx_tlp_out_chan and rx_tlp_peek_chan in svt_pcie_tl transactor. You can connect to those channels for access to all received TLPs.

Additionally, there is a tl_svc_in_chan and tx_tlp_in_chan that is used for service requests and user applications.

8.7 Transaction Layer Verilog Interface

The Verilog 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.

8.7.1 Transaction Layer Module IOs

The Transaction Layer module I/O signals listed in <Xref>Table 8-6 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 8-6 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

9 Using the Driver Application

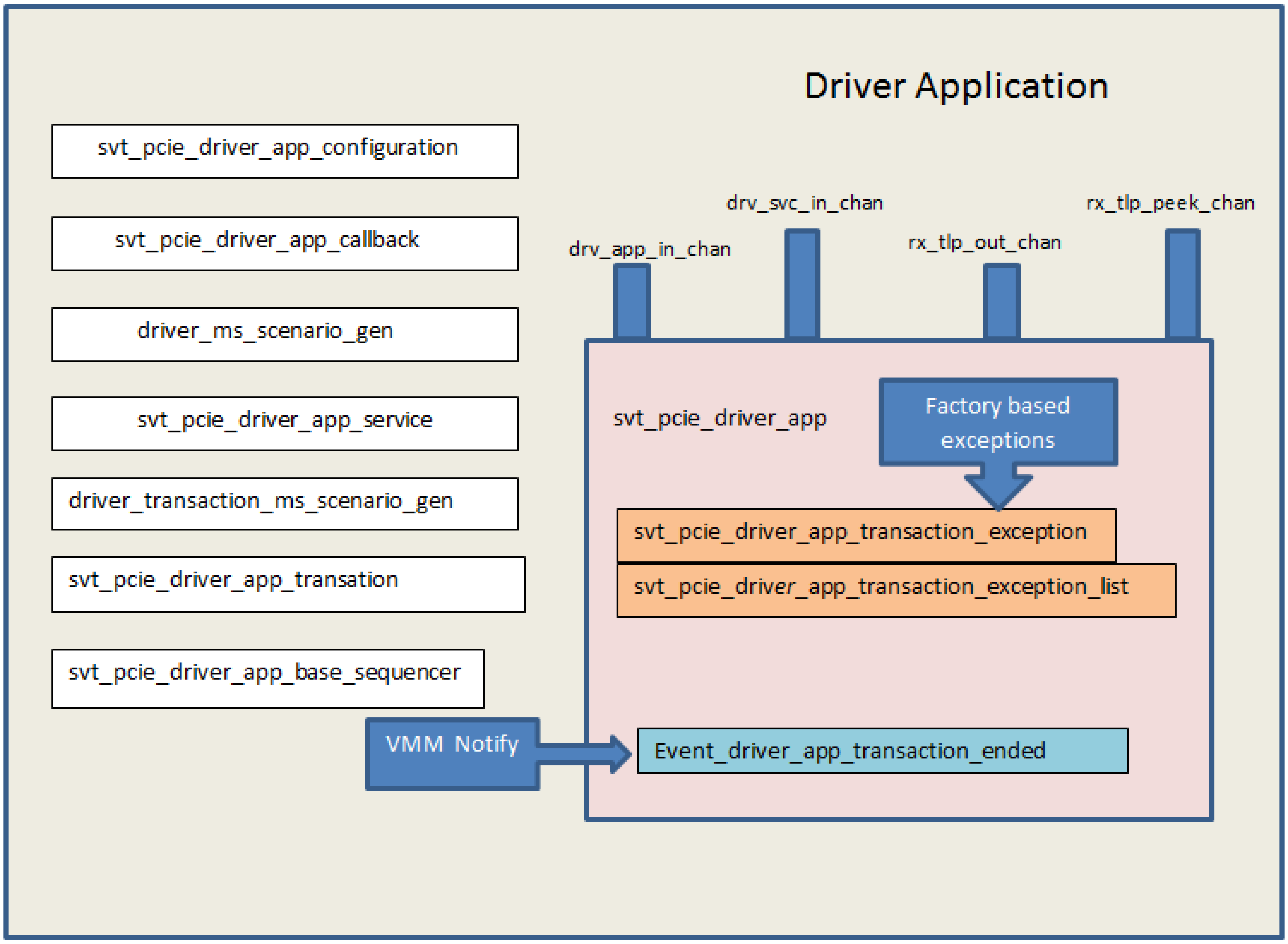
9.1 Introduction

The Driver application is implemented as a vmm_xactor transactor . The Driver has the following functions:

- ❖ Provides configuration and service scenarios and transaction scenarios for the creation of PCIe transactions
- ❖ Tracks completions for a given transaction
- ❖ Interfaces with the Global Shadow and built-in scoreboard to validate data in the PCIe bus

The driver consists of several s, as shown in Figure 9-1.

Figure 9-1 PCIe Driver Applications


Driver Application
svt_pcie_driver_app_configuration
svt_pcie_driver_app_callback
driver_ms_scenario_gen
svt_pcie_driver_app_service
driver_transaction_ms_scenario_gen
svt_pcie_driver_app_transation
svt_pcie_driver_app_base_sequencer
VMM Notify
drv_svc_in_chan
rx_tlp_peek_chan
drv_app_in_chan
rx_tlp_out_chan
svt_pcie_driver_app
Factory based exceptions
svt_pcie_driver_app_transaction_exception
svt_pcie_driver_app_transaction_exception_list
Event_driver_app_transaction_ended

9.2 Driver Application Configuration

The Driver application configuration class is svt_pcie_driver_app_configuration. The members within the class control the settings of the Driver.

The Driver has a few parameters that are only configurable in Verilog instantiation models. See <Xref>“Verilog Configuration Parameters and Tasks” for information about those parameters.

The svt_pcie_driver_app_configuration class is accessed via the following instance hierarchy:

instance-name-of-svt_pcie_device_configuration.pcie_cfg.driver_cfg[*int*]

Table 9-1 shows the svt_pcie_driver_app_configuration items.

Table 9-1 Driver application configuration members

svt_pcie_driver_app_configuration::application_number Type: rand int unsigned Default: 1 Description: Application number for the Driver application.
svt_pcie_driver_app_configuration::completion_timeout_ns Type: rand int unsigned Default: 50,000 Description: Completion timeout value. The request times out after the time as specified by this variable if completions are not received for the request.
svt_pcie_driver_app_configuration::display_outstanding_commands_period Type: rand int unsigned Default: 50000 Description: The display_outstanding_commands_period variable specifies the time after which the Driver application displays the outstanding/pending transactions.
svt_pcie_driver_app_configuration::enable_shadow_memory_checking Type: rand bit Default: 1 Description: When set to 1, read transactions are checked against the data from Global Shadow application. When set to 0, read transactions are not checked against the data from Global Shadow application.
svt_pcie_driver_app_configuration::enable_tx_tlp_reporting Type: rand bit Default: 0 Description: When set to 1, transmitted TLPs are reported to Global Shadow application. When set to 0, transmitted TLPs are not reported to Global Shadow application.
svt_pcie_driver_app_configuration::max_time_to_next_transition Type: rand int unsigned Default: 0 Description: Maximum time between transactions before the driver will transmit another queued request.
svt_pcie_driver_app_configuration::min_time_to_next_transition Type: rand int unsigned Default: 0 Description: Minimum time between transactions before the driver will transmit another queued request.

Table 9-1 Driver application configuration members (Continued)

svt_pcie_driver_app_configuration::model_instance_scope Type: string Description: The full hierarchical path name to the instance of the model in which the driver model is instantiated. The path name is concatenated with the name of this tranactor passed to the constructor to generate the lookup string used to find the SV API instance. The model_instance_scope value should not be changed at this level: it is set at the group level and propagates to all sub-levels.
svt_pcie_driver_app_configuration::percentage_use_tlp_digest Type: rand int unsigned Default: 0 Description: Percentage probability of the TD bit being set, indicating that the packet has a TLP digest.
svt_pcie_driver_app_configuration::read_completion _boundary_in_bytes Type: rand int unsigned Default: 64 Description: The variable read_completion_boundary_in_bytes specifies the RCB value. The Driver application checks all the received completions against this boundary.
svt_pcie_driver_app_configuration::requester_id Type: rand bit [15:0] Default: 32h‘000_0001 Description: Requester ID
svt_pcie_driver_app_configuration::set_ido_field Type: rand bit Default: 0 Description: Sets the IDO bit in all outgoing TLPs when applicable.

Table 9-1 Driver application configuration members (Continued)

svt_pcie_driver_app_configuration::set_no_snoop_field Type: rand bit Default: 0 Description: Sets the No Snoop bit in all outgoing TLPs when applicable.
svt_pcie_driver_app_configuration::set_relaxed_ordering_field Type: rand bit Default: 0 Description: Sets the Relaxed Ordering bit in all outgoing TLPs when applicable.
svt_pcie_driver_app_configuration::use_internal_data_buffers Type: rand bit Default: 0 Description: When set to 1, internal data buffers are used for the transactions. When set to 0, data buffers should be provided by user.

9.3 Verilog Configuration Parameters and Tasks

The items in "Compile-time Verilog Parameters" and "Runtime-changeable Verilog Parameters" are controlled only via Verilog.

9.3.1 Compile-time Verilog Configuration Parameters

Parameters that are only changeable when instantiating the Driver application as part of the instantiation model are listed in Table 9-2.

Table 9-2 Driver application compile Verilog parameters

Parameter Name	Type	Range	Default Value	Description
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. This is not typically changed by the user.
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.

9.3.2 Runtime Configuration Parameters

Driver application parameters that are changeable at runtime are listed in Table 9-3.

Table 9-3 Driver application runtime Verilog parameters

Parameter Name	Type	Range	Default Value	Description
PCIE_SPEC_VER				
	real	1.1, 2.0, 2.1, 3.0	PCIE_SPEC_VER_2_1	See Include/pciesvc_parms.v: PCIE_SPEC_VER_* Note: Set this parameter in the model. It is not changed at this level but rather at the group level

9.3.3 Runtime Verilog Tasks

Verilog tasks that are changeable at runtime are listed in Table 9-4.

Table 9-4 Driver application runtime Verilog tasks

Parameter Name	Arguments	I/O	Description
AddTLPPrefix Adds the prefix or prefixes contained in prefix_array to the next queued command.	logic [31:0] prefix_array[]	I	A dynamic array containing the prefixes to be added. It is up to the user to build and set the contents of the array.

9.4 Driver Application MS-Scenario Generator and Scenarios

The Driver layer supports both service and transaction scenarios. Service scenarios run on the driver_svc_ms_scenario_gen while transaction scenarios run on the driver_transaction_ms_scenario_gen.

Test writers can access the service ms-scenario generator via the following path:

instance-name-of-svt_pcie_device_group.driver_svc_ms_scenario_gen[int]

Test writers can access the transaction generator via the following path:

instance-name-of-svt_pcie_device_group.driver_transaction_ms_scenario_gen[int]

By default, there is one instance of the driver service generator driver_svc_ms_scenario_gen[0] and one instance of the driver transaction generator driver_transaction_ms_scenario_gen[0]. Implementing them as arrays allows for future expansion.

9.4.1 Service Scenarios

Services are commands to give the model that are related to behavior or configuration; they are not a transaction item which is to be sent across the bus. For the Driver there is a base service, svt_pcie_driver_app_service that can be used to implement service base scenarios.

The service, svt_pcie_driver_app_service, is a vmm_data type class which supports all the service transaction types supported by the Driver. A service is selected by constraining the service_type_enum of the class to one of the enumerated service types.

9.4.1.1 Driver Application Service Generator

The svt_device_pcie_group::driver_svc_ms_scenario_gen[0] is an object of multi-stream scenario generator that provides stimulus for the svt_pcie_driver_app_service_driver class. The svt_pcie_device_group class is responsible for connecting this generator to the driver if the device is configured as ACTIVE.

The driver app service generator control stop_after_n_insts and stop_after_n_scenarios can be set through svt_pcie_device_configuration class members

```
cust_cfg.root_cfg.stop_after_n_driver_insts = 2;
cust_cfg.root_cfg.stop_after_n_driver_scenarios = 1
```

9.4.1.2 Driver Application Service Scenario Examples

The following subsections show various Service Scenario examples.

9.4.1.2.1 IS_TRANSACTION_COMPLETE Example

This scenario implements Is Transaction Complete. IS_TRANSACTION_COMPLETE creates a request to check if a transaction is complete.

The scenario is useful to query the VIP about whether the Driver transaction that is queued to the Driver application transactor is complete. If the transaction with the specified command_num is complete, returned

```
status is 1'b1, otherwise the returned status is 1'b0.
task driver_app_service_is_trans_comp_scenario::execute(ref int n);
svt_pcie_driver_app_service is_trans_compl_req;
……
res= is_trans_compl_req.randomize() with {
service_type == svt_pcie_driver_app_service:: IS_TRANSACTION_COMPLETE;
command_number == 10};
out_chan.put(is_trans_compl_req); n++;
……
endtask
```

9.4.1.2.2 Driver App Service wait_for_compl_scenario Example

This scenario implements Wait For Completion. WAIT_FOR_COMPLETION creates a request to wait for completion for the specified transaction.

The command_num variable is the ID for the transaction which is available once the transaction is queued to the Driver. The scenario blocks until all the completion data for the specified request is returned.

```
task driver_app_service_wait_for_compl_scenario::execute(ref int n);
svt_pcie_driver_app_service wait_for_compl_req;
……
res= wait_until_idle_req.randomize() with {
service_type == svt_pcie_driver_app_service::WAIT_FOR_COMPLETION;
command_number == 10};
```

```
out_chan.put(wait_for_compl_req ); n++;
……
endtask
```

9.4.1.2.3 Driver App Service wait_until_idle_scenario Example

This scenario implements Wait Until Driver Idle. WAIT_UNTIL_DRIVER_IDLE creates a request to wait until the Driver application is idle.

When the Driver has some queued transactions to be transferred over the link, or if the Driver has not yet received any completion of transferred transactions (that is, if transaction are outstanding), then the Driver is said to be in a non-idle condition. When all of the outstanding transactions are complete and there are no transactions queued to the Driver, the Driver is said to be in an idle condition.

This scenario blocks until the Driver is idle.

```
task driver_app_service_wait_for_idle_scenario::execute(ref int n);
svt_pcie_driver_app_service wait_until_idle_req;
……
res= wait_until_idle_req.randomize() with {service_type ==
svt_pcie_driver_app_service::WAIT_UNTIL_DRIVER_IDLE;};
out_chan.put(wait_until_idle_req); n++;
……
endtask
```

9.4.2 Transaction Scenarios

Alternatively, the model provides several scenarios that contain the base svt_pcie_driver_app_transaction that can be used for test development. The transaction generator is described in Table 9-6.

The transaction scenario examples are listed in Table 9-6.

Transaction scenarios for a given type of transaction use the fields shown in Table 9-5.

