Verification Continuum™

VC Verification IP
PCIe Test Suite
UVM User Guide

Version R-2021.03-2, April 2021

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

www.synopsys.com

Contents

Preface

About This Manual

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

Guide Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Introduction", introduces the VC VIP PCIe Test Suite and its features.
- Chapter 2, "Products and Dependencies", lists the supported and dependent VIP products.
- Chapter 3, "Installation and Setup", describes system requirements and provides instructions on how to install, configure, and begin using the VC VIP PCIe Test Suite.
- Chapter 4, "Test Suite Testbench Architecture", describes Unified Testbench architecture and simulation framework.
- Chapter 5, "Integrating EP, RC and PHY", provides detailed steps for integrating EP/RC/PHY DUT in the environment.
- Chapter 6, "Using the Test Suite: EP DUT", describes the use model of EP VIP. It also describes detailed usage of optional features supported by EP VIP when used as DUT.
- Chapter 7, "Using the Test Suite: RC DUT", describes the use model of RC VIP. It also describes detailed usage of optional features supported by RC VIP when used as DUT.
- Chapter 8, "Using the Test Suite: PHY DUT", describes the use model of PHY VIP. It also describes detailed usage of optional features supported by PHYVIP when used as DUT
- Chapter 9, "Multi-Link Controller DUT", describes the use model of Test Suite when used in multi-link environment.
- Chapter 10, "Frequently Asked Questions", covers frequently asked questions.
- Appendix A, "Reporting Problems", outlines the process for working through and reporting VC VIP PCIe Test Suite issues.

- Appendix B, "Testbench (tb_dut_pcie) Environment Changes and New Use Model", outlines the tb_dut_pcie testbench changes and the new use model for extended base test.
- Appendix C, "Integrating Synopsys IIP (default configuration) in Test Suite Testbench", outlines the integration steps for test suite evaluation with IIP.
- Appendix D, "Implementing Partition Compile in Testbench", describes the PC features in Verification Compiler to optimize the compilation performance.

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 A, "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 A, "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

The Introduction chapter discusses the following topics:

- Overview
- Products Included in the Test Suite
- Source Code Language/Methodology

1.1 Overview

This document specifies the structure and usage of the PCIe VIP Test Suite products from the end user's perspective.

The PCIe VIP Test Suite products are packaged and delivered in a manner equivalent to that of the VIP component suites (separate products). Such products are packaged along with other products they depend on in a self-installer file, referred to as a ".run" file. When executed, the .run file extracts each included product and installs it into its own subtree within a directory specified by the user's DESIGNWARE_HOME environment variable. By referencing a common DESIGNWARE_HOME each of the user's products has access to the other products upon which it depends. This is referred to as the "product context".

The IIP core used as a demonstration RTL DUT by the PCIe Test Suite is the Synopsys DWC_pcie core. If the PCIe Test Suite testcases are run with this RTL for demonstration purposes, the core must be installed separately into the same DESIGNWARE_HOME and then built into the PCIe Test Suite testbench.

The VIP components used by the PCIe VIP Test Suite are from the UVM-capable pcie_svt, amba_svt, and mphy_svt VIP suites (see "VIP Dependencies").

1.2 Products Included in the Test Suite

The PCIe Test Suite product consist of the following:

- *pcie_test_suite_svt* – Test Suite for testing PCIe Gen1/Gen2/Gen3/Gen4/Gen5 *
- 2.5GT/s + 5GT/s + 8GT/s + 16GT/s + 32GT/s speeds supported via Gen1/Gen2/Gen3/Gen4/Gen5* protocol interfaces
- *tb_dut_pcie*: This is the testbench directory to run Test Suite test cases.

In the descriptions below the '*Product Context*' refers to the installation resulting from the product release trees.

* Gen5 is an EA feature and applicable only for serial interface and EP DUT.

The general content/usages of the subdirectories as described here:

- *doc*

  Test suite user documents, including User Manuals, HTML, Release Notes, are found in this directory.

  - Product Context:
    $DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/version/doc

- *sverilog/include*

  Non-encrypted Test Suite product common files, such as files containing `**define** symbolic defines for values used by the VIP, are found in this directory.

  - Product Context:
    $DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/version/sverilog/include

- *sverilog/src*

  Test Suite product common source files are found in this directory.

  - Product Context:
    $DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/version/sverilog/src/[vcs | mti | ncv]

Note The leaf directory is simulator-specific.

- *examples/sverilog*

  The Test Suite testbenches are found in this subtree. The Test Suite products do not support a command mode interface, so the sverilog tree is the only one under examples.

  - *tb_dut_pcie*
    - Product Context:
      $DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/version/examples/sverilog/tb_dut_pcie
    - *dut*
    - *env*
    - *tests*

1.3 Source Code Language/Methodology

All PCIe VIP Test Suite source code was written in the SystemVerilog language. The Test Suite source code is qualified with UVM 1.2 version.

2 Products and Dependencies

The Products and Dependencies chapter discusses the following topics:

2.1 Overview

The PCIe VIP Test Suite products can be seen as individual suites within the VIP product line. However, just as some VIP product suites depend on other VIP product suites, the PCIe VIP Test Suites depend on other VIP product suites. The PCIe VIP Test Suite product also depends upon the IIP `DWC_pcie Core` product to act as a DUT model in test cases that demonstrate the integration with an RTL DUT.

2.2 PCIe Test Suite Products

The PCIe VIP Test Suite product are each targeted to a specific protocol level. The products consist of the following:

- *pcie_test_suite_svt*

  This Test Suite product is intended for testing a DUT that supports only a PCIe Gen1/Gen2/Gen3/Gen4/Gen5 physical connection.

- Unified Testbench

  Previously, there were separate testbench environments based on DUT/Rate/Interface type which required multiple DUT integrations. Using the Unified testbench, you can access all test environments from a single testbench. The Unified testbench is located at the same directory level as *tb_dut_pcie* testbech examples.

Following are the available testbenches:

- `tb_dut_ep_gen2_pipe`
- `tb_dut_ep_gen2_serdes`
- `tb_dut_ep_gen3_pipe`
- `tb_dut_ep_gen3_serdes`
- `tb_dut_ep_gen4_pipe`
- `tb_dut_ep_gen4_serdes`
- `tb_dut_rc_gen2_pipe`
- `tb_dut_rc_gen2_serdes`
- `tb_dut_rc_gen3_pipe`
- `tb_dut_rc_gen3_serdes`
- `tb_dut_rc_gen4_pipe`
- `tb_dut_rc_gen4_serdes`

2.3 VIP Dependencies

The following VIP suites are utilized within the PCIe Test Suite testbenches, and thus are required to be present in the user's *DESIGNWARE_HOME/vip/svt* installation tree.

- *pcie_svt*

  The *pcie_svt* VIP is used within the PCIe Test Suite product testbenches to support the connections to the PCIe interfaces of the DUT, per the applicable protocol specification.

- *mphy_svt*

  Components from the *mphy_svt* VIP are incorporated into the *pcie_svt* VIP to support the testing of DUTs using the mPCIe interface.

- *amba_svt*

  Components from the *amba_svt* VIP (specifically AXI Master / Slave components) are included as part of the default example testbenches provided by the PCIe Test Suites, to support the *application bus* interfaces of the DUT.

  - *axi_master_svt*: Used by the testbench to provide the default register model *bus adapter* implementation, allowing DUT Driver register/memory access (written at the level of abstraction of the DUT register model) to act through the physical 'application bus' interface.
  - *axi_slave_svt:* Used by the testbench to provide the default target for the DUT (DMA) accesses to main memory.

The architecture of the PCIe Test Suite testbenches allows for the replacement of the AXI application bus with a different bus type (for example. AHB, OCP, and so on). If the replacement is AHB (for example) the necessary components are also available in the amba_svt VIP suite. However, if the replacement is some other bus (for example, OCP) it will be necessary to have installed an applicable VIP suite to provide the necessary components.

2.4 DUT Modeled by PCIe VIP Agent

The PCIe VIP Test Suite example test environments use an instance of the pcie_svt VIP product as the DUT. In addition, the various application components (and their associated module instances) are used even when running with RTL DUT. However, in order to model the sequencer-based interface to the DUT in a more realistic way, transactions (sequence items) received from the test environments sequencers targeting the layer components (TL/DL/PL part of pcie_agent) are dispatched through a DUT Driver to the underlying VIP-VIP sequencer instances. These interface with VIP driver components (instead of being passed directly to the underlying VIP sequence item pull ports). This defines a clear boundary as well as clear encapsulation (within the Driver methods) for replacement with DUT-specific Driver implementation. Transactions issued from the application sequencers are able to directly interface with the underlying application drivers irrespective of whether the DUT is modeled by a VIP agent or RTL.

3 Installation and Setup

The Installation and Setup chapter discusses the following topics:

- Product Installation (DESIGNWARE_HOME)
- Downloading Test Suite
- Installing Test Suite
- Licensing Information
- Example Testbench Setup
- Testbench Directory Structure
- Running Test Cases
- Browsing the HTML Class and Testbench Reference

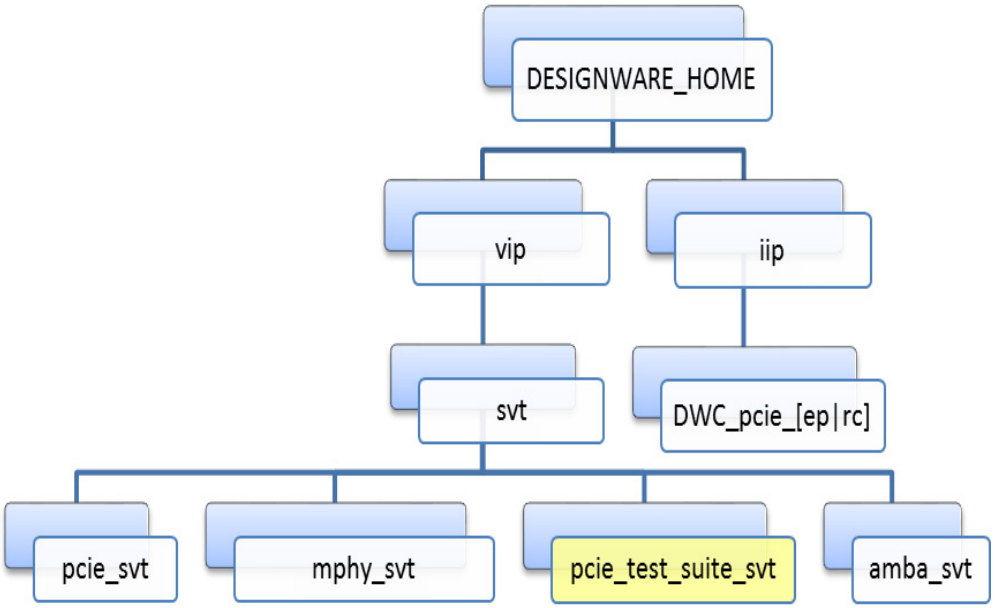
3.1 Product Installation (DESIGNWARE_HOME)

When unpacked from their self-installer (*.run* file) packages, the PCIe VIP Test Suite products are installed into the user-specified `DESIGNWARE_HOME` directory, under the *vip* sub-tree.