Table 9-5 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_device_group::driver_transaction_ms_scenario_gen This is an object of multi-stream scenario generator that provides stimulus for the svt_pcie_driver_app_transaction_driver class. The svt_pcie_devvice_group class is responsible for connecting this generator to the Driver if the devvice is configured as ACTIVE. The driver app transaction generator control stop_after_n_insts and stop_after_n_scenarios can be set through svt_pcie_device_configuration class members:

```
cust_cfg.root_cfg.stop_after_n_driver_trans_insts = 2;
cust_cfg.root_cfg.stop_after_n_driver_trans_scenarios =1
```

Table 9-6 Driver application transaction scenario examples

Driver app transaction fetchadd_scenario
This scenario generates an atomic operation fetchadd scenario

```
task fetchadd_scenario::execute(ref int n);
……
svt_pcie_driver_app_transaction tr;
out_chan = get_channel("driver_transaction_chan0");
res = tr.randomize() with {transaction_type == svt_pcie_driver_app_transaction:: ATOMIC_OP_FETCH_ADD;
                            address inside {[32'h4000_0000 : 32'h8000_0000]};
                            first_dw_be == 4'b1110;
                            last_dw_be == 4'b1111;
                            length inside {[16:32] };
                            traffic_class == 0;
                            address_translation == 0;
                            ep == 0;
                            block == 1;
                            };
      if (!res)
        `vmm_error(log, "Randomization failed");
      else
        out_chan.put(tr); n++;
….
endtask.
```

Driver App Transaction atomicop_swap_scenario
This scenario generates an atomic operation swap request.

```
task atomicop_swap_scenario::execute(ref int n);
……
svt_pcie_driver_app_transaction tr;
out_chan = get_channel("driver_transaction_chan0");
res = tr.randomize() with {transaction_type == svt_pcie_driver_app_transaction:: ATOMIC_OP_SWAP;
                            address inside {[32'h4000_0000 : 32'h8000_0000]};
                            length inside {[16:32] };
                            first_dw_be == 4'b1111;
                            last_dw_be == 4'b1111;
                            traffic_class == 0;
                            address_translation == 0;
                            ep == 0;
                            block == 0;
                            };
      if (!res)
        `vmm_error(log, "Randomization failed");
      else
        out_chan.put(tr); n++;
….
endtask
```

Table 9-6 Driver application transaction scenario examples (Continued)

Driver App Transaction cfg_rd_scenario

This scenario generates a configuration read request.

```
task cfg_rd_scenario::execute(ref int n);
……
svt_pcie_driver_app_transaction tr;
bit res;
rand bit [2:0] func_num = $random;
out_chan = get_channel("driver_transaction_chan0");
res = tr.randomize() with {transaction_type == svt_pcie_driver_app_transaction:: CFG_RD;
                           address == {8'h01, 5'b00000, func_num};
                           register_number == 7;;
                           cfg_type == 0;
                           first_dw_be == 4'hF;
                           block == 0;
                           };
    if (!res)
     `vmm_error(log, "Randomization failed for pcie_device_random_traffic_scenario MEM_WR Scenario");
    else
     out_chan.put(tr); n++;
….
endtask
```

Table 9-6 Driver application transaction scenario examples (Continued)

Driver App Transaction io_wr_scenario

This scenario generates an I/O write request.

```
task io_wr_scenario::execute(ref int n);
……
svt_pcie_driver_app_transaction tr;
out_chan = get_channel("driver_transaction_chan0");
res = tr.randomize() with {transaction_type == svt_pcie_driver_app_transaction::IO_WR;
                           address inside {[32'h4000_0000 : 32'h8000_0000]};
                           first_dw_be == 4'b1111;
                           ep == 0;
                           block == 0;
                           };
    if (!res)
     `vmm_error(log, "Randomization failed for pcie_device_random_traffic_scenario MEM_WR Scenario");
    else
     out_chan.put(tr); n++;
….
endtask
```

Driver App Transaction mem_wr_scenario

This scenario generates a memory write request.

```
task mem_wr_scenario::execute(ref int n);
……
svt_pcie_driver_app_transaction tr;
out_chan = get_channel("driver_transaction_chan0");
res = tr.randomize() with {transaction_type == svt_pcie_driver_app_transaction::MEM_WR;
                           address inside {[32'h4000_0000 : 32'h8000_0000]};
                           length inside {[16:32] };
                           traffic_class == 0;
                           address_translation == 0;
                           ep == 0;
                           block == 0;
                           };
    if (!res)
     `vmm_error(log, "Randomization failed for pcie_device_random_traffic_scenario MEM_WR Scenario");
    else
     out_chan.put(tr); n++;
….
endtask
```

Table 9-6 Driver application transaction scenario examples (Continued)

```
Driver App Transaction vendor_msg_0_scenario
This scenario generates a vendor-defined message type 0 request.
task vendor_msg_0_scenario::execute(ref int n);
……
svt_pcie_driver_app_transaction tr;
bit res;
rand bit[2:0] local_traffic_class;
out_chan = get_channel("driver_transaction_chan0");
res = tr.randomize() with { {transaction_type == svt_pcie_driver_app_transaction::MSG;
                        vendor_fields == 16'h0;
                        length == 0;
                        routing_type == svt_pcie_driver_app_transaction::ROUTE_BY_ID;
                        traffic_class == local_traffic_class;
                        ep == 1'b0;
                        message_code == svt_pcie_driver_app_transaction::VENDOR_DEFINED_0; };
        if (!res)
          `vmm_error(log, "Randomization failed for pcie_device_random_traffic_scenario MEM_WR Scenario");
        else
          out_chan.put(tr); n++;
….
endtask
```

9.5 Driver Application Callbacks and Exceptions

The Driver provides a callback class, svt_pcie_driver_app_callback. It is available for observation of transactions only. Modification of the packet at this level would have no meaning.

```
class svt_pcie_driver_app_callback extends svt_callback;
   function void new ()
   virtual function void transaction_ended ( svt_pcie_driver_app driver,
      svt_pcie_driver_app_transaction transaction )
endclass
```

The transaction_ended method is called at the conclusion of a transaction. It is called by the Driver whenever the transaction is completed by the link partner. Completion data returned by the link partner is available in the payload. The transaction_ended method is not called for partial completions.

9.5.1 Transaction Layer Exceptions

Driver level exceptions are applied via the transaction item, itself or via the factory rather than via callbacks. The svt_pcie_driver_app_transaction class contains an exception list, svt_pcie_driver_app_exception_list. By default the 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.

Reviewing the svt_pcie_driver_app_exception_list, once can setup a particular exception via the svt_pcie_driver_app_exception class.

9.6 Driver Status

The Driver does not provide status.

9.6.1 Determining if the Driver Application is Idle

The Driver provides a service to determining if the Driver is idle, i.e. all queues are empty and all transactions have been completed. This is useful for determining end-of-test.

```
Example:
driver_app_service_wait_until_idle_scenario  wait_scenario = new(…);
…
env.root.driver_svc_ms_scenario_gen[0].register_ms_scenario("wait_scenario ",
wait_scenario);…
```

9.7 Driver Application Events

The driver provides a notification, EVENT_DRIVER_APP_TRANSACTION_ENDED which is triggered at the completion of each transaction. It can be used for synchronization of activities with the driver.

9.8 Driver Application Channels

A channel named driver_svc_chan[int] is used for driver tansactions and driver_transaction_chan[int] is used for service requests.

10 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.

10.1 Enabling Functional Coverage

To enable function coverage, set the following variables in the svt_pcie_configuration class:

```
enable_cov = 4'b1111; // 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 Layer respectively.

10.2 Class Structure and Callbacks

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

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

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

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

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

10.3 Overriding the Default Coverage Class

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

10.3.1 Overriding With VMM

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

Table 10-1 Transaction override functions

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

10.4 Transaction Layer

All methods and variables are declared in pciesvc_tl_fc_base, which is 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.

10.4.1 Transaction Layer Functional Coverage

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

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

Covergroup	Coverpoints	Bins

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

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

10.4.2 Transaction Layer Callbacks

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

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

Task Name	Arguments	I/O	Values
UpdateTxTcVcMapping Called every time the Transaction Layer passes a packet to the link.	tx_tc	I	Traffic class of the transmitted TLP
	tx_vc	I	VC which maps to the traffic class of the transmitted TLP
UpdateRxTcVcMapping Called every time the 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

10.5 Data Link Layer

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

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

10.5.1 Data Link Layer Functional Coverage

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

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

Covergroup	Coverpoint	Bins	Comment

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

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	
	cp_dllp_type_fc_vc1	initfc1_p_vc1	VC1 Flow Control DLLPs
		initfc1_np_vc1	
		initfc1_cpl_vc1	
		initfc2_p_vc1	
		initfc2_np_vc1	
		initfc2_cpl_vc1	
		updatefc_p_vc1	
		updatefc_np_vc1	
		updatefc_cpl_vc1	
	cp_dllp_type_fc_vc2	initfc1_p_vc2	VC2 Flow Control DLLPs
		initfc1_np_vc2	
		initfc1_cpl_vc2	
		initfc2_p_vc2	
		initfc2_np_vc2	
		initfc2_cpl_vc2	
		updatefc_p_vc2	
		updatefc_np_vc2	
		updatefc_cpl_vc2	

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

	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 10-4 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 10-4 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 10-4 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 10-4 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 10-4 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 10-4 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 10-4 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 10-4 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	

10.5.2 Link Layer Callbacks

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

Table 10-5 Data Link Layer callbacks

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

Table 10-5 Data Link Layer callbacks (Continued)

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

Table 10-5 Data Link Layer callbacks (Continued)

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

Table 10-5 Data Link Layer callbacks (Continued)

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

Table 10-5 Data Link Layer callbacks (Continued)

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

10.6 Physical Layer

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

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

10.6.1 Physical Layer Functional Coverage

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

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

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

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

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_c onfiguration_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 10-6 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 10-6 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 10-6 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			

10.6.2 Physical Layer Callbacks

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

Table 10-7 Callbacks in the Physical Layer

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

10.7 PIPE Interface

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

10.7.1 PIPE Functional Coverage

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

Table 10-8 PIPE covergroups, coverpoints and bins

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

* Exists for 8G models only

10.7.2 PIPE Interface Callbacks

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

Table 10-9 PIPE callbacks

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

Table 10-9 PIPE callbacks (Continued)

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

11 Using Callbacks

11.1 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 transaction, 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.)

11.2 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:

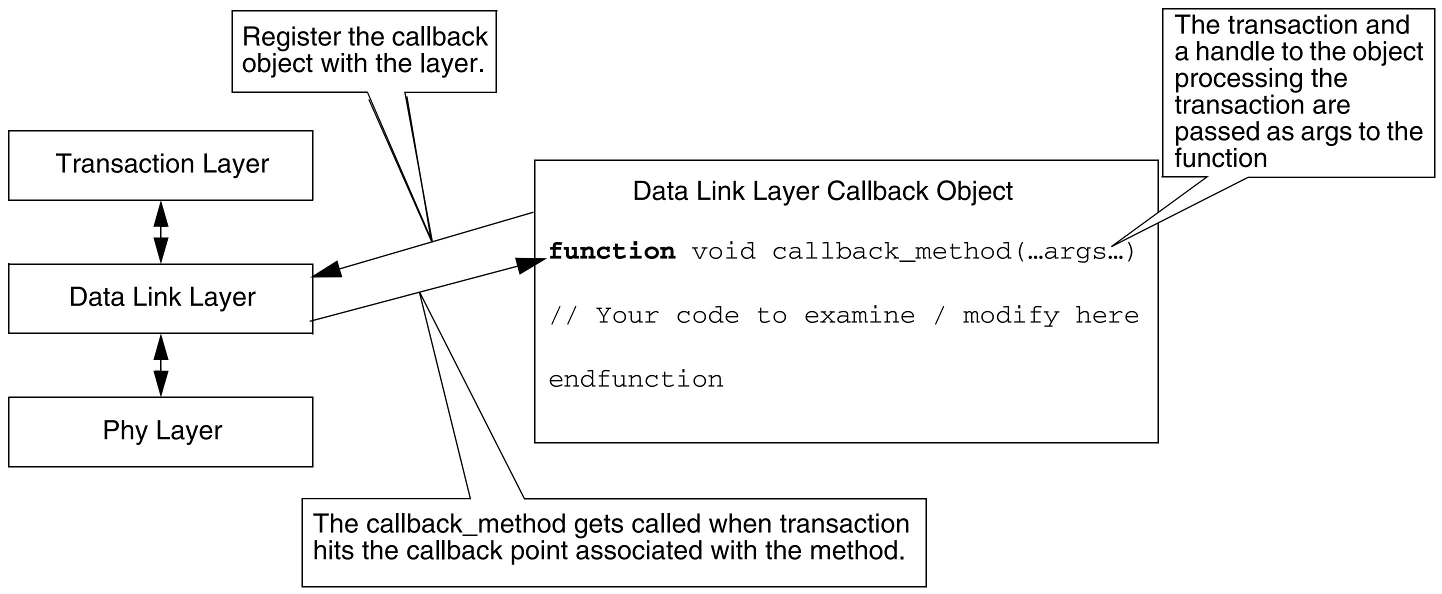
1. Identify which callback you need to use.
2. Sub-class the appropriate data type and implement the appropriate callback method with a user-defined action each time the callback is made.
3. Create an object of this type and add it to the queue of callback objects on the appropriate VIP instance.

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 11-1 is a diagram of the callback process for a callback that you request in the Data Link Layer.

Figure 11-1 Callback operation for a callback in the Data Link Layer


Register the callback object with the layer.
Transaction Layer
Data Link Layer
Phy Layer
Data Link Layer Callback Object
function void callback_method(…args…)
// Your code to examine / modify here
endfunction
The transaction and a handle to the object processing the transaction are passed as args to the function
The callback_method gets called when transaction hits the callback point associated with the method.

11.2.1 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

11.2.2 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.

11.2.3 When *Not* to Use a Callback

Although callbacks are a flexible mechanism, VMM analysis ports are preferred for statistics, coverage, scoreboarding, and so on. However VMM 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.

11.3 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.

11.3.1 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 VMM phase:
  - new() constructor
  - gen_config_ph() - Set up a test-specific configuration.
  - configure_test_ph() - Create and register the callback object
  - run_ph() - 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 11-1 Code for a basic testcase

```
typedef class tx_dl_tlp_callback; // Forward declaration

// The testcase class:
class dl_tlp_basic_callback_test extends vmm_test ;

   // Implements vmm_object::get_typename() method
   `vmm_typename( dl_tlp_basic_callback_test )

   //Declare system configuration
   pcie_shared_cfg cust_cfg;

   // Declare different scenarios
   pcie_device_random_traffic_scenario root_random_scenario;

   // Callback instance:
   tx_dl_tlp_callback dl_tlp_cb;
   // Constructor:
   function new(string name="dl_tlp_basic_callback_test");
      super.new(name);
   endfunction : new

   // Configure/build various components:
   virtual function void gen_config_ph();
      super. gen_config_ph();

      cust_cfg = new();

      /**  Reload test specific cust_cfg values. */
      cust_cfg.setup_pcie_device_system_defaults();
      cust_cfg.root_cfg.stop_after_n_driver_trans_insts = 16;
      cust_cfg.root_cfg.stop_after_n_driver_trans_scenarios = 1;

      // Set test-specific cfg values:
      cust_cfg.root_cfg.port_cfg.pl_cfg.set_link_width_values(4);
      cust_cfg.endpoint_cfg.port_cfg.pl_cfg.set_link_width_values(4);

      // Pass the configuration into the environment
      vmm_opts::set_object("env:cfg",cust_cfg);

   endfunction : gen_config_ph

   function void configure_test_ph();
      super. configure_test_ph ();

      //  build and register scenario
     root_random_scenario = new(cust_cfg.root_cfg, env.root_status,
                                 env.endpoint_status);

     env.root.driver_transaction_ms_scenario_gen[0].register_ms_scenario(
                             "root_random_scenario", root_random_scenario);


      // Create a callback object instance:
      dl_tlp_cb = new("dl_tlp_cb");
      env.root.pcie_group.dl.append_callback(dl_tlp_cb);

   endfunction : configure_test_ph

   task run_ph();
      // In this case, there is not much to do in this phase
      `vmm_note(log, "Running test dl_tlp_basic_callback_test .\n");
   endtask : run_ph

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;

   // 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);

      `vmm_note(log, $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));

      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
```

11.4 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 11-1).

11.4.1 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.

11.4.2 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:

```
// Exception handle and object
svt_pcie_dl_tlp_exception dl_tlp_exc = new();
```

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();
```

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.

11.4.3 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 <Xref>“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();
   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 gen_config_ph 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();
// Pass the constrained exception list to datalink layer transactor
   vmm_opts::set_object("@env:endpoint:port0:*dl0:dl_tlp_xact_tx_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.

11.4.4 Error Injection Using Application Layer TX Callbacks

This section shows you how to use Application Layer TX callbacks for error injection.

11.4.4.1 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 vmm_callbacks() and add() to include that callback object in the list of callback objects. This, of course, is standard VMM. Some minor additions are available.

11.4.4.2 Methods to Modify the Transaction

There are three ways to modifity the transaction:

❖ Simple Modifications.

❖ Exceptions
❖ Error Injection Exceptions

11.4.4.2.1 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.)

11.4.4.2.2 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 above 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.**

11.4.4.3 Malformed TLPs

If you intend to create 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 cancelled (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.

11.4.5 A More Comprehensive Example

In addition to the Test Case and Callback classes used in Example 11-1, three other classes are added in Example 11-2:

- 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
- Log catcher - Extended from vmm_log_catcher. It has several important methods:
    - new() constructor
- caught(vmm_log_msg msg) - Called upon an error

Example 11-2 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 cust_vmm_log_catch;
typedef class tx_dl_tlp_callback;

// The testcase class:
class dl_tlp_example_callback extends vmm_test ;
   // Implements vmm_object::get_typename() method
   `vmm_typename( dl_tlp_example_callback )


   // instantiate the relevant scenarios
      ....................

   // 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:
   cust_vmm_log_catch catcher;

   // Constructor "new":
   function new(string name="tlp_exception_via_callback");
      super.new(name);


   endfunction : new
```

```
    virtual function void gen_config_ph();
       super.gen_config_ph();
       // Load test specific cfg values:
       cust_cfg.root_cfg.port_cfg.pl_cfg.set_link_width_values(4);
       cust_cfg.endpoint_cfg.port_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);

      vmm_opts::set_object("env:cfg",cust_cfg);

     endfunction : gen_config_ph

    // build_ph:
    virtual function void build_ph();
       int err_catcher_id, warn_catcher_id;
       super.build_ph();

       // Create error report catcher object and register it:
       err_catcher = new();

       err_catcher_id = this.log.catch(catcher, "/.*/","/.*/",,,
                                      .severity(vmm_log::ERROR_SEV),
                                      .text("/Received TLP with invalid seq num /"));

       warn_catcher_id = this.log.catch(catcher, "/.*/","/.*/",,,
                                       .severity(vmm_log::WARNING_SEV),
                                       .text("/Received TLP with bad LCRC/"));

       endfunction

    function void config_test_ph();
       super. config_test_ph ();
       // Create the callback object and append it to dl transactor
       dl_tlp_cb = new("dl_tlp_cb");
       env.root.pcie_group.dl.append_callback(dl_tlp_cb);

       // Create dl_tlp_exc_list:
       dl_tlp_exc_list = new("dl_tlp_exc_list");

       // Pass the constrained exception list to Data Link Layer:
         vmm_opts::set_object("@env:endpoint:port0:*dl0:dl_tlp_xact_tx_exception_list",
 dl_tlp_exc_list);

    endfunction : config_test_ph


    task run_ph();
       // In this case, there is not much to do in this phase
       `vmm_note(log, "Running test dl_tlp_example_callback .\n");
    endtask : run_ph
```

```
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;
   // 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(log);
   // For DL/TLP transaction:
   svt_pcie_dl_tlp_exception my_dl_tlp_exc = new(log);

   // 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);
      `vmm_note(log,
         $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));
      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(log);

   // 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(log);
   // 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 WARNING and ERROR messages:
class cust_vmm_log_catch extends vmm_log_catcher;

  int n = 0;

  virtual function void caught(vmm_log_msg msg);
    msg.effective_typ = vmm_log::NOTE_TYP;
    msg.effective_severity = vmm_log::NORMAL_SEV;
```

```
      $display("Test(vmm_log_catcher) :: expected checker is fired");
      this.n++;
      this.throw(msg);
    endfunction

endclass
```

11.5 SVT VIP Callbacks Reference

The callbacks that currently are supported and their arguments are listed in <Xref>Table 11-1.

Callbacks and their methods for each layer are listed in <Xref>Table 11-2.

Table 11-1 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 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 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 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 object of the calling layer.
	svt_pcie_dlllp transaction		The dllp transaction t be examined/modified.
	bit drop	ref	When set, the transaction is dropped prior to transmission. NOTE: Currently not implemented

Table 11-1 Supported PCIe callbacks (Continued)

pre_tx_tlp_put Target Application Called just prior to sending this completion to the Transaction Layer.	svt_pcie_target_app target_app	I	The 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 reception.
post_rx_tlp_get Target Application Called just after reception of this request from the Transaction Layer.	svt_pcie_target_app target_app	I	The 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 reception.

Table 11-2 Callback classes and methods by layer

Layer	Callback Class	Method	Description
Driver App	svt_pcie_driver_app_callback	transaction_ended	Called by the 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 after recognizing a TLP transaction received immediately from the link.
		pre_tx_tlp_put	Called by the 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 after pulling a TLP out of its TLP input, but before acting on the TLP.
		pre_tlp_out_put	Called by the once the TLP is completely received and prior to putting the TLP on the rx port.

Table 11-2 Callback classes and methods by layer (Continued)

DL Layer	svt_pcie_dl_callback	post_tlp_framed_in_get	Called by the after recognizing a TLP transaction received immediately from the link.
		pre_tllp_framed_out_put	Called by the after scheduling a TLP transaction for transmission on the link, just prior to framing.
		rx_dllp_started	Called by the after receiving user DLLP transaction from an input port.
		tx_dllp_started	Called by the after constructing a DLLP transaction just prior to its further processing.
PL Layer	svt_pcie_pl_callback	pre_symbol_out_put	Called by the 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 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 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.

12 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.

12.1 Use Model

You can use the three new simulation targets in the Makefiles of the VIP VMM 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 VMM 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 VMM 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.

12.2 The “vcspcvlog” Simulator Target in Makefiles

The vcspcvlog simulator target in the Makefiles of the VIP VMM 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
```

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.

12.3 The “vcsmxpcvlog” Simulator Target in Makefiles

The vcsmxpcvlog simulator target in the Makefiles of the VIP VMM 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
```

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.

12.4 The “vcsmxpipvlog” Simulator Target in Makefiles

The vcsmxpipvlog simulator target in the Makefiles of the VIP VMM 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.

12.5 Partition Compile and Precompiled IP Implementation in Testbenches with Verification IPs

You can use the Makefiles in the VIP VMM 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 vmm_pkg and svt_vmm_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.

12.6 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 VMM Intermediate Example, use the following command:

   $VC_VIP_HOME/bin/dw_vip_setup -e pcie_svt/tb_pcie_svt_vmm_intermediate_sys -svtb

   cd examples/sverilog/pcie_svt/tb_pcie_svt_vmm_intermediate_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.

13 Integrated Planning for VC VIP Coverage Analysis

To leverage the Verdi planning and management solutions, the VIP verification plans in the .xml format were required to be converted to the .hvp format manually. Now, VC VIPs provide an executable Verification Plan in the .hvp format together with the .xml format. You can now load the VIP verification plans easily to the Verdi verification and management solutions in a single step. Further, you can easily integrate these plans to the top level verification plan by a single click drag and drop.

Coverage results are annotated to the plan that helps to map the verification completeness on a feature by feature basis at the aggregate level.

13.1 Use Model

The Verdi Coverage flow requires the .hvp files that capture the Verification Plan to be loaded on the Verdi Coverage Graphical User Interface (GUI), and the coverage annotated within the GUI.

To leverage the verification plan, perform the following steps:

1. Navigate to the example directory using the following command:

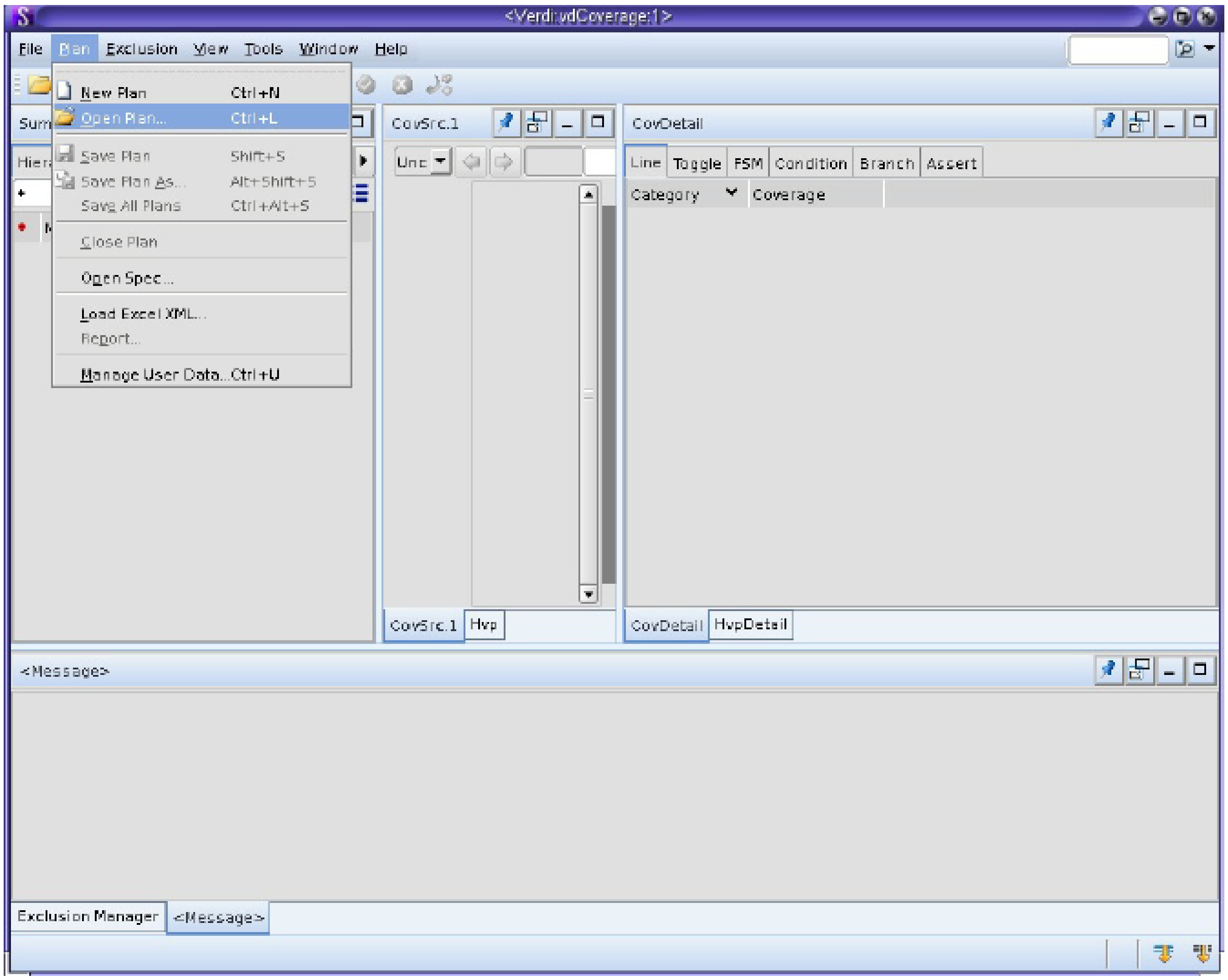
   cd <intermediate_example_dir>
2. 2.Invoke the make command with the name of the test you want to run, as follows:

   gmake USE_SIMULATOR=vcsvlog <test_name>
3. Invoke the Verdi GUI using the following command:

   verdi –cov &
4. Copy the Verification Plans folder from the installation path to the current work area, as follows:

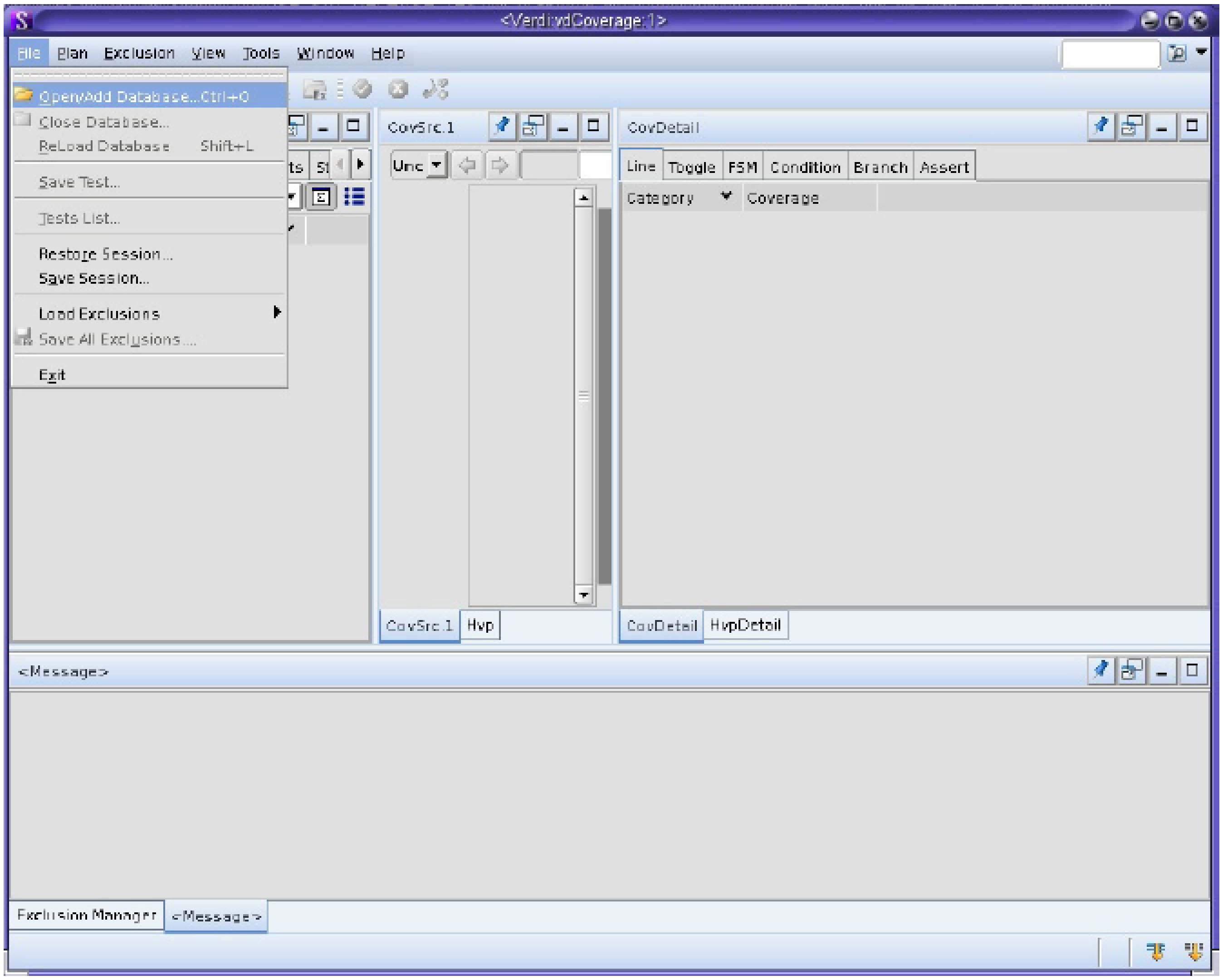
   cp $VC_VIP_HOME/vip/svt/pcie_svt/latest/doc/VerificationPlans .
5. Select the Load Plan menu itme from the Plan drop down as shown in the following illustration.

Figure 13-1 Open Plan



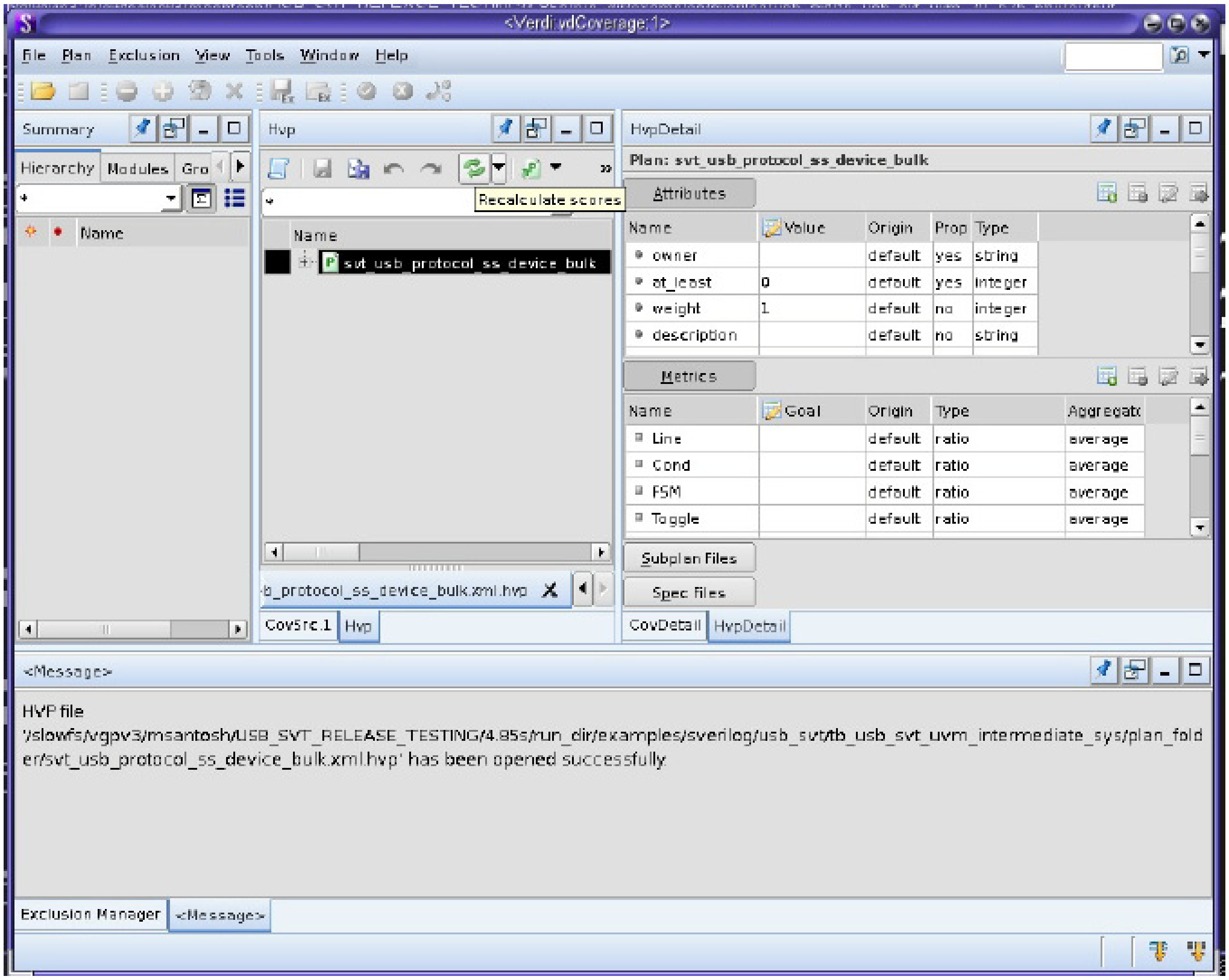
Load the coverage database (simvcssvlog.vdb) from the output folder in the current directory using the Verdi drop down menu, as shown in the following illustration.

Figure 13-2 Verdi GUI



Click Recalculate button to annotate the coverage results, as shown in the following illustration.

Figure 13-3 Recalculate



Plan: svt_usb_protocol_ss_device_bulk

Name	Value	Origin	Prop	Type
owner		default	yes	string
at_least	0	default	yes	integer
weight	1	default	no	integer
description		default	no	string

Name	Goal	Origin	Type	Aggregate
Line		default	ratio	average
Cond		default	ratio	average
FSM		default	ratio	average
Toggle		default	ratio	average