Since they are related dependent products the *pcie_svt*, *mphy_svt*, and *amba_svt* VIP products are also included in the self-installer for the Test Suite product (along with other supporting items such as *common* and *svt*). However, the IIP *DWC_pcie Core* is not included in the PCIe VIP Test Suite product delivery: Users of that Core acquire it using a separate license and delivery.

The basic structure of a user's *DESIGNWARE_HOME* tree that includes the PCIe IIP Core (purchased and installed separately), and the PCIe VIP Test Suite product is shown in Figure 3-1.

Figure 3-1 Structure of a DESIGNWARE_HOME tree that includes the PCIe IIP Core



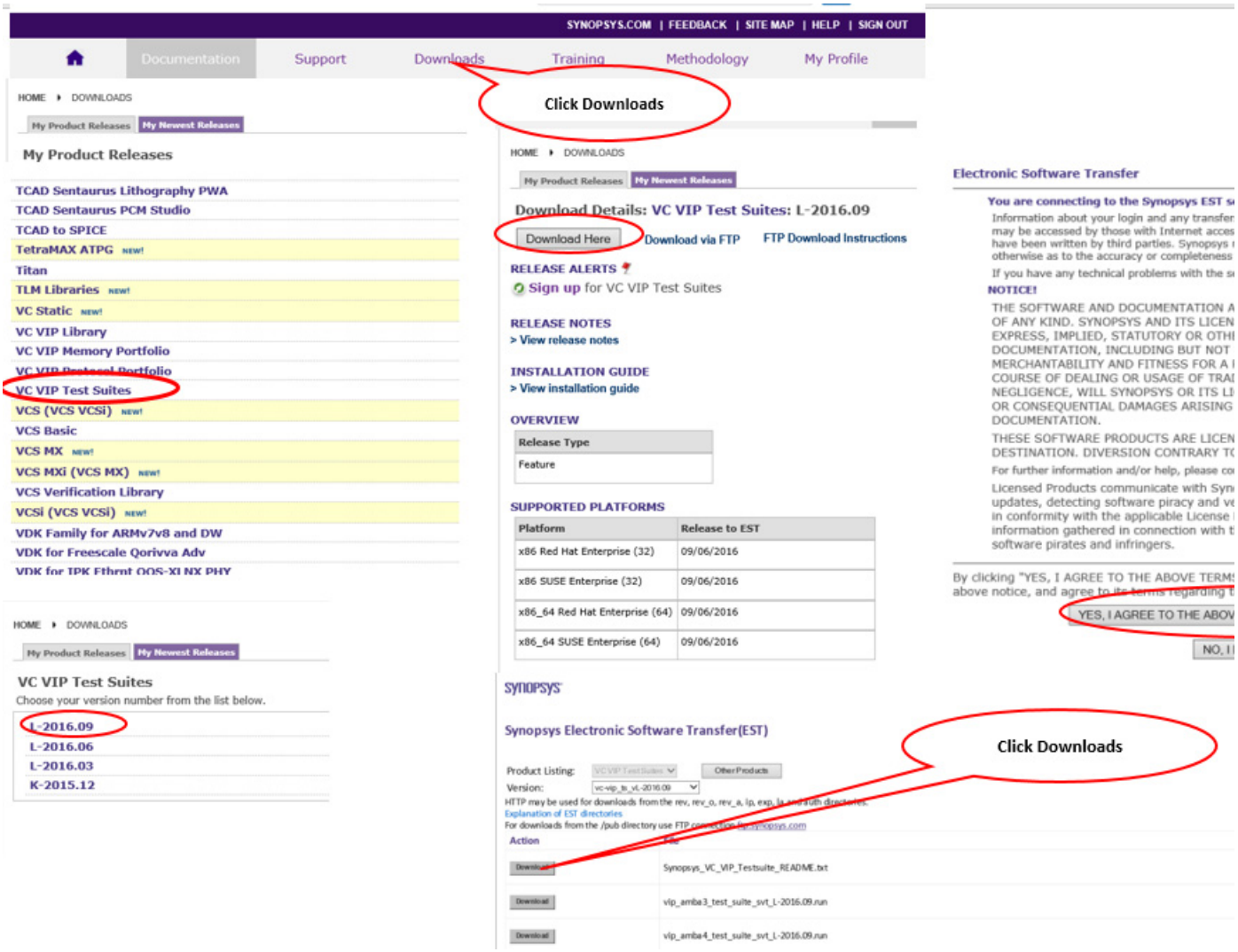
Synopsys tools such as `coreConsultant` and `dw_vip_setup` are used to set up a workspace that may include a configured core and an example testbench as a starting point.

3.2 Downloading Test Suite

The following are the steps to download a Test Suite:

1. Log in to the SolvNetPlus online support site https://solvnetplus.synopsys.com
2. Perform the following selection on SolvNetPlus to download the .run file of the Test suite based upon your license access:
   a. Navigate to the **Downloads** tab.
   b. Go to the Test Suite link from the list of Product Releases.
   c. Select the required <release_version> from the list of release version.
   d. Click the **Download Here** button.
   e. Read the Electronic Software Transfer notice.
   f. If you agree that you have read and understood the notice, and agree to its terms regarding the downloading of the software, then click the **Yes, I Agree to the Above Terms** button. Otherwise, click **No, I do not Agree**.
3. Download the .run file for the test suite.

Figure 3-2 SolvNet Selections for VIP Download



3.3 Installing Test Suite

After the.run file is downloaded, execute the .run file by invoking its filename. The Test Suite is unpacked and all files and directories are installed under the path specified by the DESIGNWARE_HOME environment variable. The .run file can be executed from any directory.

Perform the following steps to install the Test Suite:

1. Set the `DESIGNWARE_HOME` environment variable to the path where you want to install the Test Suite (see Figure 3-1).

   ```
   % setenv DESIGNWARE_HOME VIP_installation_path
   ```

   For example,

   ```
   % setenv DESIGNWARE_HOME /<test_suite>
   % vip_pcie_test_suite_svt_R-2021.03-2.run
   ```

2. The following license checkout request order to authorize Test Suite run file decryption:

Either the Test Suite license or the VIP Library license is required for .run extraction.

- VIP Library license: `VIP-LIBRARY19-SVT`
- Test Suite license:
  - For Gen3 license
    - `VIP-PCIE-GEN3-TESTSUITE-SVT`
  - For Gen4 license
    - `VIP-PCIE-GEN4-TESTSUITE-SVT`
  - For Gen5 license
    - `VIP-PCIE-GEN5-TESTSUITE-SVT`

The following license server variables are considered:

- `DW_LICENSE_FILE`
- `SNPSLMD_LICENSE_FILE`
- `LM_LICENSE_FILE`

If set, the `DW_LICENSE_FILE` overrides `SNPSLMD_LICENSE_FILE` and `LM_LICENSE_FILE`.

If set, the `SNPSLMD_LICENSE_FILE` overrides `LM_LICENSE_FILE`.

For more details on the licenses, see “Environment Variables “in the UVM User Guide.

3. Extracting the .run file.
   a. Agree to the `TEST SUITE TERMS AND CONDITIONS`.

```
Do you agree to these terms and conditions (Yes or No)? Yes
```

   This will unpack the required files in the installation directory.

   b. To verify your acceptance, send the <*vip*>*_test_suite_svt_*<*version*>*_*<*date*>*_ts_terms_conds.acp* file (a new file created in the installation directory) to vip@synopsys.com.
4. For running the example tests, one of the following PCIe VIP licenses is used:
   - VIP Library license: `VIP-LIBRARY19-SVT`
   - VIP license
     - Gen5: `VIP-PCIE-G5-SVT`
     - Gen4: `VIP-PCIE-G4-SVT`
     - Gen3: `VIP-PCIE-G3-SVT`

Note For more details on PCIe VIP licensing, see "Licensing Information" in the *VC Verification IP PCIe UVM User Guide*.

3.4 Licensing Information

The PCIe test suite product licensing is based on the PCIe specification versions.

Table 3-1 Test Suite Licensing Information

License	Speed	Specification version
Gen3	8GT	3.0
Gen4	16GT	4.0
Gen5	32GT	5.0

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

3.4.1 Supported Features

Table 3-2 lists the scope of the license and features supported in the PCIe VIP product.

Table 3-2 Supported Features

License	Supported Features	DUT Type
VIP-PCIE-GEN3-TESTSUITE-SVT	Gen3/Gen2/Gen1 and common speed tests	• EP • RC • PHY
VIP-PCIE-GEN4-TESTSUITE-SVT	• All Gen3 license features and Gen4 speed tests	• EP • RC • PHY • Retimer
VIP-PCIE-GEN5-TESTSUITE-SVT	• All Gen3 license features • All Gen4 license features and Gen5 speed tests with LPC • PIPE4.4.1 and Serial support	• EP • RC • PHY • Retimer

3.5 Example Testbench Setup

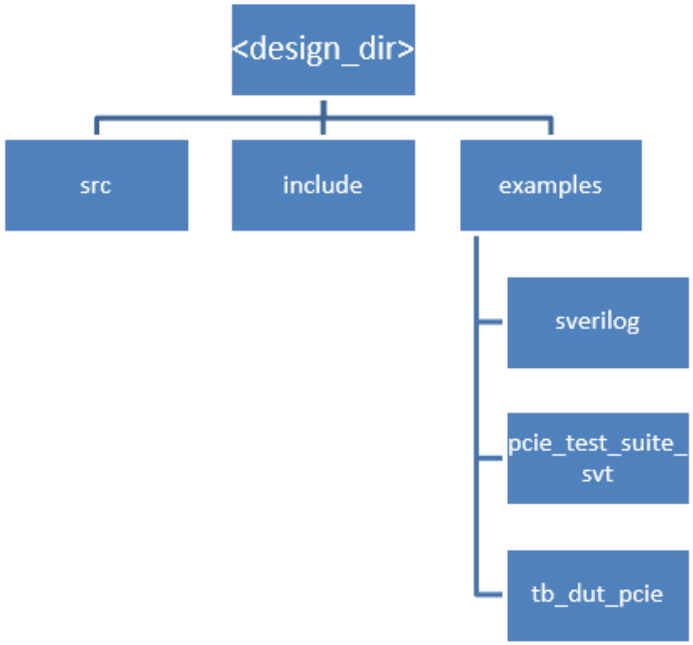
The VIP product setup script is `$DESIGNWARE_HOME/bin/dw_vip_setup`. It is used to set up the example testbench in a specified directory.

For example,

$DESIGNWARE_HOME/bin/dw_vip_setup -p design-dir -svlog pcie_test_suite_svt/tb_dut_pcie

The directory structure produced by `dw_vip_setup` for setting up a *tb_dut_pcie* (Unified) Test Suite testbench, starting at the *design-directory*, is shown in Figure 3-3.

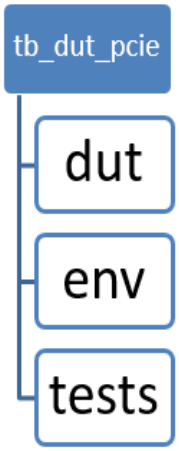
Figure 3-3 Directory Structure Produced by dw_vip_setup



3.6 Testbench Directory Structure

The initial directory structure within the testbench will be as shown in Figure 3-4.

Figure 3-4 Initial Testbench Directory Structure



The general usage of the structure is as follows:

- *tb_dut_pcie*

  Top level of testbench. Includes top level Verilog module definition, Makefile, and scripts supporting the execution of the testbench.

- *dut*

  The DUT RTL is built under this directory. If using the Synopsys DWC_pcie IIP core for the RTL, the coreKit is built into this directory.

Note The example testbench as set up by default includes one file in this directory, a **.tcl** file that is used to configure the example RTL DUT using the Synopsys coreConsultant tool, and a version.txt file specifying the IIP version with which TS is qualified.

- *env*

  All of the files that define elements of the structural UVM verification environment are found in this directory.

- *tests*

  The test case files themselves (files with names of the form *ts.*.sv* are test case files) are found in this directory. These test case files define test classes that extend the `uvm_test` class, set up the configuration, create the test environment, and assign the default sequences to be run for the test case.

Note The test sequences are included in the following directory:
$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/version/sverilog/src/[vcs|mti|ncv]

In future releases, the sequences would be migrated to a *pcie_svt* VIP package.

3.7 Running Test Cases

In each testbench directory there is a file named README that contains basic instructions for running test cases. You can use one of the following two supported approaches:

- Using the `Makefile`.

  In the testbench directory, the provided `Makefile` file is the entry point for running test cases using make (intended to be run with `gmake`).

  For example, the testbench in VIP-VIP mode for link_width=4 and check for Gen1 linkup. Following are the command-line options:

Table 3-3 gmake Command Options

VIP as DUT	Mode	Command
RC	Serial	`gmake demo_pl_gen1_linkup_test-rc SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_rc_vip_serdes.f SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_vip.svi SVT_PCIE_MAX_LINK_WIDTH=4 USE_SIMULATOR=vcsvlog`

Table 3-3 gmake Command Options

VIP as DUT	Mode	Command
RC	PIPE	`gmake demo_pl_gen1_linkup_test-rc SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_rc_vip_pipe.f SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_vip.svi SVT_PCIE_MAX_LINK_WIDTH=4 USE_SIMULATOR=vcsvlog`
EP	Serial	`gmake demo_pl_gen1_linkup_test-ep SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_ep_vip_serdes.f SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_vip.svi SVT_PCIE_MAX_LINK_WIDTH=4 USE_SIMULATOR=vcsvlog`
EP	PIPE	`gmake demo_pl_gen1_linkup_test-ep SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_ep_vip_pipe.f SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_vip.svi SVT_PCIE_MAX_LINK_WIDTH=4 USE_SIMULATOR=vcsvlog`

To enable symbol logging and transaction logging, you can add `SVT_PCIE_ENABLE_TRANSACTION_LOGGING=1` to the above command line.

- Using the `run` script.

  In the testbench directory, the provided *run_dut_ep_** file is a C-shell run script, which allows an alternate method for running test cases.

  For example,

  ```
  ./run_dut_phy demo_pl_gen1_linkup_test  vcsvlog
  ```

3.7.1 Running the Example with +incdir+

In the current setup, you install the VIP under *DESIGNWARE_HOME* followed by creation of a design directory which contains the versioned VIP files. With every newer version of the already installed VIP requires the design directory to be updated. This results in:

- Consumption of additional disk space
- Increased complexity to apply patches

The new alternative approach of directly pulling in all the files from *DESIGNWARE_HOME* eliminates the need for design directory creation. VIP version control is now in the command line invocation.

The following code snippet shows how to run the basic example from a script:

```
cd <testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/

// To run the example using the generated run script with +incdir+

./run_dut_pcie -verbose -incdir demo_dl_link_up_test-ep vcsvlog
```

For example, the following compile log snippet shows the paths and defines set by the new flow to use VIP files right out of *DESIGNWARE_HOME* instead of *design_dir*.

```
vcs -l ./logs/compile.log -q -Mdir=./output/csrc
+define+DESIGNWARE_INCDIR=<DESIGNWARE_HOME> \
+define+SVT_LOADER_UTIL_ENABLE_DWHOME_INCDIRS
+incdir+<DESIGNWARE_HOME>/vip/svt/pcie_test_suite_svt/O-2018.12/sverilog/include \
+incdir+<DESIGNWARE_HOME>/vip/svt/amba_svt/O-2018.12/sverilog/include \
+incdir+<DESIGNWARE_HOME>/vip/svt/mphy_svt/O-2018.12/sverilog/include \
+incdir+<DESIGNWARE_HOME>/vip/svt/pcie_svt/O-2018.12/sverilog/include \
-ntb_opts uvm -full64 -sverilog +define+UVM_DISABLE_AUTO_ITEM_RECORDING -f
svt_pcie_test_suite_dut_compile_file.f \
-y <testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/env/. \
-y <testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/. \
+define+SVT_PCIE_EXCLUDE_MONITOR -debug_acc -timescale=1ns/1fs +libext+.v+.sv -y \
<testbench_dir>/src/verilog/vcs -y <testbench_dir>/src/sverilog/vcs \
-debug_acc -P pli.tab msglog.o +define+SVT_UVM_TECHNOLOGY +define+SYNOPSYS_SV
+incdir+<testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/. \
+incdir+<testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/../../env \
+incdir+<testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/../env \
+incdir+<testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/env \
+incdir+<testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/dut \
+incdir+<testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/hdl_interconne
ct \
+incdir+<testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/lib \
+incdir+<testbench_dir>/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/tests \
-o ./output/simvcssvlog -f top_files -f hdl_files
```

Note For VIPs with dependency, include the `+incdir+` for each dependent VIP.

3.7.1.1 Supported Methodologies with Simulators

Table 3-4 lists the methodologies supported with simulators.

Table 3-4 Supported Methodologies with Simulators

Methodology	VCS	MTI	IUS
UVM	Supported	Supported	Not Supported
OVM	Not supported	Not supported	Not supported
VMM	Not supported	Not supported	Not supported
HDL	Not Supported	Not Supported	Not Supported

3.7.2 Customizing Compile Options and Run Options

The testbench directory also includes the following files, which may be edited to customize the compile-time and runtime switches (`plusargs`) used for the test run.

- *sim_build_options*

  Used to set compile time options for test runs using any supported simulator.

  Example Usage: Define compile time macros that result in scaled down PCIe timer values for faster simulation.

This file specifies `+define+SVT_PCIE_TEST_SUITE_VIP_DUT`, which results in the testbench compiling/running in example mode using a PCIe VIP Agent to model the DUT behavior. Changing this to `+define+SVT_PCIE_TEST_SUITE_RTL_DUT` will cause the testbench to compile/run as if there is an RTL DUT integrated into the DUT wrapper.

- *sim_run_options*

  Used to set runtime options for test runs using any supported simulator.

  Example Usage: Specify tests to run. Specify message verbosity. Set runtime configuration options.

- *vcs_build_options*

  Used to set additional compile time options for tests only if running with VCS as the simulator.

  Example Usage: Specify simulation timescale using simulator specific switch.

- *vcs_run_options*

  Used to set additional runtime options for tests only if running with VCS as the simulator.

  Example Usage: Demote simulator specific warning messages, using simulator specific switch.

3.7.3 Predefined Testbench Options

The Test Suite is also delivered with a set of predefined options to enable important testbench features. Values for these options can either be set as environment variables or they can be provided as command-line arguments when running using the provided *Makefile*. (The `run` script solution does not support command line options at this time).

Alternatively refer to the *README* file in the *tb_dut_dp_*/env* area for details about predefined options and usage information.

For test suite command-line options, see section 6.23 for EP and section 7.23 for RC.

3.8 Browsing the HTML Class and Testbench Reference

Note the following when browsing the HTML Class and Testbench Reference:

- Synopsys recommends using FireFox version 16.0 and above to *directly* open the HTML Class and Testbench pages within a UNIX file system.
- You can also access the HTML Class and Testbench Reference in the following ways.
  - HTML class reference documentation path: *$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/doc/pcie_test_suite_svt_uvm_reference/html/index.html*
  - On Unix: Create a soft link (ln -s) in an internal home page to the location of the HTML Class and Testbench Reference. Using this method, you can use earlier versions of FireFox.
  - On Windows: Create a shortcut in a home page to the location of the HTML Class and Testbench Reference.
- HTML class reference documentation can be used for the following:

- Test suite key features.
- Viewing individual classes and its inheritance hierarchy.
- Viewing all test cases (for EP/RC/PHY DUT) present in tb_dut_pcie.
- Finding members and source code for a particular class easily.

4 Test Suite Testbench Architecture

This testbench architecture chapter discusses the following topics:

4.1 Overview

The arrangement shown in Figure 4-1 highlights the following main building blocks of the testsuite framework.

- Testbench top module (*top.sv*) provides options for user-defined customization.
- Key child instances of `uvm_test_top`: `ep_env` and `rc_env` instances can be disabled at runtime to run tests against a DUT.
- *svt_pcie_test_suite_dut_compile_file.f* simplifies setup and integration of DUT and existing user-defined testbench components.
- Script and text files provide the flexibility of setting up the required testbench at runtime, and provides a scalable (multiple instances of the VIP for multi-port DUT) control mechanism.

Details of this framework are listed in following sections.