```
HVP file
'/slowfs/vgpv3/msantosh/USB_SVT_RELEASE_TESTING/4.85s/run_dir/examples/sverilog/usb_svt/tb_usb_svt_uvm_intermediate_sys/plan_fold
er/svt_usb_protocol_ss_device_bulk.xml.hvp' has been opened successfully
```

A Protocol Checks

A number of automatic protocol checks are built into the PCIe VIP, to test for compliance with the PCIe specification. The following tables list the protocol checks, with a short description each check and the version of the PCIe specification in which each check is specified.

- Phy Layer protocol checks are listed in Table A-1.
- Data Layer protocol checks are listed in Table A-2.
- Link Layer protocol checks are listed in Table A-3.
- Application Layer protocol checks are listed in Table A-4.

Table A-1 Phy Layer protocol compliance checks

Protocol Check Name and Description	Spec Version
MSGCODE_PCIESVC_PHY_EXCESS_LATENCY_DIFF RxDeskewLanes - Excessive latency differential between lanes.	All
MSGCODE_PCIESVC_PHY_MISMATCHED_DATA_RATES This lane's data rate identifier does not match previous lane's.	>=2.1
MSGCODE_PCIESVC_PHY_DETECT_QUIET_TIMEOUT Timeout in state LTSSM_DETECT_QUIET.	All
MSGCODE_PCIESVC_PHY_DETECT_ACTIVE_TIMEOUT Timeout in state LTSSM_DETECT_ACTIVE.	All
MSGCODE_PCIESVC_PHY_POLLING_ACTIVE_TIMEOUT Timeout in state LTSSM_POLLING_ACTIVE.	All
MSGCODE_PCIESVC_PHY_POLLING_CONFIGURATION_TIMEOUT Timeout in state LTSSM_POLLING_CONFIGURATION	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_LINKWIDTH_START_TIMEOUT Timeout in state LTSSM_CONFIGURATION_LINKWIDTH_START	All

Table A-1 Phy Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_PHY_CONFIGURATION_LINKWIDTH_ACCEPT_TIMEOUT Timeout in state LTSSM_CONFIGURATION_LINKWIDTH_ACCEPT	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_LANENUM_ACCEPT_TIMEOUT Timeout in state LTSSM_CONFIGURATION_LANENUM_ACCEPT	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_LANENUM_WAIT_TIMEOUT Timeout in state LTSSM_CONFIGURATION_LANENUM_WAIT	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_COMPLETE_TIMEOUT Timeout in state LTSSM_CONFIGURATION_COMPLETE	>=2.1
MSGCODE_PCIESVC_PHY_CONFIGURATION_IDLE_TIMEOUT Timeout in state LTSSM_CONFIGURATION_IDLE	<=2.1
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRLOCK_TIMEOUT Timeout in state LTSSM_RECOVERY_RCVRLOCK	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_TIMEOUT Timeout in state LTSSM_RECOVERY_RCVRCFG	All
MSGCODE_PCIESVC_PHY_RECOVERY_IDLE_TIMEOUT Timeout in state LTSSM_RECOVERY_IDLE	All
MSGCODE_PCIESVC_PHY_WRONG_LINK_WIDTH Negotiated link width does not match expected link width.	All
MSGCODE_PCIESVC_PHY_RESERVED_SPEED_BIT0 SetSupportedSpeeds task sets rate bits in Tx TS OS. rate[0] is reserved but may be set using this task. Msg issued to prevent unintended results.	All
MSGCODE_PCIESVC_PHY_MAX_RX_SKP_INTERVAL ReceivePhy: Exceeded maximum interval before receiving a skp ordered set.	All
MSGCODE_PCIESVC_PHY_DATA_OUTSIDE_PACKET Received data outside of packet instead of logical idle.	All
MSGCODE_PCIESVC_PHY_BAD_LINK_NUMBER_POLLING_ACTIVE LTSSM: Incoming training sets have link number set to non-PAD value.	All
MSGCODE_PCIESVC_PHY_BAD_LANE_NUMBER_POLLING_ACTIVE . LTSSM: Incoming training sets have lane number set to non-PAD value	All
MSGCODE_PCIESVC_PHY_ELASTIC_BUFFER_OVERFLOW Elastic buffer has overflowed.	All
MSGCODE_PCIESVC_PHY_ELASTIC_BUFFER_UNDERFLOW Elastic buffer has underflowed	All
MSGCODE_PCIESVC_PHY_BAD_LANE_SKEW RxDeskewLanes: Lane skew has exceeded the maximum allowed lane skew.	>=2.1

Table A-1 Phy Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_PHY_TOO_MANY_SKP_SETS "Received skip ordered sets on a lane, but expected to receive at fewer skip ordered sets."	All
MSGCODE_PCIESVC_PHY_TOO_FEW_SKP_SETS "Received skip ordered sets on a lane, but expected to receive at more skip ordered sets."	All
MSGCODE_PCIESVC_PHY_COMMA_IN_PACKET Received unexpected K28.5 character in middle of a packet.	All
MSGCODE_PCIESVC_PHY_CONTROL_CHAR_IN_PACKET Received unexpected control character in middle of a packet.	All
MSGCODE_PCIESVC_PHY_INFER_ELEC_IDLE_SKIP Electrical idle has been inferred on all lanes due to lack of SKP ordered sets.	>=2.1
MSGCODE_PCIESVC_PHY_UNEXPECTED_POLARITY_CHANGE Detected rx_polarity go high but SVC hasn't inverted polarity.	All
MSGCODE_PCIESVC_PHY_SPEED_CHANGE_NOT_IN_P0P1 Detected speed change while in powerdown is in state other than P0 or P1.	All
MSGCODE_PCIESVC_PHY_UNEXPECTED_PHYSTATUS Detected phystatus unexpectedly asserted.	All
MSGCODE_PCIESVC_PHY_PHYSTATUS_TIMEOUT Timeout waiting for phystatus acknowledgement.	All
MSGCODE_PCIESVC_PHY_VALID_SIGNAL_IN_P1 Detected a valid signal when attached device is supposed to be in P1 state.	All
MSGCODE_PCIESVC_PHY_VALID_SIGNAL_DURING_SPEED_CHANGE Detected valid signal on one or more lanes during a rate change.	All
MSGCODE_PCIESVC_PHY_TXDETRX_NOT_IN_P1 Receiver detect requested when powerdown not in P1 state.	All
MSGCODE_PCIESVC_PHY_DISABLED_TIMEOUT_NO_EIOS Timeout in this state. No EIOS was detected.	All
MSGCODE_PCIESVC_PHY_TXDETECTRX_DEASSERTED_BEFORE_PHYSTATUS Detected txdetectrx deasserted before phy_status acknowledgement.	All
MSGCODE_PCIESVC_PHY_SKP_INSERTED_NOT_ON_COMMA SKP inserted status received on pipe interface. Data was not K28.5 (Comma).	All
MSGCODE_PCIESVC_PHY_SKP_DELETED_NOT_ON_COMMA SKP deleted status received on pipe interface. Data was not K28.5 (Comma).	All
MSGCODE_PCIESVC_PHY_NO_LOOPBACK_PATTERN_DETECTED Did not detect transmitted loopback pattern on this lane before exiting loopback.	All

Table A-1 Phy Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_PHY_POLLING_COMPLIANCE_BAD_COMPLIANCE_RECEIVE Lane does not support 2.5GT/s. All lanes must advertise support for 2.5GT/s in compliance.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_START_NO_EIEOS LTSSM: Exceeded EIEOS interval of 32 without receiving an EIEOS.	>=2.0
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_UNEXPECTED_TS2 LTSSM: Received unexpected TS2 training set.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_UNEXPECTED_EIOS LTSSM: Received unexpected EIOS.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_UNEXPECTED_FTS LTSSM: Received unexpected FTS.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_BAD_LINK_NUM LTSSM: Detected incoming TS1 sets with link number set to value other than what SVC is configured to.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_LANE_NUM_NOT_PAD LTSSM: Incoming TS1 sets have lane number set to non-PAD.	All
MSGCODE_PCIESVC_PHY_CONFIG_LINKWIDTH_ACCEPT_BAD_LANENUM LTSSM: Incoming TS1 sets have lane number set to wrong value.	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRLOCK_NO_EIEOS LTSSM: Exceeded EIEOS interval of 32 without receiving an EIEOS in this state.	>=2.0
MSGCODE_PCIESVC_PHY_RECOVERY_SPEED_ELECTRICAL_IDLE_WITH_NO_EIOS LTSSM: Electrical idle detected on this lane without preceding EIOS.	All
MSGCODE_PCIESVC_PHY_RECOVERY_SPEED_INFERRED_ELECTRICAL_IDLE LTSSM: Inferred electrical idle on this lane.	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_NO_EIEOS LTSSM: Exceeded EIEOS interval of 32 without receiving an EIEOS.	>=2.0
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_INFERRED_ELECTRICAL_IDLE LTSSM: Inferred electrical idle on this lane.	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_BAD_LANE_SKEW LTSSM: TIMEOUT in this state - Unable to deskew lanes.	All
MSGCODE_PCIESVC_PHY_HOT_RESET_NO_TS1_ACKNOWLEDGEMENT LTSSM: Hot Reset timer expired without receiving ts1 hot reset handshake.	All
MSGCODE_PCIESVC_PHY_RX_N_FTS_TIMEOUT “N_FTS timeout, SKP OS not received after N_FTS ordered sets.”	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_COMPLETE_MISMATCHED_N_FTS LTSSM: This lane’s N_FTS value does not equal previous lane’s.	>=2.1

Table A-1 Phy Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_MISMATCHED_N_FTS LTSSM: This lane's N_FTS value does not equal previous lane's.	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_COMPLETE_BAD_LANE_SKEW LTSSM: Unable to deskew lanes.	>=2.1
MSGCODE_PCIESVC_PHY_LOOPBACK_ACTIVE_MISSING_EIOS Detected electrical idle on this lane without first detecting EIOS.	All
MSGCODE_PCIESVC_PHY_L0S_FTS_TIMEOUT LTSSM: N_FTS timeout on receiver in L0s.	All
MSGCODE_PCIESVC_PHY_L0_SIGNAL_LOSS Electrical idle detected on all lanes without first detecting EIOS.	All
MSGCODE_PCIESVC_PHY_CONFIGURATION_COMPLETE_SUPPORTED_SPEEDS_CHANGE Detected change of supported_speeds field. This field must not change in state.	All
MSGCODE_PCIESVC_PHY_RECOVERY_RCVRCFG_SUPPORTED_SPEEDS_CHANGE Detected change of supported_speeds field. This field must not change in state.	>=2.1
MSGCODE_PCIESVC_PHY_BAD_SPEED_CHANGE_REQUEST Speed change was requested but SVC did not advertise support for speeds greater than 2.5GT/s.	All
MSGCODE_PCIESVC_PHY_LOOPBACK_ENTRY_NO_COMPLIANCE_TIMEOUT Timeout in this state. No TS1 was detected with loopback enable set.	All
MSGCODE_PCIESVC_PHY_L0S_BAD_RX_FTS_COUNT Lane's FTS transmit count did not equal expected count before exiting L0S.	All
MSGCODE_PCIESVC_PHY_VALID_SIGNAL_DURING_PO_POWERDOWN_CHANGE Detected valid signal on one or more lanes before SVC has acknowledged transition to P0.	All
MSGCODE_PCIESVC_PHY_WRONG_PIPE_CLK_FREQ Pipe clock period does not match expected period.	All
MSGCODE_PCIESVC_PHY_LANE_REVERSAL_DETECTED SVC has detected a lane reversal. Swapping lane assignments.	All
MSGCODE_PCIESVC_PHY_MISMATCHED_LINK_UPCONFIGURE_BIT This lane's link upconfigure bit does not match previous lane's.	>=2.1
MSGCODE_PCIESVC_PHY_CONFIG_NON_PAD_LINK_LANE_NUM_FOR_NON_LINK_LANE LINK/LANE not PAD for lanes >= this lane. This lane is no longer part of the link.	All
MSGCODE_PCIESVC_PHY_CONFIG_NON_PAD_LINK_LANE_NUM_FOR_REVERSED_NON_LINK_LANE LINK/LANE not PAD for reversed lanes <= this lane. This lane is no longer part of the link.	All
MSGCODE_PCIESVC_PHY_RX_START_BLOCK_GEN1_GEN2 Detected rx_start_block != 0 asserted at speed less than 8G.	>=3.0

Table A-1 Phy Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_PHY_RX_SYNC_HEADER_GEN1_GEN2 Detected rx_sync_header != 0 at speed less than 8G.	>=3.0
MSGCODE_PCIESVC_PHY_RX_DATAK_GEN3 Detected nonzero value on pipe signal rx_datak at 8G.	>=3.0
MSGCODE_PCIESVC_PHY_8G_RATE_PIPE2 Rate set to 8G with the pipe spec version set to 2. Please check your configuration.	>=3.0
MSGCODE_PCIESVC_PHY_BLOCK_ALIGN_CONTROL_GEN1_GEN2 Detected block_align_control asserted at speed other than 8G.	>=3.0
MSGCODE_PCIESVC_PHY_RX_STP_TOKEN_BAD_FCRC_GEN3 Detected bad FCRC on incoming STP token.	>=3.0
MSGCODE_PCIESVC_PHY_RX_STP_TOKEN_BAD_PARITY_GEN3 Detected bad parity on incoming STP token.	>=3.0
MSGCODE_PCIESVC_PHY_RX_SDP_TOKEN_BAD_GEN3 Unexpected data in SDP token.	>=3.0
MSGCODE_PCIESVC_PHY_RX_TOKEN_MISMATCH_GEN3 Received token does not map to a recognizable token.	>=3.0
MSGCODE_PCIESVC_PHY_RX_NON_IDL_TOKEN_GEN3 Received non-IDL token outside of packet.	>=3.0
MSGCODE_PCIESVC_PHY_DSP_TS2_EQ_SYM6_BIT7_MUST_BE_1_IN_RCVRCFG “rx EQ TS2 symbol6, bit 7 should be 1 for Downstream port in Recovery.RcvrCfg @ 2.5G/5G.”	>=3.0
MSGCODE_PCIESVC_PHY_USP_TS2_SYM6_MUST_BE_45_NOT_EQ_IN_RCVRY rx EQ TS2 symbol6 should be x45 for Upstream port in Recovery @ 2.5G/5G.	>=3.0
MSGCODE_PCIESVC_PHY_RX_INVALID_BLOCK_SIZE_GEN3 CheckBlockSize: block size not 16.	>=3.0
MSGCODE_PCIESVC_PHY_DUT_ASSERTED_CLKREQ_L1_2_ENTRY “LTSSM: CLKREQ# asserted by DUT. SVC is in L1_2_ENTRY, not L1_2_IDLE.”	>=3.0
MSGCODE_PCIESVC_PHY_EQUALIZATION_BYPASSED This lane bypassed Equalization. Received TS2 in this state @ 8G when equalization_complete == 0.	>=3.0
MSGCODE_PCIESVC_PHY_EQUALIZATION_PHASE_2_NOT_ENABLED This lane executed Equalization Phase despite setting of ENABLE_EQUALIZATION_PHASE_VAR.	>=3.0
MSGCODE_PCIESVC_PHY_L0S_TOO_MANY_FTS_BETWEEN_EIEOS Received more than 32 FTS without detecting an EIEOS on this lane.	>=3.0
MSGCODE_PCIESVC_PHY_L0S_TOO_FEW_FTS_BETWEEN_EIEOS Received less than 32 FTS without detecting an EIEOS on this lane.	>=3.0

Table A-1 Phy Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_PHY_UNEXPECTED_COEFFICIENTS_VALID PIPE signal get_local_preset_coefficients asserted unexpectedly.	>=3.0
MSGCODE_PCIESVC_PHY_COEFFICIENTS_VALID_ASSERTED_TOO_LONG PIPE signal local_tx_coefficients_valid asserted for more than 1 clk cycle.	>=3.0
MSGCODE_PCIESVC_PHY_RXEQEVAL_DEASSERTED_EARLY PIPE signal RxEqEval deasserted before the evaluation was signaled complete.	>=3.0
MSGCODE_PCIESVC_PHY_GET_LOCAL_PRESET_COEFFICIENTS_TIMEOUT Timeout for get local preset coeffecient request.	>=3.0
MSGCODE_PCIESVC_SERDES_NEW_MIN_SEEN New min half bit period seen - SERDES unlocked.	All
MSGCODE_PCIESVC_SERDES_CLK_SLOWDOWN “Violation of 5 same bits period seen or New larger bit seen, but not at least 2x old bit - clock has likely slowed down - SERDES unlocked.”	All
MSGCODE_PCIESVC_SERDES_BYTE_UNLOCK “Lost valid signal level on receiver, PLL (clock) recovery reset.”	All
MSGCODE_PCIESVC_SERDES_PLL_UNLOCK “Lost valid signal level on receiver, PLL (clock) recovery reset.”	All
MSGCODE_PCIESVC_SERDES_LOSS_OF_BYTE_SYNC “Comma character detected in other than byte 0, bit 0. SERDES unlocked.”	All
MSGCODE_PCIESVC_ENDEC_ILLEGAL_ENCODING “Encoder: Kcode requested, but no legal encoding found!”	All
MSGCODE_PCIESVC_ENDEC_ILLEGAL_DECODE Decoder: Illegal decode received!	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_N_FTS Detected illegal control character in N_FTS field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_POLARITY_INVERSION Lane polarity inversion detected on incoming TS1 ordered set.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_OVERSIZE_EIOS Detected oversize EIOS.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_UNDERSIZE_EIOS Detected undersize EIOS.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_OVERSIZE_FTS Detected oversize FTS.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_UNDERSIZE_FTS Detected undersize FTS.	All