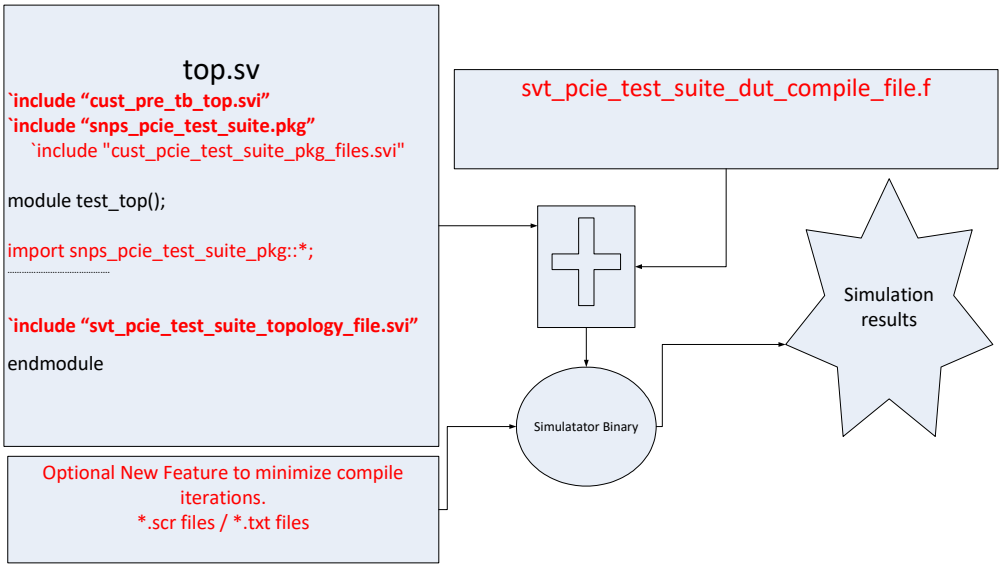
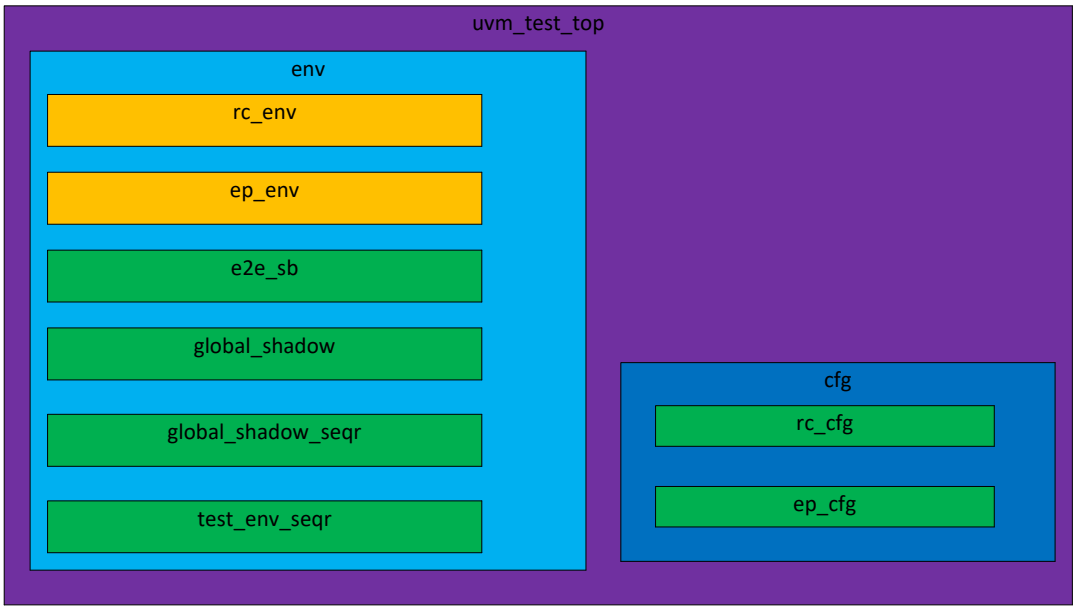


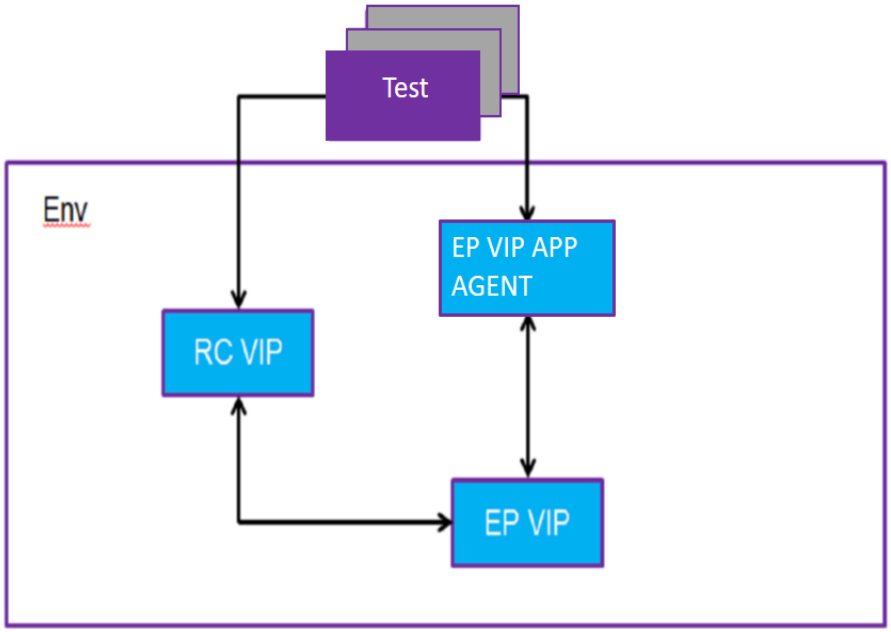
Figure 4-1 General Architecture of the Unified Testbench



4.2 Block Diagram With the VIP as the DUT

The PCIe Test Suite testbenches as delivered run in an demo mode, where the role of the DUT is played by an instance of the PCIe VIP Agent, as represented in Figure 4-2.

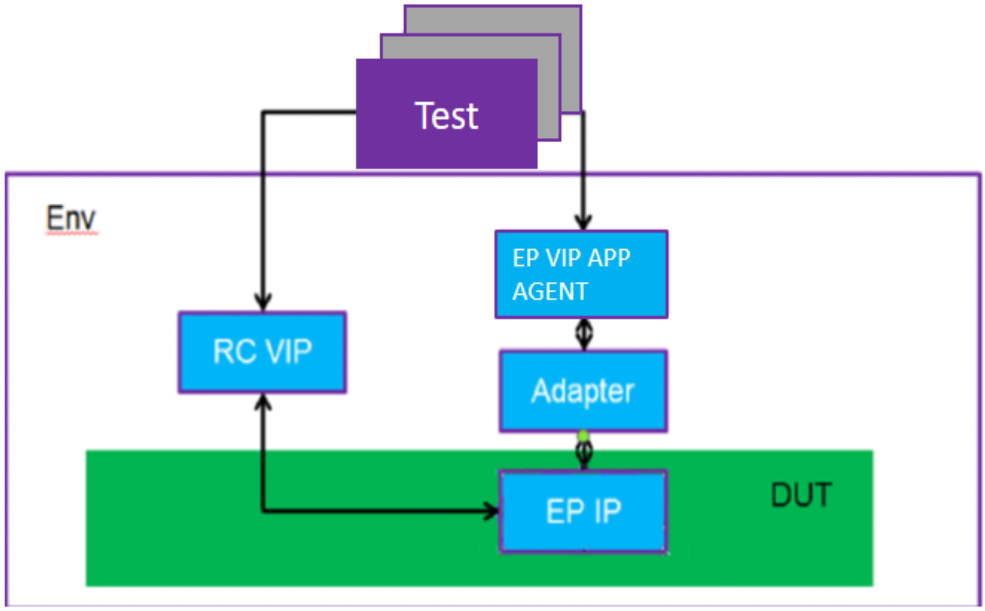
Figure 4-2 Diagram with the VIP as the DUT



4.3 Block Diagram with an RTL DUT

When customized and run for an RTL DUT, the PCIe Test Suite testbenches do not include the PCIe VIP Agent in the DUT side's agent. Only the Application agent containing the various application components are present. In this configuration (see Figure 4-3), it is the job of the DUT-specific Driver logic in the External Application BFM to translate sequence items provided by or used by the test sequences.

Figure 4-3 Diagram with an RTL DUT



4.4 Test Suite Compile and Simulation Framework

4.4.1 Generic Command to Run Simulation

```
gmake USE_SIMULATOR=<vcsvlog> \
<testname: passed to +UVM_TESTNAME> \
SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=<dut_specific_compile_file.f> \
SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=<dut_specific_topology.svi>
SVT_PCIE_TEST_SUITE_PLUSARG_FILE=<plusarg file> \
```

4.4.2 Make Option SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE

The file supplied via this option will be passed to compile with a `-f` option. The file must include compiler directives (`+incdir, -y +libext+,` and so on) required to compile the DUT files.

The default value of the switch `SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE` is *compile_dut_snps_pcie_bifurcated_phy_vip.f*, which can be used to run a demo of the multi-link SERDES PHYtestbench.

4.4.3 Make Option SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE

The file supplied via this option must include the PCIe unified VIP (module and interface) instances, PCIe signal connections between the PCIe unified VIP and DUT instance and the DUT instance.

Default value of switch in *tb_dut_pcie/Makefile* is `SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE` is *topology_dut_snps_pcie_bifurcated_phy_vip.svi*, which can be used to run a demo of the mult-link SERDES PHY Testbench.

The *top.sv* file includes the top-level module named `test_top`. The module file includes a number of other files that is open for customizations such as *cust_pre_tb_top.svi*, *svt_pcie_test_suite_cmd_line_defines.svi* and so on. The module name `test_top` needs to be retained, but not necessarily as the top-level instance.

4.4.4 Make Option SVT_PCIE_TEST_SUITE_PLUSARG_FILE

You can supply the run time switches to simulator using `gmake` switch `SVT_PCIE_TEST_SUITE_PLUSARG_FILE=<user defined file>`. If specified, the Makefile simply invokes the simulator executable with `-f <user specified file>`.

The file supplied via this options must include the runtime (`plusarg`) switches to the simulator. An example of such a file (files with *.scr* extension) can be found inside the *tests* directory of the unified test environment.

Example content of *.scr* file used for testing a SERDES PHY DUT is shown below. Properties of VIP instances representing Endpoint and Root complex are modified using `plusargs`.

```
// Following line has a plusarg does width setting for link 0 by controlling EP and RC
VIP instances. Link 0 operates at x2 and the lane reversal is enabled
+cfg[0]=ep_cfg.pcie_cfg.pl_cfg.link_width:2,rc_cfg.pcie_cfg.pl_cfg.link_width:2,rc_cfg.p
cie_cfg.pl_cfg.lane_reversal_mode:FORCED,ep_cfg.pcie_cfg.pl_cfg.lane_reversal_mode:SUPPO
RTED
// Following line has a plusarg to do width setting for link 1 by controlling EP and RC
VIP instances. Link 1 operates at x4
+cfg[1]=ep_cfg.pcie_cfg.pl_cfg.link_width:4,rc_cfg.pcie_cfg.pl_cfg.link_width:4
```

Hint		
	,	Used to separate one property setting from other in the concatenated list of settings passed via single `plusarg`.
	:	Acts a separator between the configuration class attribute and the value assigned to it.

The format allows the specification of `plusargs` on individual PCIe VIP instances (or links in a multi-link environment). This controllability is unaffected by the depth of the configuration attributes in a hierarchical configuration structure. The example above shows modification of properties of VIP instances for two different (multi-link) link instances, hence two standard Verilog `plusargs(+cfg[0]=…. & +cfg[1]=……)` are used, where index 0 represents link 0 and index 1 represents link 1.

Note You can also pass standard Verilog `plusargs` (not related to configuration attributes) via.*scr* files.

The format specified above is difficult to read when multiple properties are specified on a single line. The unified test environment allows for all the configuration properties to be listed in a single text file and have them all passed via plusarg using `+svt_pcie_test_suite_all_link_prop_file`.

Example content of a *.txt* file listing down configuration properties and their respective values

- run_time_cfg_overrides_rc.txt
  - `cfg[0].rc_cfg.pcie_cfg.pl_cfg.polling_configuration_timeout_ns=80000`
  - `cfg[0].rc_cfg.pcie_cfg.pl_cfg.lane_reversal_mode=FORCED_AND_SUPPORTED`

  The corresponding *.scr* file that processes the text file is shown below. In this text file, you can override all configuration class attributes.

- ts.pcie_some_test-rc.scr

```
// Configuration class name value pair settings to be loaded to test Root
Complex DUT
+svt_pcie_test_suite_all_link_prop_file=run_time_cfg_overrides_rc.txt
```

The plusarg file is parsed during the `build_phase` of `uvm_test_top` instance and its content is used to decide the topology and modes of the class components.

4.4.5 Creating New Tests Using plusarg Files (.scr or .txt)

The following section describes steps to create new tests (stimulus) without adding new classes extended from base tests. It combines an existing UVM test class with a plusarg file (*.txt* file or *.scr* file) to create a new test.

In the following example, the default sequence of an existing test class to exercise EP or RC DUT is changed. This is achieved without doing a recompile of the testbench.

Files of interest are

- tests/ts.pcie_unified_test.sv
- tests/ts.pcie_unified_test-ep.scr
- tests/run_time_cfg_overrides_ep.txt
- tests/ts.pcie_unified_test-rc.scr
- tests/run_time_cfg_overrides_rc.txt

Class `pcie_unified_test` *(tests/ts.pcie_unified_test.sv)* is the starting point in this illustration.

This class acts as top level test class which is extended from `pcie_unified_base_test` which in turn is extended from `uvm_test`.

Using plusarg `+svt_pcie_test_suite_all_link_prop_file` all of the configuration properties listed and initialized in the *.txt* files are used to load the VIP instances used in the test environment.

- *tests/run_time_cfg_overrides_ep.txt* (configuration class settings for testing EP DUT)
- *tests/run_time_cfg_overrides_rc.txt* (configuration class settings for testing RC DUT)

Content of *tests/ts.pcie_unified_test-rc.scr*

```
// Configuration class name=value pair settings to be loaded to test Endpoint DUT
+svt_pcie_test_suite_all_link_prop_file=tests/run_time_cfg_overrides_ep.txt
```

Content of *tests/ts.pcie_unified_test-ep.scr*

```
// Configuration class name=value pair settings to be loaded to test Root Complex DUT
+svt_pcie_test_suite_all_link_prop_file=tests/run_time_cfg_overrides_rc.txt
```

Program the `default_sequence` in *run_time_cfg_overrides_ep.txt* and *run_time_cfg_overrides_rc.txt* files to alter the `default_test_sequence` invoked by `pcie_unified_test` during run phase.

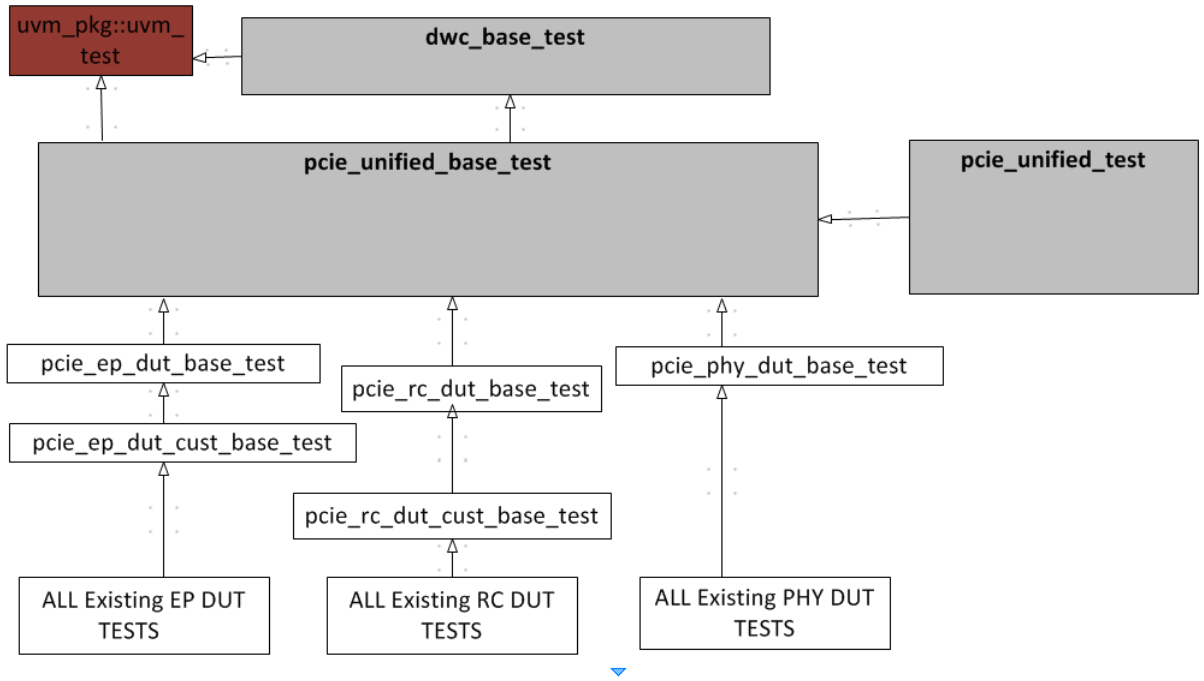
```
cfg[0].default_test_seq=svt_pcie_ts_device_system_virtual_speed_change_from_vip_sequence
```

4.5 TEST and Environment Classes Framework

This section describes how the environment classes are partitioned into base classes and extended classes. The reason behind such a distribution is to reduce duplication of code and align the implementation to leverage UVM factory based (type/instance) overrides done at runtime.

4.5.1 Test Class Framework

Figure 4-4 Test Layer Class Framework



- **`dwc_base_test`**
  - Applicable to all topologies where DUT is SNPS DWC IIP controller. Not applicable to other topologies.
  - Extension of `uvm_test`
  - Contains Implementation of methods applicable to SNPS IIP users. These methods will eventually become part of `pcie_unified_base_test`.
- **`pcie_unified_base_test`**
  - Applicable to all topologies
  - Extension of `uvm_test` or (`dwc_base_test` if DUT is DWC PCIe IIP Controller)
  - Extracts topology/compile time settings from Unified interface instances.
  - Optionally loads name value pairs of configuration attributes from text files.
  - Builds Instance of configuration object for one or more links.
  - In case of multi-link, setup creates additional instances of extensions of `pcie_unified_base_test`.
  - When you migrate from the existing deployments of `tb_dut_phy` and `tb_dut_ep_*` and `tb_dut_rc_*` examples to `tb_dut_pcie` area this class will replace `pcie_phy_dut_base_test`, `pcie_ep_dut_base_test`, `pcie_ep_dut_cust_base_test`, `pcie_rc_dut_base_test`, and `pcie_rc_dut_cust_base_test`.
- **`pcie_phy_dut_base_test`**

- Applicable to all topologies where DUT is a Phy (Single link/Multi-link)
- Setting DUT capabilities
- `cfg` object creation
  - env overrides are accomplished
  - processing of certain plusargs
  - Builds Instance of `snps_pcie_link_env` unless `svt_pcie_test_suite_configuration::enable_integration_in_user_environment` is set to 1.
  - Demotes Errors
  - Registers Callbacks
  - Processes plusargs using `$test$plusargs` and `$value$plusargs` (legacy approach)
  - Processes plusargs using SVT hook `set_prop_val_via_plusargs`. For more details, see Make Option SVT_PCIE_TEST_SUITE_PLUSARG_FILE.

- **`pcie_ep_dut_base_test`** and **`pcie_ep_dut_cust_base_test`**
  - Applicable to all topologies where DUT is a EP controller (EP only controller or a Dual Mode controller configured at compile time to operate in EP mode)
  - Class code is shared with existing `tb_dut_ep_*` environment with few modifications around following areas
    - `cfg` object creation
    - env overrides are accomplished
    - processing of certain plusargs
  - Builds Instance of `snps_pcie_link_env` unless `svt_pcie_test_suite_configuration::enable_integration_in_user_environment` is set to 1.
  - Demotes Errors
  - Registers Callbacks
  - Processes plusargs using `$test$plusargs` and `$value$plusargs` (legacy approach)
  - Processes plusargs using SVT hook `set_prop_val_via_plusargs`. For more details, see Make Option SVT_PCIE_TEST_SUITE_PLUSARG_FILE.

- **`pcie_rc_dut_base_test`** and **`pcie_rc_dut_cust_base_test`**
  - Applicable to all topologies where DUT is a RC controller (RC only controller or a Dual Mode controller configured at compile time to operate in RC mode)
  - Class code is shared with existing `tb_dut_rc_*` environment with few modifications around following areas
    - `cfg` object creation
    - env overrides are accomplished
    - processing of certain plusargs
  - Builds Instance of `snps_pcie_link_env` unless `svt_pcie_test_suite_configuration::enable_integration_in_user_environment` is set to 1.
  - Demotes Errors

- Registers Callbacks
- Processes plusargs using `$test$plusargs` and `$value$plusargs` (legacy approach)
- Processes plusargs using SVT hook `set_prop_val_via_plusargs`. For more details, see Make Option SVT_PCIE_TEST_SUITE_PLUSARG_FILE.

❖ **`pcie_unified_test`**

- It is a single test class not serving as base class for any of the existing tests
- It demonstrates a new approach of writing tests that can be applied to topologies where DUT is
  - Dual mode controller with RC/EP role decided at Run time (0ns during uvm_build_phase)
  - EP only controller
  - RC only controller
- Builds Instance of `snps_pcie_link_env` unless `svt_pcie_test_suite_configuration::enable_integration_in_user_environment` is set to 1.
- Processes plusargs using SVT hook `set_prop_val_via_plusargs`. For more details, see mechanics of `SVT_PCIE_TEST_SUITE_PLUSARG_FILE`.

4.5.2 Link Environment Class Framework

❖ **`snps_pcie_link_env`**

- Applicable to all DUT topologies.
- Is functional equivalent of existing class `snps_pcie_test_env` which is used by `tb_dut_phy`, `tb_dut_ep_*`, `tb_dut_rc_*`.
- Consists of End to End Data Scoreboard Instance and its connections to sub-env components
- Consists of two instances of `snps_pcie_device_env` (`rc_env` and `ep_env`). These instances are overridden with appropriate classes based on DUT type by invoking `pcie_unified_base_test::set_env_overrides();`
- Supports compile time override via macro `` `SVT_PCIE_TEST_SUITE_TEST_ENV ``. This allows to add additional components for customization of test suite.

Table 4-1 Classes Overriding ep_env and rc_env.

Type of DUT	ep_env Instance Override Type	rc_env Instance Override Type
VIP Controller	snps_pcie_dm_vip_env	snps_pcie_dm_vip_env
Phy	snps_pcie_dm_vip_env	snps_pcie_dm_vip_env
DWC PCIe IIP EP* only OR DWC PCIe IIP DM as EP	dwc_pcie_dm_dut_env	snps_pcie_dm_vip_env
DWC PCIe IIP RC only DWC PCIe IIP DM as RC	snps_pcie_dm_vip_env	dwc_pcie_dm_dut_env

Table 4-1 Classes Overriding ep_env and rc_env.