Table A-1 Phy Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_ORDERED_SET_CHECKER_OVERSIZE_SKIP Detected oversize SKP.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_UNDERSIZE_SKIP Detected undersize SKP.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_LINKNUM Detected illegal control character in Linknum field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_LANENUM Detected illegal control character in Lanenum field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_DATA_RATE Detected illegal control character in Data Rate Identifier field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_TRAINING_CTRL Detected illegal control character in Training Control field of TS1.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_LINKNUM Detected illegal control character in Linknum field of TS2.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_LANENUM Detected illegal control character in Lanenum field of TS2.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_N_FTS Detected illegal control character in N_FTS field of TS2.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_DATA_RATE Detected illegal control character in Data Rate Identifier field of TS2.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_TRAINING_CTRL Detected illegal control character in Training Control field of TS2.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_UNEXPECTED_COMMA Received K28.5 (Comma) in ordered set.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_ORDERED_SET Received bad ordered set.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKP_INSERT_STATUS Received SKP insert status on non-SKP ordered set.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_LINK_PAD_LANE_NON_PAD Received TS1 with PAD link number and non-PAD lane number.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_LINK_PAD_LANE_NON_PAD Received TS2 with PAD link number and non-PAD lane number.	All
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKIP_SIZE_8G “Invalid SKP ordered set length. Valid lengths are 8,12,16,20 and 24.”	>=3.0

Table A-1 Phy Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS1_SYMBOL_14_15_DC_BALANCE_REDUCE_ONES “Expected TS1 symbols 14 and 15 = {8’h20, 8’h08} for dc balance.”	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS1_SYMBOL_14_15_DC_BALANCE_REDUCE_ZEROES “Expected TS1 symbols 14 and 15 = {8’hdf, 8’hf7} for dc balance.”	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS1_SYMBOL_15_DC_BALANCE_REDUCE_ONES Expected TS1 symbol 15 = 8’h08 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS1_SYMBOL_15_DC_BALANCE_REDUCE_ZEROES Expected TS1 symbol 15 = 8’hf7 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_SYMBOL_15_DC_BALANCE_BYTE14 Expected TS1 symbol 14 = 8’h4a for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_IDENTIFIER_BYTE14 Bad TS1 identifier on byte 14.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS1_IDENTIFIER_BYTE15 Bad TS1 identifier on byte 15.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS2_SYMBOL_14_15_DC_BALANCE_REDUCE_ONES “Expected TS2 symbols 14 and 15 = {8’h20, 8’h08} for dc balance.”	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS2_SYMBOL_14_15_DC_BALANCE_REDUCE_ZEROES “Expected TS2 symbols 14 and 15 = {8’hdf, 8’hf7} for dc balance.”	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS2_SYMBOL_15_DC_BALANCE_REDUCE_ONES Expected TS2 symbol 15 = 8’h08 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_TS2_SYMBOL_15_DC_BALANCE_REDUCE_ZEROES Expected TS2 symbol 15 = 8’hf7 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_SYMBOL_15_DC_BALANCE_BYTE14 Expected TS2 symbol 14 = 8’h45 for dc balance.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_IDENTIFIER_BYTE14 Bad TS2 identifier on byte 14.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_TS2_IDENTIFIER_BYTE15 Bad TS2 identifier on byte 15.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_INVALID_BLOCK Received invalid ordered set block.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_EIOS_5G_2_NOT_RECEIVED	>=2.0

Table A-1 Phy Layer protocol compliance checks (Continued)

SVC Did not receive 2 back to back EIOS from DUT at 5G. Only recieved 1 EIOS.	
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKP_8G_SCRAMBLER_VALUE The LFSR value contained in the SKP ordered set does not match the receive scrambler value.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKP_8G_PARITY SKIP ordered set parity does not match expected value.	>=3.0
MSGCODE_PCIESVC_PCS_INVALID_SYNC_HEADER ReceivePMA: Invalid sync hdr detected.	>=3.0
MSGCODE_PCIESVC_PCS_SKP_END_NOT_DETECTED SKP_END was not detected before byte 20.	>=3.0
MSGCODE_PCIESVC_PHY_RX_SKP_NOT_ALL_LANES_8G Detected 8G SKP on at least 1 lane but not all.	>=3.0
MSGCODE_PCIESVC_PHY_RX_SKP_LEN_ALL_LANES_8G Detected 8G SKPs with different lengths.	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_SKP_MISSING_END_8G “Received SKP did not have SKP End indication on byte 4, 8, 12, 16 or 20.”	>=3.0
MSGCODE_PCIESVC_ORDERED_SET_CHECKER_BAD_SKP_8G Received start of next training set before previous SKP has completed.	>=3.0

Table A-2 Data Layer protocol compliance checks

Protocol Check Name and Description	Spec Version
MSGCODE_PCIESVC_DL_RECEIVE_TLP_LCRC ProcessReceivedTLP: Received TLP with bad LCRC	All
MSGCODE_PCIESVC_DL_RECEIVE_ILLEGAL_SEQ_NUM ProcessReceivedTLP: Received TLP with invalid sequence number.	All
MSGCODE_PCIESVC_DL_RECEIVE_NULLIFIED_TLP ProcessReceivedTLP: Received nullified TLP.	All
MSGCODE_PCIESVC_DL_RECEIVE_NULLIFIED_TLP_LCRC ProcessReceivedTLP: Received TLP with EDB delimiter and non inverted or bad LCRC	All
MSGCODE_PCIESVC_DL_RECEIVE_DUPLICATE_SEQ_NUM ProcessReceivedTLP: Received duplicate TLP.	All
MSGCODE_PCIESVC_DL_RECEIVE_DLLP_CRC ProcessReceivedDLLP: Received DLLP crc error.	All

Table A-2 Data Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_DL_REPLAY_TIMER_TIMEOUT UpdateReplayTimer: Replay timer timed out.	All
MSGCODE_PCIESVC_DL_REPLAY_LATE_TIMEOUT CheckAttachedTimers: Replay received late.	All
MSGCODE_PCIESVC_DL_REPLAY_EARLY_TIMEOUT "ProcessReceivedDLLP: Retry rcvd too early, attached replay timeout "	All
MSGCODE_PCIESVC_DL_RECEIVED_ACK_EXP_REPLAY_TIMEOUT "ProcessReceivedDLLP: EI - Rcvd ACK, replay timeout was expected."	All
MSGCODE_PCIESVC_DL_RECEIVED_NAK_EXP_REPLAY_TIMEOUT ProcessReceivedDLLP: EI - Received NAK when a replay timeout was expected.	All
MSGCODE_PCIESVC_DL_RETRY_BUFFER_FULL_BYTES GetNextTLP: Retry buffer full. consumed bytes > MAX_NUM_RETRY_BUFFER_DWORDS.	All
MSGCODE_PCIESVC_DL_RETRY_BUFFER_FULL_PKTS GetNextTLP: Retry buffer full. Num Pkts exceeded MAX_NUM_RETRY_BUFFER_PKTS.	All
MSGCODE_PCIESVC_DL_RECEIVED_DUPLICATE_ACK ProcessReceivedDLLP: Received duplicate ACK.	All
MSGCODE_PCIESVC_DL_RECEIVED_UNEXPECTED_NAK ProcessReceivedDLLP: Received Unexpected NAK.	All
MSGCODE_PCIESVC_DL_RECEIVED_UNEXPECTED_NAK_ACK ProcessReceivedDLLP: Received unexpected NAK which ACK'd some packets.	All
MSGCODE_PCIESVC_DL_RECEIVED_NAK_DURING_REPLAY "ProcessReceivedDLLP: EI - Received NAK as expected, but replay already in progress."	All
MSGCODE_PCIESVC_DL_RECEIVED_MULT_NAK_DURING_REPLAY "ProcessReceivedDLLP: Received more NAK during replay, expected 1. "	All
MSGCODE_PCIESVC_DL_RECEIVED_ACK_EXP_NAK ProcessReceivedDLLP: Received ACK when NAK expected.	All
MSGCODE_PCIESVC_DL_RECEIVED_ACK_PROTOCOL_VIOL "ProcessReceivedDLLP: Rcvd ACK with wrong sequence number, protocol violation ((((next_transmit_seq - 1) - acknak_seq_num) % 4096) > 2048)."	All
MSGCODE_PCIESVC_DL_RECEIVED_ACK_SEQ_NUM_NOT_FOUND PurgeRetryBuffer: Retry buffer is empty. seq num not found in retry buffer.	All
MSGCODE_PCIESVC_DL_RECEIVED_NAK_PROTOCOL_VIOL ProcessReceivedDLLP: Received NAK with protocol violation ((((next_transmit_seq - 1) - acknak_seq_num) % 4096) > 2048).	All
MSGCODE_PCIESVC_DL_RECEIVED_NAK_EXP_ACK	All

Table A-2 Data Layer protocol compliance checks (Continued)

PurgeRetryBuffer: EI - Purged packet from retry buffer. NAK not received as expected.	
MSGCODE_PCIESVC_DL_RECEIVED_NAK_WRONG_SEQ_NUM ProcessReceivedDLLP: Received NAK for wrong TLP. Packet was not sent with EI.	All
MSGCODE_PCIESVC_DL_ACKNAK_LATENCY_EARLY_TIMEOUT ProcessReceivedDLLP: ACK rcvd early.	All
MSGCODE_PCIESVC_DL_ACKNAK_LATENCY_LATE_TIMEOUT CheckAttachedTimers: ACK/NAK received late.	All
MSGCODE_PCIESVC_DL_UPDATEFC_CREDIT_TIMEOUT TransmitPhy: No VC/FC type Update credits were received within period specified.	All
MSGCODE_PCIESVC_DL_INITFC_CREDIT_TIMEOUT TransmitPhy: All 3 VC InitFC credit types were not received within specified period.	All
MSGCODE_PCIESVC_DL_RECEIVE_TLP_BEFORE_INITFC1 ReceiveTLP: Received TLP on VC before all 3 InitFC1s(P/NP/CPL). FC_INIT is not yet finished.	All
MSGCODE_PCIESVC_DL_RECEIVE_TLP_BEFORE_INITFC2 ReceiveTLP: Received TLP on VC before InitFC2. An InitFC2 is recommended before TLP.	All
MSGCODE_PCIESVC_DL_RECEIVED_DLLP_UNKNOWN_TYPE ProcessReceivedDLLP: Received DLLP with unknown type.	All
MSGCODE_PCIESVC_DL_ACK_NAK_RSVD_FIELD_NON_ZERO CheckDLLPRsvdFields: Received Ack/Nak DLLP with reserved field.	All
MSGCODE_PCIESVC_DL_FC_RSVD_FIELD_NON_ZERO CheckDLLPRsvdFields: Received InitFC with reserved field != 0.	All
MSGCODE_PCIESVC_DL_PM_RSVD_FIELD_NON_ZERO CheckDLLPRsvdFields: Received PM DLLP with reserved field !=0.	All
MSGCODE_PCIESVC_DL_RETRY_WRONG_EI_CODE PurgeRetryBuffer: EI - Retry Buffer packet has wrong EI_CODE.	All
MSGCODE_PCIESVC_DL_RETRY_WRONG_SEQ_NUM ProcessReceivedTLP: EI - Expected retry but received retry with wrong seq num.	All
MSGCODE_PCIESVC_DL_IDLE_STARTED_WRONG_LANE ReceivePhy: Idle started on wrong lane.Not implemented.	All
MSGCODE_PCIESVC_DL_IDLE_RECEIVED_IN_MIDDLE_OF_PACKET ReceivePhy: Received IDLE in middle of a packet.	All
MSGCODE_PCIESVC_DL_2_STP_RECEIVED_IN_1_SYMBOL_TIME "ReceivePhy: Received 2 STPs per symbol time, 2nd tlp started wrong lane."""	All

Table A-2 Data Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_DL_STP_RECEIVED_MISSING_END_EDB ReceivePhy: Received STP before previous packet's END/EDB.	All
MSGCODE_PCIESVC_DL_STP_NOT_ON_LANE0_AFTER_IDLE ReceivePhy: Received TLP not aligned to lane 0.	All
MSGCODE_PCIESVC_DL_STP_WRONG_LANE ReceivePhy: Received STP on lane which is not modulo 4.	All
MSGCODE_PCIESVC_DL_2_SDP_RECEIVED_IN_1_SYMBOL_TIME ReceivePhy: Received 2 SDPs per symbol time.	All
MSGCODE_PCIESVC_DL_SDP_RECEIVED_MISSING_END_EDB ReceivePhy: Received SDP before previous packet's END/EDB.	All
MSGCODE_PCIESVC_DL_SDP_NOT_ON_LANE0_AFTER_IDLE ReceivePhy: Received DLLP SDP on lane other than 0. Expected SDP on lane 0.	All
MSGCODE_PCIESVC_DL_SDP_WRONG_LANE ReceivePhy: Received SDP on lane which is not modulo 4.	All
MSGCODE_PCIESVC_DL_PACKET_EXCEEDED_CONTAINER_SIZE ReceivePhy: Packet exceeded SVC container size.	All
MSGCODE_PCIESVC_DL_DATA_RECEIVED_OUTSIDE_PACKET ReceivePhy: Data received outside packet.	All
MSGCODE_PCIESVC_DL_DLLP_SIZE_NOT_8_BYTES ReceivePhy: Received DLLP with length which is not 8 bytes.	All
MSGCODE_PCIESVC_DL_END_WRONG_LANE ReceivePhy: Received END in wrong lane.	All
MSGCODE_PCIESVC_DL_END_OUTSIDE_PACKET ReceivePhy: Received END outside packet.	All
MSGCODE_PCIESVC_DL_NULLIFIED_DLLP_RECEIVED ReceivePhy: Received nullified DLLP. This is illegal in PCIE.	All
MSGCODE_PCIESVC_DL_EDB_WRONG_LANE ReceivePhy: Received EDB on wrong.	All
MSGCODE_PCIESVC_DL_EDB_OUTSIDE_PACKET ReceivePhy: Received EDB outside packet.	All
MSGCODE_PCIESVC_DL_DLLP_EXCEEDED_8_BYTES ReceivePhy: DLLP exceeded 8 bytes.	All
MSGCODE_PCIESVC_DL_EDB_OUTSIDE_PACKET_8G ReceivePhy: Received errant EDB token on wrong lane or LCRC from previous nullified TLP was bad.	>=3.0

Table A-2 Data Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_DL_EDB_TOKEN_MISSING_8G ReceivePhy: Expected to Receive Phy EDB indication on particular lane.	>=3.0
MSGCODE_PCIESVC_DL_RECEIVED_TLP_FIFO_FULL ProcessReceivedTLP: TLP receive fifo full.	All
MSGCODE_PCIESVC_DL_RECEIVED_DLLP_FIFO_FULL ProcessReceivedDLLP: DLLP received fifo full. Packet will not be queued.	All
MSGCODE_PCIESVC_DL_SENT_DLLP_FIFO_FULL QueueSentDLLP: Sent DLLP fifo full. Packet will not be placed on the sent packet queue.	All
MSGCODE_PCIESVC_DL_CREDIT_SENT_VC_UNINITIALIZED TransmitCredits: Attempting to send credit on uninitialized VC.	All
MSGCODE_PCIESVC_DL_CREDIT_DUPL_INIT TransmitCredits: Init credit already received for VC/fc type.	All
MSGCODE_PCIESVC_DL_RECEIVED_PM_ENTER_L1_DOWNSTREAM "ProcessReceivedDLLP: PM_ENTER_L1 DLLP can only flow upstream, toward root complex."	All
MSGCODE_PCIESVC_DL_RECEIVED_PM_ENTER_L23_DOWNSTREAM "ProcessReceivedDLLP: PM_ENTER_L23 DLLP can only flow upstream, toward root complex."	All
MSGCODE_PCIESVC_DL_RECEIVED_PM_ASPM_REQUEST_L1_DOWNSTREAM "ProcessReceivedDLLP: PM_ACTIVE_STATE_REQUEST_L1 can only flow upstream, toward root complex."	All
MSGCODE_PCIESVC_DL_RECEIVED_PM_REQUEST_ACK_UPSTREAM "ProcessReceivedDLLP: PM_REQUEST_ACK can only flow downstream, away from root complex."	All
MSGCODE_PCIESVC_DL_RECEIVED_PM_IPG ProcessReceivedDLLP: PM_ENTER_L1 DLLP received with IPG > 4.	All
MSGCODE_PCIESVC_DL_ASPM_L0S_TIMEOUT CheckL0sIdle: ASPM L0s. Phy did not report TX L0s status before timeout occurred.	All
MSGCODE_PCIESVC_DL_ASPM_L1_HANDSHAKE_TIMEOUT ASPMHandshakeSm: ASPM L1 Handshake timeout. SVC Phy not in L1 before timeout occurred.	All
MSGCODE_PCIESVC_DL_ASPM_L1_1_HANDSHAKE_TIMEOUT ASPMHandshakeSm: ASPM L1_1 Handshake timeout. SVC Phy not in L1_1 before timeout occurred.	All
MSGCODE_PCIESVC_DL_ASPM_L1_2_HANDSHAKE_TIMEOUT ASPMHandshakeSm: ASPM L1_2 Handshake timeout. SVC Phy not in L1_2 before timeout occurred.	All
MSGCODE_PCIESVC_DL_PM_L1_HANDSHAKE_TIMEOUT PMHandshakeSm: PM L1 Handshake timeout. SVC Phy not in L1 before timeout occurred.	All
MSGCODE_PCIESVC_DL_PM_L1_1_HANDSHAKE_TIMEOUT PMHandshakeSm: PM L1_1 Handshake timeout. SVC Phy not in L1_1 before timeout occurred.	All

Table A-2 Data Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_DL_PM_L1_2_HANDSHAKE_TIMEOUT PMHandshakeSm: PM L1_2 Handshake timeout. SVC Phy not in L1_2 before timeout occurred.	All
MSGCODE_PCIESVC_DL_PM_L23_HANDSHAKE_TIMEOUT PMHandshakeSm: PM L23 Handshake timeout. SVC Phy not in L23 before timeout occurred.	All

Table A-3 Transaction Layer protocol compliance checks

Protocol Check Name and Description	Spec Version
MSGCODE_PCIESVC_TL_RECEIVE_TLP_ECRC ReceiveTLP: Received TLP with ECRC Error.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_IO_TC "CheckTLPHeader: Received IO TLP with TC != 0, expected TC0."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_IO_ATTR "CheckTLPHeader: Received IO TLP with ATTR[1:0] != 0, expected ATTR[1:0] == 0."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_IO_AT "CheckTLPHeader: Received IO TLP with AT[1:0] != 0, expected AT[1:0] == 0 per."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_IO_LENGTH "CheckTLPHeader: Received IO TLP with Length != 1 or last_dw_be != 0, expected Length = 1 and last_dw_be == 0."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_ROUTE_IO_TO_FUNCTION_FAILED ReceiveTLP: Could not map I/O Addr to Appl ID.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_ROUTE_MEM_TO_FUNCTION_FAILED ReceiveTLP: Could not map MEM Addr to Appl ID.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_ROUTE_MSG_TO_FUNCTION_FAILED ReceiveTLP: Could not map Rid to Appl ID.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_ROUTE_CPL_TO_FUNCTION_FAILED "ReceiveTLP: Could not map Rid to Appl ID, check BDF field."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_ROUTE_AT_TO_FUNCTION_FAILED ReceiveTLP: Could not map Mem addr to Appl ID.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_MSG_INT_TC "CheckTLPHeader: Received ERR* MSG TLP with TC != 0, but expected TC == 0"	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_MSG_TC "CheckTLPHeader: Received ERR* MSG TLP with TC != 0, but expected TC == 0."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_MSG_PWR_MGT_TC	All