Type of DUT	ep_env Instance Override Type	rc_env Instance Override Type
Non DWC PCIe IIP EP only OR Non DWC PCIe IIP DM as EP	snps_pcie_dm_dut_env but should be changed by customer	snps_pcie_dm_vip_env
Non DWC PCIe IIP RC only OR Non DWC PCIe IIP DM as RC	snps_pcie_dm_vip_env	snps_pcie_dm_dut_env but should be changed by customer
* EP: End Point, RC: Root Complex, DM: Dual Mode core		

4.5.3 Environment (snps_pcie_device_env) Class Framework

Figure 4-5 Snps_pcie_link_env

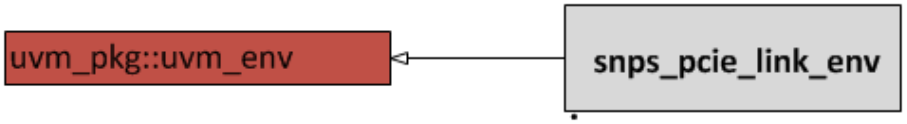
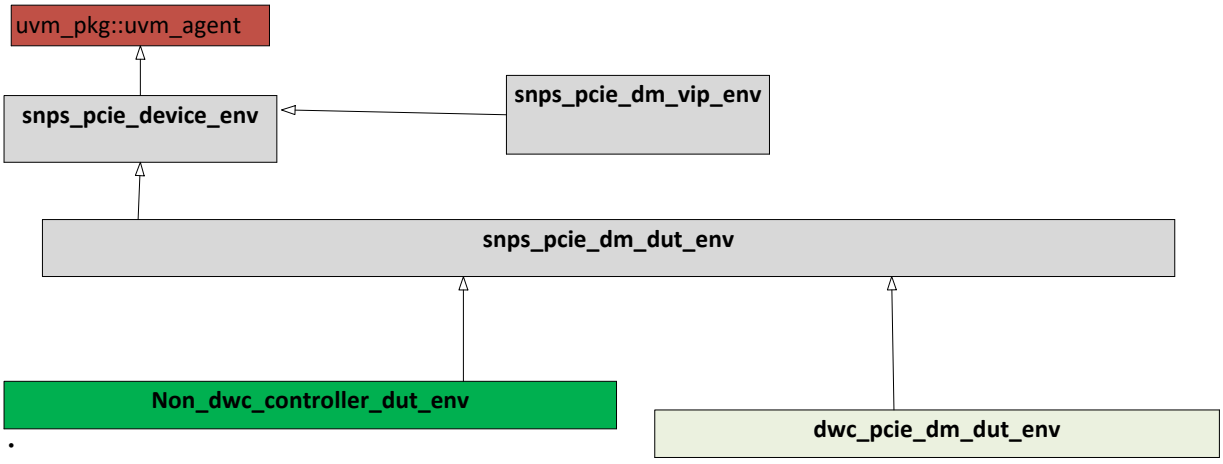


Figure 4-6 Snps_pcie_device_env



- Applicable to all topologies of environment.
- Instances must be overridden with derived classes at runtime.
- Creates component/object instances that are common to a single link partner irrespective of role (RC versus EP) and implementation type (VIP versus IIP). Example child components are VIP, `dut_status`, `tl_event_pool`, `dl_event_pool`, and so on. While creating sequences these components can be safely referenced irrespective of the type (VIP/IIP/SNPS IIP) of DUT.
- Is extended as `snps_pcie_dm_vip_env` and `snps_pcie_dm_dut_env`. For more details on the use model, see the descriptions of the extended versions.

- `snps_pcie_dm_vip_env`
  - Applicable to all topologies of environment except the mode where 2 IIP controllers are used to form a single PCIe link.

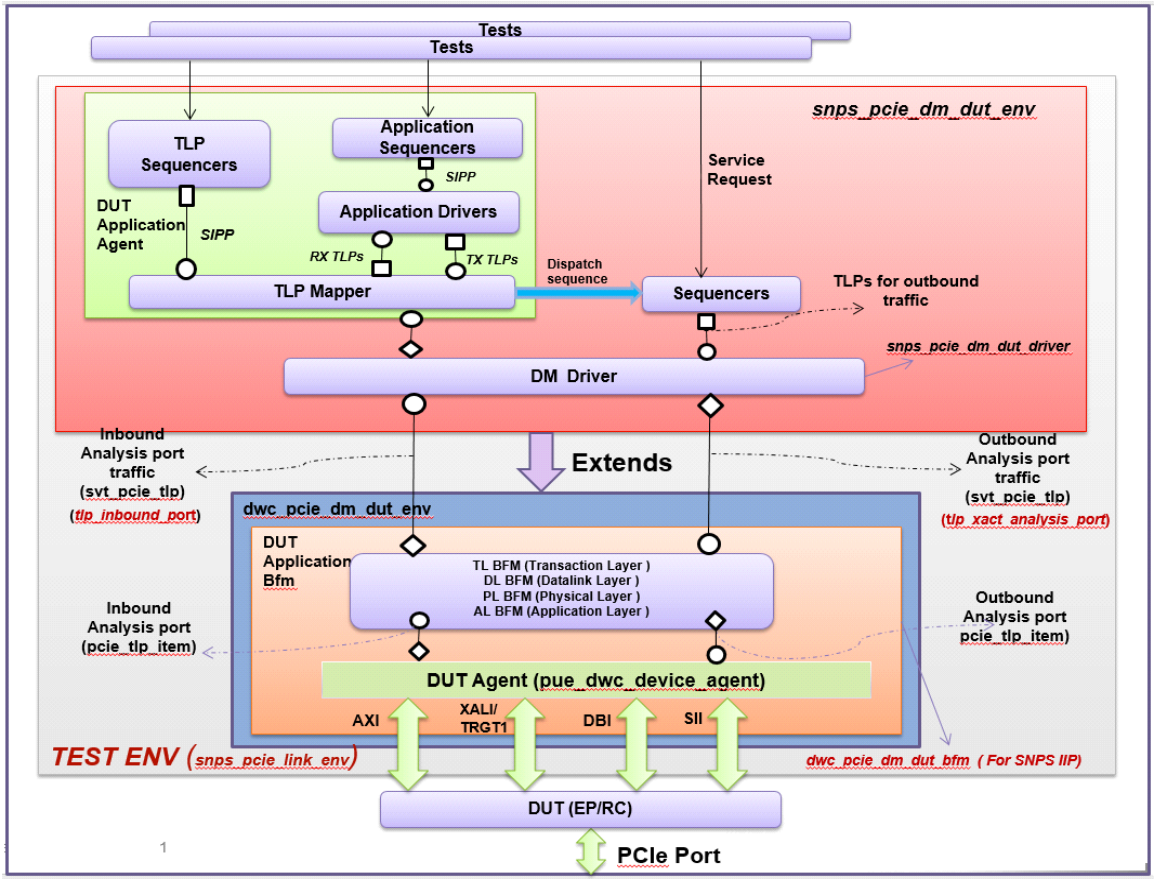
This topology is not yet supported.

  - Contains instance of ACTIVE VIP AGENT (all layers APP, TL, DL, PL)
  - RC versus EP mode selection is done at runtime.
  - Is pure VIP ENV irrespective of DUT implementation type
- **`snps_pcie_dm_dut_env`**
  - Must be extended and the extended type should be used to do instance overrides.
  - Applicable to all topologies where DUT is an IIP controller. Does not apply for PHY DUT or VIP-to-VIP mode.
  - Contains instance of ACTIVE VIP AGENT (only APP layer).
  - Contains instance of `snps_pcie_dm_dut_driver`
  - Contains connections between `snps_pcie_dm_dut_driver` and Active VIP instance APP layer
  - RC versus EP mode selection is done at runtime.
  - Instance of this class is overridden with extended class `dwc_pcie_dm_dut_env` when DUT is DWC Controller IIP.
  - Instance of this class is created for test purpose only when DUT is Non DWC Controller IIP. For more details, see `non_dwc_controller_dut_env`.
- **`snps_pcie_dm_dut_driver`**
  - Applicable to all topologies where DUT is an IIP controller.
  - Does not apply for PHY DUT or VIP-to-VIP mode.
  - Contains threads to pull items from pull ports and pass them to analysis ports.
  - It is used when DUT is Controller IIP (SNPS or 3rd Party)
  - The following TLM ports are used by `snps_pcie_dm_dut_driver` class to interconnect to the Application BFM.
    - Outbound TLP Analysis port: `uvm_analysis_port#(svt_pcie_tlp) tlp_xact_analysis_port`
    - Inbound TLP Analysis port: `uvm_analysis_export #(svt_pcie_tlp) tlp_inbound_port`
    - `uvm_seq_item_pull_port#(svt_pcie_cfg_database_service) cfg_database_service_port;`
    - `uvm_seq_item_pull_port#(svt_pcie_tl_service) tl_service_port;`
    - `uvm_seq_item_pull_port#(svt_pcie_dl_service) dl_service_port;`
    - `uvm_seq_item_pull_port#(svt_pcie_pl_service) pl_service_port;`
  - In the current release Outbound TLP Analysis port and Inbound TLP Analysis port are mandatory for TLP traffic flow, going forward other TLM ports will also be mandatory for the tests and sequences to function as desired.

- **dwc_pcie_dm_dut_env**
  - Applicable to all topologies where DUT is a SNPS IIP controller.
  - Does not apply for PHY DUT or VIP-to-VIP mode.
  - Contains instance of `dwc_pcie_dm_dut_bfm` to drive NON-PCIe interfaces of DUT
  - This instance must always be named as `external_app_bfm`
  - Contains connections between driver instance and BFM instance ports
  - Contains connection between `dut_status` object within the BFM and the one used by environment.

Figure 4-7 provides the top-level details of overall hook up of PCIe application layer agent with the DWC IIP controller Bus Functional Models.

Figure 4-7 Overview of DWC Controller Environment Framework

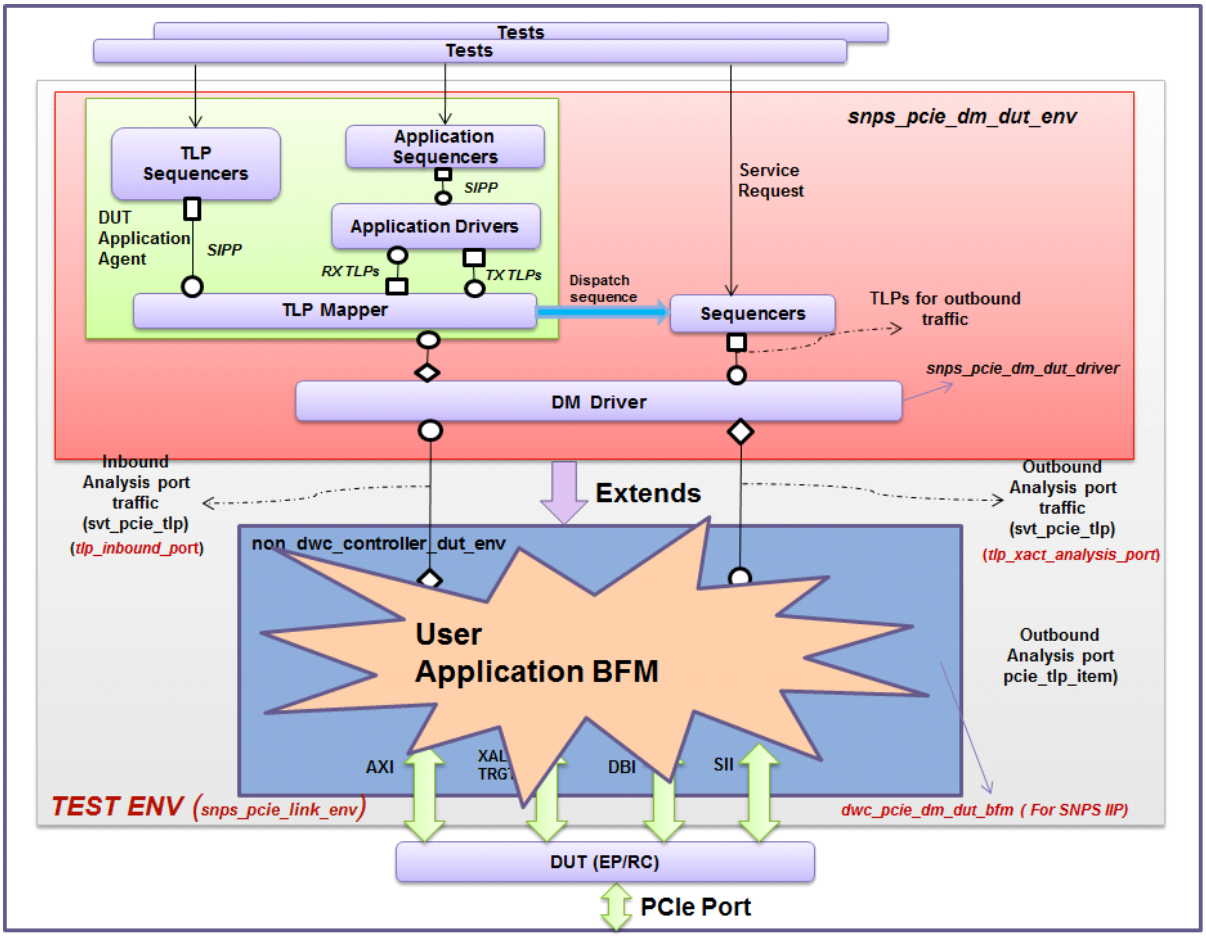


- **non_dwc_controller_dut_env**
  - Applicable to all topologies where DUT is NON-SNPS IIP controller.
  - Not applicable for PHY DUT or VIP-to-VIP mode.
  - Must be used while trying to verify a Non-DWC PCIe Controller DUT.

    For implementation details, see `dwc_pcie_dm_dut_env`.

Figure 4-8 provides the top-level details of overall hook up of PCIe application layer agent with the NON-SNPS IIP controller Bus Functional Models. It is important that the User Application BFM is appropriately hooked up in this environment.

Figure 4-8 Overview of Non-DWC Controller Environment Framework



4.6 Sequence Grouping

The sequences in the PCIe VIP Test Suite are logically grouped based on protocol layer, for easy identification. Table 4-2 shows the groupings and corresponding filenames.

The files containing sequences and callback classes are present in the *$(design_dir)/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie/seq_and_cb* directory in the product installation tree. For details about sequences used in the test, see the PCIe Test Suite HTML class reference (accessed using the TestCases tab in the specific testbench area).

Table 4-2 Sequence Collection Groupings

Protocol Layer - Test Group	Sequence Collection Filename
System Level primitive sequences.	svt_pcie_ts_device_system_virtual_sequence_collection.sv
TL - TLP Cpl Timeout, TLP Data Poison, TLP ECRC	svt_pcie_ts_device_system_virtual_tl_cpl_timeout_data_poison_ecrc_sequence_collection.sv
TL - TLP Error Msg, TLP Ordering, TLP TC_VC, TLP VC	svt_pcie_ts_device_system_virtual_tl_err_msg_ordering_tc_vc_collection.sv
TL - TLP Flow Control	svt_pcie_ts_device_system_virtual_tl_flow_control_common_sequence_collection.sv
TL - TLP Flow Control	svt_pcie_ts_device_system_virtual_tl_flow_control_txrx_sequence_collection.sv
TL - TLP Transactions	svt_pcie_ts_device_system_virtual_tl_tlp_transactions_sequence_collection.sv
TL - TLP Transactions	svt_pcie_ts_device_system_virtual_tl_tlp_rules_sequence_collection.sv
TL - TLP Transactions	svt_pcie_ts_device_system_virtual_tl_be_td_cpl_rxtlp_sequence_collection.sv
TL - TLP Transactions	svt_pcie_ts_device_system_virtual_tl_request_handling_data_returned_sequence_collection.sv
PL – LTSSM (All Polling)	svt_pcie_ts_device_system_virtual_pl_ltssm_polling_sequence_collection.sv
PL – LTSSM (All Configuration)	svt_pcie_ts_device_system_virtual_pl_ltssm_configuration_sequence_collection.sv
PL – LTSSM (All Recovery)	svt_pcie_ts_device_system_virtual_pl_ltssm_recovery_sequence_collection.sv
PL – Equalization LTSSM	svt_pcie_ts_device_system_virtual_pl_ltssm_equalization_sequence_collection.sv
PL – Framing	svt_pcie_ts_device_system_virtual_pl_framing_error_sequence_collection.sv
TL - SRIOV	svt_pcie_ts_device_system_virtual_tl_sriov_sequence_collection.sv
TL - ATS	svt_pcie_ts_device_system_virtual_tl_ats_sequence_collection.sv
TL-RN	svt_pcie_ts_device_system_virtual_tl_readiness_notification_sequence_collection.sv
PL-LowPower	svt_pcie_ts_device_system_virtual_pl_ltssm_lowpower_sequence_collection.sv
Config Space	svt_pcie_ts_device_system_virtual_config_space_sequence_collection.sv
DL	svt_pcie_ts_device_system_virtual_dl_sequence_collection.sv
PL Encoding	svt_pcie_ts_device_system_virtual_pl_encoding_sequence_collection.sv
PL LTSSM	svt_pcie_ts_device_system_virtual_pl_ltssm_sequence_collection.sv
RX Margining	svt_pcie_ts_device_system_virtual_rx_margining_sequence_collection.sv
TL Config Space	svt_pcie_ts_device_system_virtual_tl_config_space_sequence_collection.sv
TL Transaction Ordering	svt_pcie_ts_device_system_virtual_tl_transaction_ordering_sequence_collection.sv
DL - Callback collection	svt_pcie_ts_dl_callback_collection.sv
PL - Callback collection	svt_pcie_ts_pl_callback_collection.sv
TL - Callback collection	svt_pcie_ts_tl_callback_collection.sv
TL - Driver App Callback collection	svt_pcie_ts_driver_app_callback_collection.sv

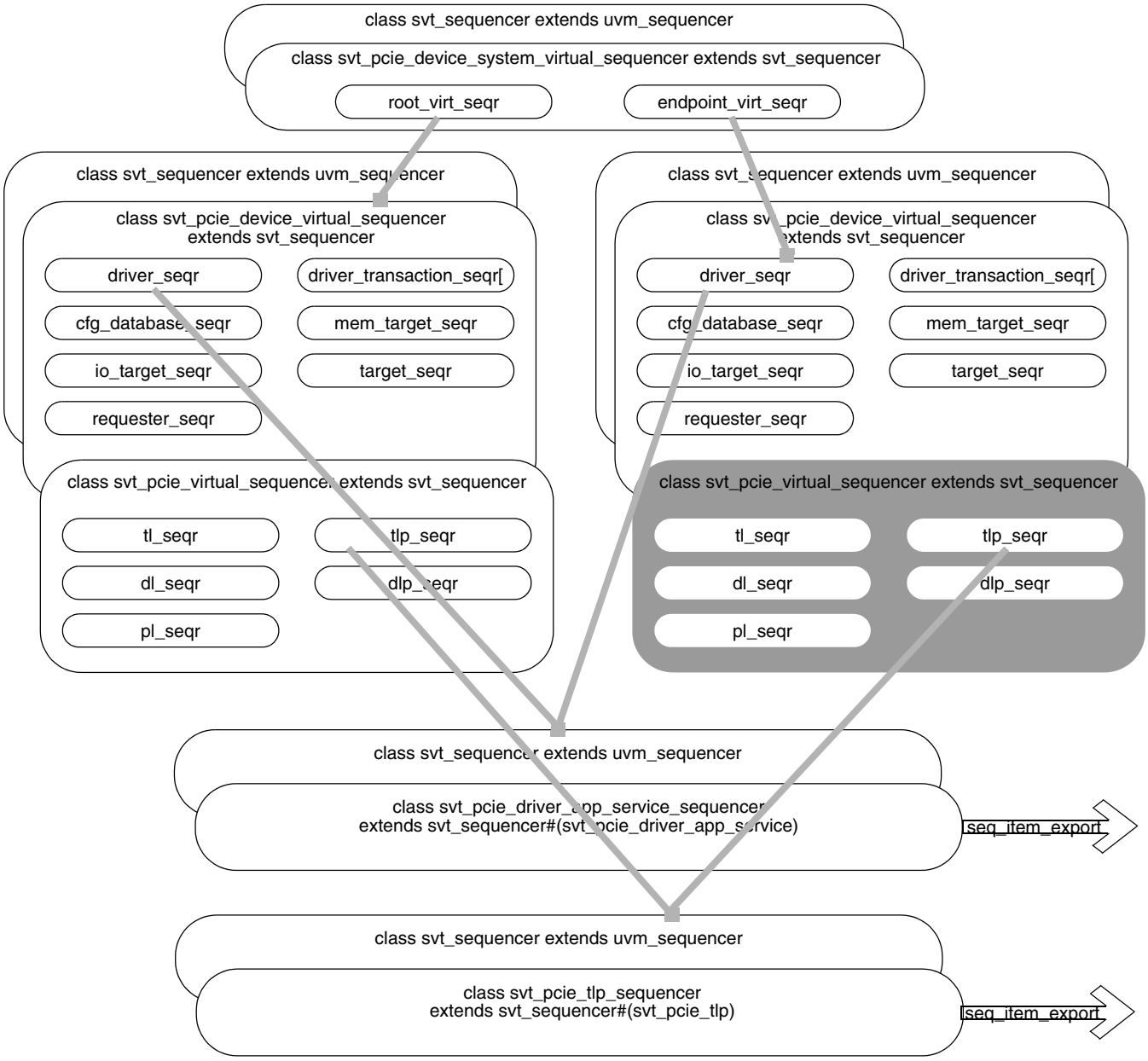
Table 4-2 Sequence Collection Groupings (Continued)

TL - Target App Callback collection	svt_pcie_ts_target_app_callback_collection.sv
PL - Gen5 sequence collection	svt_pcie_ts_device_system_virtual_gen5_sequence_collection.sv
PL - Retimer sequence collection	svt_pcie_ts_device_system_virtual_retimer_sequence_collection.sv
TL - PTM sequence collection	svt_pcie_ts_device_system_virtual_tl_ptm_sequence_collection.sv
TL - User enumeration sequence collection	svt_pcie_ts_device_system_virtual_user_enumeration_sequence_collection.sv

4.7 Common Sequencer Hierarchy

The UVM sequencers that are part of the common architecture of the testbench (whether RTL or a VIP Agent is acting as the DUT) are organized in a hierarchy, as shown in Figure 4-9.

Figure 4-9 Hierarchy of UVM Sequencers that are Part of the Testbench



For the sake of simplicity, all non-virtual sequencer types are not shown in Figure 4-9. It should be understood that each non-virtual sequencer provides sequence items of the class implied by its name.