Table A-3 Transaction Layer protocol compliance checks (Continued)

"CheckTLPHeader: Received PM* MSG TLP with TC != 0, but expected TC == 0"	
MSGCODE_PCIESVC_TL_RECEIVE_TLP_MSG_UNLOCK_TC "CheckTLPHeader: Received Unlock MSG TLP with TC != 0, but expected TC == 0."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_MSG_SET_SLOT_POWER_LIMIT_TC "CheckTLPHeader: Received Set Slot Power Limit MSG TLP with TC != 0, but expected TC == 0."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_MSG_LTR_TC "CheckTLPHeader: Received LTR MSG TLP with TC != 0, but expected TC == 0."	>=2.1
MSGCODE_PCIESVC_TL_RECEIVE_TLP_MSG_OBFF_TC "CheckTLPHeader: Received OBFF MSG TLP with TC != 0, but expected TC == 0."	>=3.0
MSGCODE_PCIESVC_TL_RECEIVE_TLP_FMT_2_NONZERO "CheckTLPHeader: Received TLP with FMT[2] != 0, but expected FMT[2] = 0 if PCIE Spec Version >=."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_UNKNOWN_TYPE "CheckTLPHeader: Received TLP with FMT[2] == 0, but expected FMT[2] == 0 if PCIE Spec Version < 2.1."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_PREFIX_UNSUP GetNumLocalEndPrefixes: Received TLP with prefixes. PCIE Spec Vers < 2.1 do not support prefixes.	<=2.1
MSGCODE_PCIESVC_TL_RECEIVE_TLP_UNSUPPORTED_LOCAL_PREFIX GetNumLocalEndPrefixes: Received TLP with unknown local prefix type.	>=2.1
MSGCODE_PCIESVC_TL_RECEIVE_HDR_RX_CREDIT_WHEN_INFINITE_SET UpdateLimitCredits: Non-zero updatefc header credit received after infinite advertisement.	All
MSGCODE_PCIESVC_TL_RECEIVE_DATA_RX_CREDIT_WHEN_INFINITE_SET UpdateLimitCredits: Non-zero updatefc data credit received after infinite advertisement.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_EXCEEDED_HDR_TX_CREDITS_ALLOCATED Received TLP hdr credits > allocated credits.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_EXCEEDED_DATA_TX_CREDITS_ALLOCATED Received TLP data credits > allocated credits.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_VC_NOT_ENABLED ReceiveTLP: Received TLP on VC which is not enabled.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_TC_NOT_MAPPED_TO_VC ReceiveTLP: Received TLP on TC which is not enabled.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_LEN_MISMATCH "ReceiveTLP: Malformed TLP, size mismatch."	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_ROUTE_CFG_TYPE0_TO_FUNCTION_FAILED ReceiveTLP: Could not map type0 Cfg Rid to Appl ID.	All

Table A-3 Transaction Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_TL_RECEIVE_TLP_ROUTE_CFG_TYPE1_TO_FUNCTION_FAILED ReceiveTLP: Could not map type1 Cfg Rid to Appl ID.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_TOO_MANY_END_TO_END_PREFIXES "GetNumLocalEndPrefixes: Received TLP with too many end-to-end prefixes. Per spec, max allowed = 4."	>=2.1
MSGCODE_PCIESVC_TL_TLP_LOCAL_PREFIX_INSIDE_END_TO_END_PREFIX GetNumLocalEndPrefixes: Received TLP with local prefix inside end-to-end prefixes.	>=2.1
MSGCODE_PCIESVC_TL_RECEIVE_TLP_POWER_LIMIT_MSG_NON_ZERO_DATA_31_10 ReceiveTLP: Received MSG TLP with non zero data[31:10].	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_FMT_TYPE_UNDEFINED ReceiveTLP: Received TLP XID has tlp_type which is not defined.	All
MSGCODE_PCIESVC_TL_TRANSMIT_TLP_FMT_TYPE_UNDEFINED GetHdrTLPFcType: Fmt/Type fields do not map to a known TLP type.	All
MSGCODE_PCIESVC_TL_TRANSMIT_TLP_DROPPED_TC_MAPPING_DISABLED SendPacket: TC mapping is not enabled.	All
MSGCODE_PCIESVC_TL_TRANSMIT_TLP_PKT_DROPPED_VC_DISABLED ParseVcQueue: VC is not enabled. TLP will be dropped.	All
MSGCODE_PCIESVC_TL_VC_ALREADY_ENABLED SetVcEnable: Attempting to enable VC which is already enabled. Ignoring.	All
MSGCODE_PCIESVC_TL_VC_ENABLED_DL_DOWN SetInitTxCredits: Attempting to set VC init credits but dl_status != DL_DOWN. Ignoring request.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_NO_SNOOP_SET ReceiveTLP: Received TLP with NO_SNOOP attribute set when ENABLE_NO_SNOOP configuration was not set.	All
MSGCODE_PCIESVC_TL_RECEIVE_TLP_UNSUPPORTED_END_PREFIX GetNumLocalEndPrefixes: Received TLP with unknown end-end prefix type.	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_HDR_DWORD0_RSVD_NON_ZERO CheckTLPHeader: Received TLP hdr word0 reserved field not 0	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_FIRST_DW_BE_0 "CheckTLPHeader: Received TLP hdr_length > 1, exp first_dw_be != 0	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_LAST_DW_BE_0 "CheckTLPHeader: Received TLP hdr_length > 1, exp last_dw_be != 0"	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_LAST_DW_BE_NOT_0 "CheckTLPHeader: Received TLP hdr_length == 1, exp last_dw_be == 0"	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_FIRST_DW_BE_NOT_CONTIGUOUS	All

Table A-3 Transaction Layer protocol compliance checks (Continued)

"CheckTLPHeader: Received TLP hdr_length >= 3 or (hdr_length == 2 && not qwork aligned), exp contiguous first_dw_be"	
MSGCODE_PCIESVC_TL_MALFORMED_TLP_LAST_DW_BE_NOT_CONTIGUOUS "CheckTLPHeader: Received TLP hdr_length >= 3 or (hdr_length == 2 && not qwork aligned), exp contiguous last_dw_be"	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_ATTR_NOT_ZERO "CheckTLPHeader: Received TLP, expected hdr_attr[1:0] = 0."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_READ_REQ_GREATER_MAX_READ_SIZE "CheckTLPHeader: Received TLP, hdr_length > max read request size."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_NON_MEM_REQ_TH_NON_ZERO" CheckTLPHeader: Received TLP, expected th = 0 for non memory requests"	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MEM_REQ_AT_3_RSVD_PER_ATS "CheckTLPHeader: Received TLP, AT != 0. AT is reserved memory requests."	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MEM_REQ_AT_1_LEN_0_PER_ATS CheckTLPHeader: Received TLP at == 1 but length[0] == 0 in violation of ATS spec 2.1.	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_FIRST_DW_BE_NON_ZERO "CheckTLPHeader: Received Atomic Op TLP with th == 0, exp first_dw_be == 0"	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_LAST_DW_BE_NON_ZERO "CheckTLPHeader: Received Atomic Op TLP with th == 0, exp last_dw_be == 0"	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_FETCH_ADD32_LEN_NOT_1_2 "CheckTLPHeader: Received 32 bit Fetch TLP, expect length == 1 or 2 "	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_FETCH_ADD64_LEN_NOT_1_2 "CheckTLPHeader: Received 64 bit Fetch TLP, expect length == 1 or 2"	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_FETCH_ADD64_ADDR_NOT_ALIGNED CheckTLPHeader: Received 64 bit Fetch TLP with address not aligned to operand size == 2	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_SWAP32_LEN_NOT_1_2 "CheckTLPHeader: Received 32 bit Swap TLP, expect length == 1 or 2"	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_SWAP64_LEN_NOT_1_2 "CheckTLPHeader: Received 64 bit Swap TLP, expect length == 1 or 2"	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_SWAP64_ADDR_NOT_ALIGNED CheckTLPHeader: Received 64 bit Swap TLP with address not aligned to operand size == 2	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_CAS32_LEN_NOT_2_4_8 "CheckTLPHeader: Received 32 bit Cas TLP, expect hdr_length == 2,4,8 "	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_CAS32_ADDR_NOT_ALIGNED "CheckTLPHeader: Received 32 bit Cas TLP, Address not aligned to operand size == 2	>=2.1

Table A-3 Transaction Layer protocol compliance checks (Continued)

MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_CAS64_LEN_NOT_2_4_8 "CheckTLPHeader: Received 64 bit Cas TLP, expect hdr_length == 2,4,8 "	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_ATOMIC_CAS64_ADDR_NOT_ALIGNED "CheckTLPHeader: Received 64 bit Cas TLP, Address not aligned to operand size == 2"	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_READ_CROSSED_4K_BOUNDARY CheckTLPHeader: Received TLP read which crossed 4kB boundary	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_EXT_TAG_EN_TAG_7_5_NON_ZERO "CheckTLPHeader: Received non-Posted TLP with Extended Tag Enable bit set, expect tag bits [7:5] != 0"	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_CPL_HDR_DWORD0_RSVD_NON_ZERO "CheckTLPHeader: Received CPL TLP, expect Reserved hdr0 bits[11:10] == 0."	1.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_CPL_AT_NON_ZERO "CheckTLPHeader: Received CPL TLP, expect AT[1:0] == 0."	>=2.0
MSGCODE_PCIESVC_TL_MALFORMED_TLP_CPL_NON_SUCCESSFUL_LEN_NON_ZERO "CheckTLPHeader: Received CPL TLP, Cpl status == non successful, expected length = 0."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_CPL_LEN_GREATER_MAX_PAYLOAD_SIZE "CheckTLPHeader: Received CPL TLP, Expect length < REMOTE_MAX_PAYLOAD_SIZE_VAR."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_RCVD_UPSTREAM_OF_SWITCH_R_OF_0 CheckTLPHeader: Received MSG TLP with r[2:0] == 0 on upstream port of a switch. spec 2.3.	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_PME_TO_ACK_RCVD_UPSTREAM_OF_SWITCH CheckTLPHeader: Received PME TO Ack TLP on upstream port of a switch. spec 2.3.	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_RCVD_DOWNSTREAM_OF_SWITCH_R_OF_3 CheckTLPHeader: Received MSG TLP with r[2:0] = 3 on upstream port of a switch. spec 2.3.	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_NOT_PME_TO_ACK_WITH_R_5 "CheckTLPHeader: Received MSG TLP with r[2:0] = 3'b101, only PME_TO_ACK can have this routing. spec 2.3."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MUST_HAVE_DATA_IF_EP_ASSERTED CheckTLPHeader: Received TLP with EP = 1. EP can only be asserted on TLPs with data.	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_AT_NON_ZERO "CheckTLPHeader: Received TLP with AT[1:0] != 0, but expected AT[1:0] == 0."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_INT_LEN_NON_ZERO "CheckTLPHeader: Received INT* MSG TLP, Length field != 0, but expected Length == 0 "	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_NON_ZERO_FUNCTION "CheckTLPHeader: Received INT* MSG TLP with non zero function, spec 2.2.8.1."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_LEN_NON_ZERO	All

Table A-3 Transaction Layer protocol compliance checks (Continued)

"CheckTLPHeader: Received ERR* TLP with Length field != 0, but expected Length == 0 ."	
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_PM_LEN_NON_ZERO "CheckTLPHeader: Received PM* TLP with Length field != 0, but expected Length == 0."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_PM_NON_ZERO_FUNCTION "CheckTLPHeader: Received PM* TLP with Function != 0, but expected Function == 0."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_UNLOCK_LEN_NON_ZERO "CheckTLPHeader: Received Unlock MSG TLP with Length field != 0, but expected Length == 0 ."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_SLOT_POWER_LIMIT_LEN_NON_ZERO "CheckTLPHeader: Received Slot Power Limit MSG TLP with Length field != 1, but expected Length = 1."	All
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_LTR_LEN_NON_ZERO "CheckTLPHeader: Received LTR MSG TLP with Length field != 0, but expected Length = 0."	>=2.1
MSGCODE_PCIESVC_TL_MALFORMED_TLP_MSG_OBFF_LEN_NON_ZERO "CheckTLPHeader: Received OBFF MSG TLP with Length field != 0, but expected Length = 0."	>=3.0
MSGCODE_PCIESVC_TL_MALFORMED_TLP_HDR_64_BIT_ADDR_LESS_THAN_4G CheckTLPHeader: Received TLP Address[63:32] != 0. Must not be 0	All All
MSGCODE_PCIESVC_TL_TLP_MSG_RCVD_DOWNSTREAM "CheckTLPHeader: Received INT* MSG TLP from downstream port, INTx MSGs can't be sent by downstream ports."	All
MSGCODE_PCIESVC_TL_TLP_MSG_SLOT_POWER_LIMIT_RCVD_DOWNSTREAM "CheckTLPHeader: Received Slot Power Limit MSG TLP at downstream port, which must not be generated by upstream ports."	All
MSGCODE_PCIESVC_TL_CFG_RCVD_UPSTREAM CheckTLPHeader: Received Cfg request from a downstream port.	All
MSGCODE_PCIESVC_TL_LOST_ALLOC_VS_RCVD_HDR_CREDIT_NOT_EQUAL_INIT_VAL CheckFinalCredits: VC allocated hdr credits != initial hdr credits.	All
MSGCODE_PCIESVC_TL_LOST_ALLOC_VS_RCVD_DATA_CREDIT_NOT_EQUAL_INIT_VAL CheckFinalCredits: VC allocated data credits != initial data credits.	All
MSGCODE_PCIESVC_TL_LOST_LIMIT_VS_CONSUME_HDR_CREDIT_NOT_EQUAL_INIT_VAL CheckFinalCredits: VC (limit - consumed) hdr credits != initial limit hdr credits.	All
MSGCODE_PCIESVC_TL_LIMIT_VS_CONSUME_DATA_CREDIT_NOT_EQUAL_INIT_VAL CheckFinalCredits: VC (limit - consumed) data credits != initial limit data credits.	All
MSGCODE_PCIESVC_TL_RECEIVED_CREDIT_ON_DISABLED_VC ReceiveCredits: Received credits on VC which is not enabled.	All

Table A-4 Application Layer protocol compliance checks

Protocol Check Name and Description	Spec Version
MSGCODE_PCIESVC_APPL_TARGET_INVALID_ID ReceivePacket: Bad appl_id received by application.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_REQ ReceivePacket: Received TLP with unsupported type field.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_MEM_REQ HandleMemReq: Received unsupported memory request. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_UNINITIALIZED_MEM_DATA HandleMemReq: MemRd accessed uninitialized memory data.	All
MSGCODE_PCIESVC_APPL_TARGET_HDR_WORD_SIZE_MISMATCH "HandleMemReq: Request has 4 dwords, but addr[63:32] == 0. Request is not a 64-bit memory request."	All
MSGCODE_PCIESVC_APPL_TARGET_DEPRECATED_CFG_REQ HandleCfgReq: Received unsupported (deprecated) Cfg request.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_CFG_TC HandleCfgReq: Received CFG with Illegal Traffic Class - expected 0.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_CFG_DATA_LEN HandleCfgReq: Received CFG with illegal data length - expected 1.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_CFG_AT HandleCfgReq: Received CFG with AT != 2'b00.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_CFG_LAST_BE HandleCfgReq: Received CFG with last DW byte enable != 0.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_IO_TC HandleIOReq: Received IO with illegal Traffic Class - expected 0.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_IO_DATA_LEN HandleIOReq: Received IO with illegal data length - expected 1.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_IO_AT HandleIOReq: Received IO with AT != 2'b00.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_IO_LAST_BE HandleIOReq: Received IO with unexpected last DW byte enable != 0.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_VENDOR0_MSG "HandleMsgReq: Received unsupported Vendor Defined 0 message, this is a UR."	All
MSGCODE_PCIESVC_APPL_TARGET_UNKNOWN_MSG_CODE HandleMsgReq: Received MSG with unknown message code.	All
MSGCODE_PCIESVC_APPL_TARGET_NONZERO_LAST_BE	All

Table A-4 Application Layer protocol compliance checks (Continued)

ByteEnableCheck: Last DW Byte Enable != 0 for length=1.	
MSGCODE_PCIESVC_APPL_TARGET_ZERO_FIRST_BE ByteEnableCheck: 1st DW Byte Enable == 0 for length > 1.	All
MSGCODE_PCIESVC_APPL_TARGET_ZERO_LAST_BE ByteEnableCheck: Last DW Byte Enable == 0 for length > 1.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_FIRST_BE_NCB_NQA ByteEnableCheck: Bad first DW Byte Enable - non-contiguous bits – non-QW-aligned.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_FIRST_BE_NCN_NQA ByteEnableCheck: Bad first DW Byte Enable - non-contiguous with next dword – non-QW-aligned.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_LAST_BE_NCB_NQA ByteEnableCheck: Bad last DW Byte Enable - non-contiguous bits – non-QW-aligned.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_LAST_BE_NCP_NQA ByteEnableCheck: Bad last DW Byte Enable - non-contiguous with prev dword – non-QW-aligned.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_FIRST_BE_NCB ByteEnableCheck: Bad first DW Byte Enable - non-contiguous bits.	All
MSGCODE_PCIESVC_APPL_TARGET_BAD_LAST_BE_NCB ByteEnableCheck: Bad last DW Byte Enable - non-contiguous bits.	All
MSGCODE_PCIESVC_APPL_TARGET_POISON_CFG_REQ HandleCfgReq: Received CFG with Poison bit set. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_INVALID_64_BIT_ADDR HandleMemReq: Received Unsupported 64-bit read request. SVC will return Completer Abort status	All
MSGCODE_PCIESVC_APPL_TARGET_POISON_IO_REQ HandleCfgReq: Received IOWr with Poison bit set. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_MSG_DATA_LEN HandleMsgReq: Received MSG with reserved data length field !=0.	All
MSGCODE_PCIESVC_APPL_TARGET_MSG_INTX_ON_DOWNSTREAM_PORT HandleMsgReq: Received INTx on downstream port.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_MSG_AT HandleMsgReq: Recevied MSG TLP with AT != 2'b00.	All
MSGCODE_PCIESVC_APPL_TARGET_ILLEGAL_MSG_RSVD_HDR "HandleMsgReq: Reserved vendor defined msg has routing by addr (illegal) [2.2.8.6], hdr[2] == 0, or hdr[2] == 0."	All
MSGCODE_PCIESVC_APPL_TARGET_UNEXPECTED_TYPE_1_CFG HandleCfgReq: Received unexpected Type 1 Cfg request. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_ATOMIC_OP_REQ	<=2.0

Table A-4 Application Layer protocol compliance checks (Continued)

HandleMemReq: Received Illegal atomic-op request. Atomic Ops not supported in PCIE version < 2.1. Svc will return UR.	
MSGCODE_PCIESVC_APPL_TARGET_CFG_INVALID_FUNC_NUM HandleCfgReq: Received CfgRd with Invalid Function Number. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_CFG_INTERNAL_PROBLEM HandleCfgReq: Received CfgRd with invalid function.(returning CA).	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_VENDOR1_MSG HandleMsgReq: Received unsupported Vendor Defined 1 message.	All
MSGCODE_PCIESVC_APPL_TARGET_POISON_MEM_WR HandleMemReq: Poison bit set on received MemWr. SVC will return UR.	All
MSGCODE_PCIESVC_APPL_TARGET_UNSUPPORTED_PREFIX Unsupported TLP prefix type received. Discarding malformed TLP.	>=2.1
MSGCODE_PCIESVC_APPL_TARGET_MAX_SUPPORTED_PREFIX TLP contains too many end-end TLP prefixes.	>=2.1
MSGCODE_PCIESVC_APPL_DRIVER_COMMAND_TIMEOUT CheckForCommandTimeout: Request with a particular tag has timed out.	All
MSGCODE_PCIESVC_APPL_DRIVER_INVALID_MEM_DATA CheckReceivedData: Data uninitialized at shadow addr.	All
MSGCODE_PCIESVC_APPL_DRIVER_DATA_MISMATCH CheckReceivedData: Data mismatch at shadow addr.	All
MSGCODE_PCIESVC_APPL_DRIVER_LOW_BYTE_COUNT Byte count field is less than expected value. Discarding completion.	All
MSGCODE_PCIESVC_APPL_DRIVER_HIGH_BYTE_COUNT Byte count field is greater than expected value. Discarding completion.	All
MSGCODE_PCIESVC_APPL_DRIVER_MISSING_GOOD_STATUS ProcessReceivedPacket: Received completion with unexpected status.	All
MSGCODE_PCIESVC_APPL_DRIVER_UNEXPECTED_GOOD_STATUS "ProcessReceivedPacket: Received completion status of GOOD, but expected status other than GOOD."	All
MSGCODE_PCIESVC_APPL_DRIVER_BAD_RECD_LEN CheckReceivedData: TLP received length != expected length in bytes	All
MSGCODE_PCIESVC_APPL_DRIVER_SPURIOUS_CPL Spurious completion received.	All
MSGCODE_PCIESVC_APPL_DRIVER_MISMATCHED_CPL_ATTR Completion attr does not match request attr. Discarding completion	All
MSGCODE_PCIESVC_APPL_DRIVER_MISMATCHED_CPL_TC	All

Table A-4 Application Layer protocol compliance checks (Continued)

Completion TC does not match request TC. Discarding completion	
MSGCODE_PCIESVC_APPL_REQUESTER_UNINITIALIZED_MEM_DATA CheckReceivedData: Data uninitialized at shadow addr.	All
MSGCODE_PCIESVC_APPL_REQUESTER_DATA_MISMATCH CheckReceivedData: Data mismatch at shadow addr.	All
MSGCODE_PCIESVC_APPL_REQUESTER_BAD_RECD_LEN CheckReceivedData: Bad received length != expected length in bytes.	All
MSGCODE_PCIESVC_APPL_REQUESTER_BAD_RECD_BYTE_CNT HandleMemCpl: Received byte_cnt != expected byte_cnt.	All
MSGCODE_PCIESVC_APPL_REQUESTER_COMMAND_TIMEOUT ReceivePacket: Request has timed out waiting for Cpl.	All
MSGCODE_PCIESVC_APPL_REQUESTER_SPURIUS_CPL HandleMemCpl: Spurious Cpl packet received. Potentially late Cpl for timed-out request.	All
MSGCODE_PCIESVC_APPL_REQUESTER_TIMEOUT_BEYOND_RETRY_COUNT "ReceivePacket: Timed out request, retry_count is beyond max retry count - failing request."	All
MSGCODE_PCIESVC_APPL_REQUESTER_TIMEOUT_MISSING_CPL MarkTimedOutRequests: Timeouts caused some missing Cpl.	All

B PCIe Compile-time Parameters

Parameters that must be set before compilation are listed in the following sections:

- ❖ Model Parameters
- ❖ Driver Application Parameters
- ❖ Requester Parameters
- ❖ Completion Target Parameters
- ❖ Memory Target Parameters
- ❖ Transaction Layer Parameters
- ❖ Data Link Layer Parameters
- ❖ Physical Layer Parameters
- ❖ Physical Coding Sublayer (PCS) Parameters
- ❖ Serializer/Deserializer (SERDES) Parameters

B.1 Model Parameters

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

Table B-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
PCIE_SPEC_VER	Real	1.1, 2.0, 2.1, 3.0	PCIE_SPEC_VER_3_ 0	See Include/pciesvc_parms.v: PCIE_SPEC_VER_* Note: Please set this here, not in the individual layers.

Table B-1 Parameters set inthe model (Continued)

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.
ENABLE_SHADOW_ MEMORY_CHECKING	Integer	0-1	1	If set, applications will check memory reads against the shadow memory.
DISPLAY_NAME	String		"pciesvc_xx_yy_model _zz." (Specific to each model).	String prefixed to messages to display in the output log.

B.2 Driver Application Parameters

Driver parameters are listed in Table B-2.

Table B-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.

B.3 Requester Parameters

Requester parameters are listed in Table B-3.

Table B-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 B-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.
DISPLAY_NAME				
	String		pciesvc_ requester	Default prefix in $msglog calls.

B.4 Completion Target Parameters

Completion target parameters are listed in Table B-4.

Table B-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.

B.5 Memory Target Parameters

Memory target parameters are listed in Table B-5.

Table B-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				

Table B-5 Memory target parameters (Continued)

	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.

B.6 Transaction Layer Parameters

Transaction Layer parameters are listed in Table B-6.

Table B-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.
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				

Table B-6 Transaction Layer parameters (Continued)

	Integer	0-1	0	Stack direction. Used for TLP header checking/routing.
IS_SWITCH				
	Integer	0-1	0	Used for TLP header checking/routing.

B.7 Data Link Layer Parameters

Data Link Layer parameters are listed in Table B-7.

Table B-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.

B.8 Physical Layer Parameters

B.8.1 General Parameters

General Physical Layer parameters are listed in Table B-8.

Table B-8 Physical Layer general parameters

Parameter Name	Type	Range	Default	Description
DISPLAY_NAME				
	String		pciesvc_pl0	Default display name for the Physical Layer.

B.8.2 Physical Layer LTSSM-specific Parameters

Physical Layer LTSSM-specific parameters are listed in Table B-9.

Table B-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.

B.8.3 Equalization Parameters

Equalization parameters are listed in Table B-10.

Table B-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.

B.9 Physical Coding Sublayer (PCS) Parameters

PCS parameters are listed in Table B-11.

Table B-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.

B.10 Serializer/Deserializer (SERDES) Parameters

Parameters that affect the behavior of the SERDES are listed in Table B-12.

Table B-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				
	Integer	Any value	0.0001	Tolerance in ns. Equivalent to 100 PPM SAS requirement. This should not be changed as it affects SSC clock tracking.
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.

C Verilog Task/Parameter to SVT Class Mapping

This appendix provides mapping from SVC Verilog or SVT Verilog tasks and parameters to SVT VMM class members, for users who are migrating from the SVC or SVT Verilog PCIe VIP to the VC VMM PCIe.

This appendix contains the following sections:

- Transaction Layer Verilog Tasks and Parameters to VMM Class Members Map
- Data Link Layer Verilog Tasks and Parameters to VMM Class Member Maps

C.1 Transaction Layer Verilog Tasks and Parameters to VMM Class Members Map

This section contains the following tables:

- Transaction Layer Verilog Task to VMM Class Member Map
- Transaction Layer Verilog Parameters to VMM Class Members Map

C.1.1 Transaction Layer Verilog Task to VMM Class Member Map

Transaction Layer Verilog tasks are mapped to SVT class members in Table C-1, listed alphabetically by Verilog task.

Table C-1 Map of Transaction Layer Verilog tasks to VMM class members

Verilog Task	VMM 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
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

Table C-1 Map of Transaction Layer Verilog tasks to VMM class members (Continued)

Verilog Task	VMM Class Member
DisplayIOAddrApplidMap()	svt_pcie_tl_service::service_type
DisplayMemAddrApplidMap()	svt_pcie_tl_service::service_type
DisplayRequesterIdApplidMap()	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

C.1.2 Transaction Layer Verilog Parameters to VMM Class Members Map

Transaction Layer Verilog parameters are mapped to VMM class members in Table C-2, listed alphabetically by Verilog parameter.

Table C-2 Map of Transaction Layer class members to parameters

Verilog Parameter	VMM 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
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

Table C-2 Map of Transaction Layer class members to parameters (Continued)

Verilog Parameter	VMM Class Member
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

C.2 Data Link Layer Verilog Tasks and Parameters to VMM Class Member Maps

- Data Link Layer Verilog Task to VMM Class Member Map
- Data Link Layer Verilog Parameter to VMM Class Member Map

C.2.1 Data Link Layer Verilog Task to VMM Class Member Map

Data Link Layer Verilog tasks are mapped to VMM class members in Table C-3, listed alphabetically by Verilog task.

Table C-3 Map of Data Link Layer Verilog tasks to VMM class members

Verilog Task	VMM 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
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

Table C-3 Map of Data Link Layer Verilog tasks to VMM class members (Continued)

Verilog Task	VMM Class Member
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

C.2.2 Data Link Layer Verilog Parameter to VMM Class Member Map

Data Link Layer Verilog parameters are mapped to VMM class members in Table C-4, listed alphabetically by Verilog parameter.

Table C-4 Map of Data Link Layer class members to tasks

Verilog Parameter	VMM 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
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

Table C-4 Map of Data Link Layer class members to tasks (Continued)

Verilog Parameter	VMM Class Member
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
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

Table C-4 Map of Data Link Layer class members to tasks (Continued)

Verilog Parameter	VMM Class Member
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

D Reporting Problems

D.1 Introduction

This chapter outlines the process for working through and reporting problems. It shows you how to use Debug Automation to enable all the debug capabilities of any SVT-based VIP. In addition, the VIP provides a case submittal tool to help you pack and send all pertinent debug information to Synopsys Support.

D.2 Debug Automation

Every Synopsys model contains a feature called “debug automation”. Debug Automation allows you to enable all relevant debug information. 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.
- Generates a file named *svt_debug.out* which contains information about the debug feature itself, along with information about the environment that the VIP is operating in (such as package timeunits and VIP versions). The file is located in the directory where you invoke the simulator.

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*.

D.2.1 Enabling and Specifying Debug Automation Features

Debug Automation is enabled through the use of a run-time plusarg named +*svt_debug_opts*. This 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 below.

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_string>,type:<string>,feature:<string>,start_time:<longint>,end_time:<longin
t>,verbosity:<string>
```

The following table explains each control string.

Table D-1 Control Strings for Debug Automation plusarg

Field	Description
inst_string	Identifies the VIP instance to apply the debug automation features to. 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 SVT VIP instances.
type	Identifies an SVT class type to apply the debug automation features to. 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.
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 in. 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 in. 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 have been enabled for debug will have their messages routed to a file named svt_debug.transcript.

D.2.2 Debug Automation Outputs

The Automated Debug feature generates a file named *svt_debug.out*. 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

D.3 Sending Debug Information to Synopsys

To help you debug testing issues, follow the instructions below 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. Create a value change dump (VCD) file. For a Verilog or SystemVerilog simulation, generate a VCD waveform file by specifying the following in your top level testbench.

   ```
   initial begin
     $dumpvars;
   end
   ```

   When the simulation is completed, a VCD file named *verilog.dump* will be present in the runtime directory.
4. 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 "<username>.<uniqid>.svd" file is a TGZ type file that contains all pertinent transcripts, trace files, dump files and/or debug output files that exist in the current directory at the time of tool invocation.

   The -directory switch can be specified to select an alternate source directory.
5. You will be prompted by the case submittal tool with the option to include additional files within the SVD file.
6. The case submittal tool will display options on how to send the file to Synopsys.

To unpack the *. svd file, use the following command:

```
tar zxf <svd_file>
```