4.7.1 Sequencer TLM Port Connections

On the VIP side of the test environment, the `seq_item_export` TLM ports are connected to the Sequence Item Pull Ports (SIPPs) implemented by the layer-specific sub-blocks of the `pcie_svt` VIP agent (`svt_pcie_device_agent`).

On the DUT side of the test environment, the application sequencers' `seq_item_export` TLM ports are connected to corresponding Sequence Item Pull Ports (SIPPs) implemented by the Application Drivers in the Application agent. This is true whether the DUT is modeled by actual RTL or another PCIe VIP Agent.

In addition, when the DUT is an actual RTL, the `tlp_sequencer` is also connected to the TLP Mapper component inside the Application Agent via another SIPP. This covers SIPP implementations for the following `pcie_svt` transactions types:

- svt_pcie_driver_app_transaction
- svt_pcie_tlp
- svt_pcie_driver_app_service
- svt_pcie_requester_app_service
- svt_pcie_target_app_service
- svt_pcie_mem_target_service
- svt_pcie_io_target_service
- svt_pcie_global_shadow_service

The DUT Driver base class implements SIPPs for the following `pcie_svt` transaction types:

- svt_pcie_cfg_database_service
- svt_pcie_tl_service
- svt_pcie_tlp
- svt_pcie_dl_service
- svt_pcie_dllp
- svt_pcie_pl_service

These SIPPs are connected to the `seq_item_export` TLM ports of the corresponding sequencers (even when the DUT is being modeled by another PCIe VIP Agent). It is then the Driver's job to dispatch the received sequence items, either to the External Application BFM (when the DUT is actual RTL), or by forwarding them to the underlying PCIe VIP Agent (when the DUT is modeled by another PCIe VIP Agent).

Note that the only source of `svt_pcie_tlp` sequence items received by the DUT driver is the Application Agent (when the DUT is actual RTL). When the DUT is modeled by another PCIe VIP Agent, the tests directly interface with this `tlp_sequencer`.

4.8 External Application BFM (SNPS IIP Core)

External Application BFM is a placeholder for most of the user-specific code in the default use model of PCIe Test Suite. This helps to isolate your specific implementation from the Test Suite code. This section includes the details of major parts of the implementation of an External Application BFM which the Test Suite provides to act as a reference. It is designed to integrate the Synopsys `DWC_pcie` core with PCIe Test Suite.

External Application BFM is a wrapper around controller that serves the following purpose:

- Process incoming request and handles outgoing request.
- In case of AXI interface, convert PCI TLP to AXI and AXI to TLP for completion.
- Drive transaction to controller interface.
- Process service request by driving signals from the testbench.
- Program controller registers.

You must design your Application BFM for Non-SNPS core.

4.8.1 Application BFM Mode Support

Application BFM supports the following two interface modes.

- Native Interface

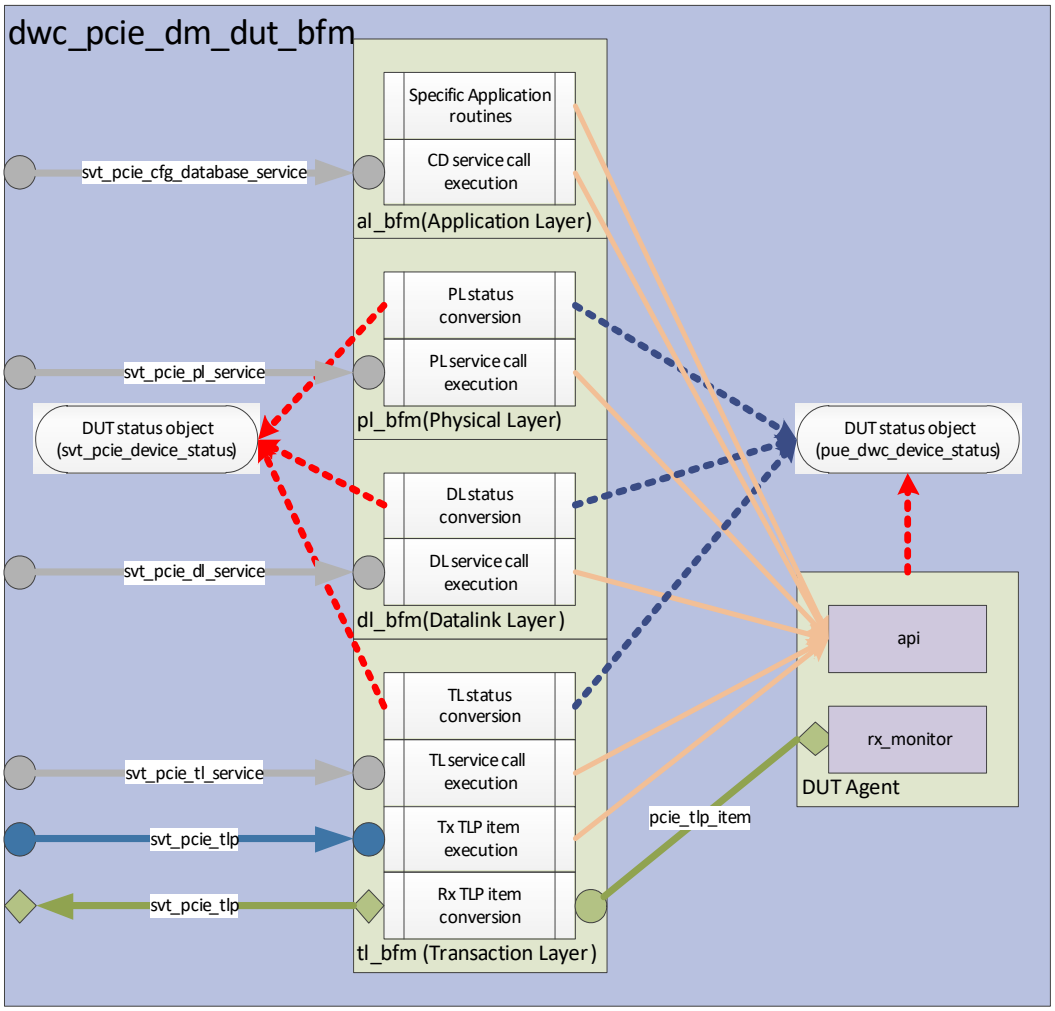
  This interface mode supports DUT in both SERDES and PIPE mode. This mode is specific to SNPS IIP only.

- AXI Interface (here SNPS AXI VIP is used)

  This interface mode supports DUT in both SERDES and PIPE mode.

4.8.2 Application BFM Structure

Figure 4-10 Application BFM Structure



Application BFM consists of four BFMs as described in Figure 4-10.

- `ts_dut_tl_bfm`
  - Used for TLP conversion from AXI to PCIE and PCIE to AXI in case of AMBA interface.
  - Executes service calls.
- `ts_dut_dl_bfm`
  - Used for updating status
  - Executes service calls.
- `ts_dut_pl_bfm`
  - Used for LTSSM states.
  - Executes service calls.
- `ts_dut_al_bfm`
  - Executes service calls.

4.8.3 Analysis Port

- `rx_tlp_analysis_port`

  Implemented to accept transactions of type `svt_pcie_tlp`.

- `cd_service_response_analysis_port`

  Implemented to accept transactions of type `svt_pcie_cfg_database_service`.

4.8.4 Analysis Export

Analysis port for the outbound TLPs (requests/completions) transmitted by the DUT. In the environment, it is connected to the driver. Optionally, this port can be used for connecting to an external Scoreboard

- `tx_tlp_analysis_export`

4.8.5 Service Port

- `tl_service_analysis_export`

  This is implemented to pass transactions of type `svt_pcie_tl_service`. This API is provided for future enhancements in Test Suite which may involve sending TL service transactions.

- `dl_service_analysis_export`

  This is implemented to pass transactions of type `svt_pcie_pl_service`. This API is provided for future enhancements in Test Suite which may involve sending DL service transactions.

- `pl_service_analysis_export`

  This is implemented to pass transactions of type `svt_pcie_pl_service`. This API is provided for future enhancements in Test Suite which may involve sending PL service transactions.

- `cd_service_analysis_export`

  This is implemented to drive response transactions of type `svt_pcie_cfg_database_service` populated with DUT's response to the backdoor configuration accesses.

4.9 Scoreboard

PCIe Test Suite Scoreboard has a uniform scoreboarding architecture for [VIP - VIP/VIP - DUT] modes, which compares:

- Inbound TLP (Received by DUT EP) with TLP VIP has transmitted.
- Outbound TLP (Transmitted by DUT EP) with TLP VIP has received.

The Test Suite environment provides analysis ports at an external Application BFM and DL layer. The RC VIP provides the write and response data.

Note the following:

- Address based transaction are stored into associative array with addresses as index.
- ID based transaction are stored in to associative array with Requester ID as index.

- Run phase `uvm_post_shutdown_phase` waits for associative arrays to be empty.

Note MISCMP messages coming from `UVM compare()` method by default are set to `UVM_INFO`. To change the severity to `UVM_WARNING`, set the `enable_sb_miscmp_severity_warning` test suite configuration attribute to 1.

Test suite configuration provides some controlling points. By using those comparison they can be controlled. Table 4-3 shows the controlling points.

Table 4-3 Fields of TLP in Test Suite Configuration

Field Name	Type	Default	Description
enable_scoreboard	bit	1'b1 => enable	Enable Scoreboard.
enable_sb_miscmp_severity_warning	bit	1'b0 => disable	Configuration to change the severity of MISCMP messages of `UVM compare()` method from `UVM_INFO` to `UVM_WARNING`.
enable_sb_dut_tx_path	bit	1'b1 => enable	Enable Scoreboard in DUT Tx path.
enable_sb_dut_rx_path	bit	1'b1 => enable	Enable Scoreboard in DUT Rx path.
enable_sb_dut_tx_cpl_compare	bit	1'b1 => enable	Configuration to control comparison of completion TLPs generated by DUT.
enable_sb_dut_rx_cpl_compare	bit	1'b1 => enable	Configuration to control comparison of completion TLPs generated by VIP.
enable_sb_dut_tx_cfg_cpl_compare	bit	1'b0 => disable.	Configuration to control comparison of completion TLPs in response of Cfg transactions, generated by DUT.
enable_sb_dut_rx_cpl_no_data_compare	bit	1'b0 => disable.	Configuration to control comparison of received completion with no data TLPs
enable_sb_dut_tx_err_msg_compare	bit	1'b0 => disable.	Configuration to control comparison of error messages transmitted by DUT
check_sb_dut_tx_err_msg	bit	1'b1 => enable.	Configuration control to enable/disable protocol check for unexpected error messages transmitted by DUT.
enable_sb_dut_tx_tag_compare	bit	1'b0 => disable.	Configuration to control comparison of tag field generated by DUT. (TLP Header field)
enable_sb_dut_tx_rid_compare	bit	1'b1 => enable	Configuration to control comparison of requester id field generated by DUT. (TLP Header field)
enable_sb_dut_tx_cid_compare	bit	1'b1 => enable.	Configuration to control comparison of requester id field generated by DUT. (TLP Header field)
enable_sb_dut_rx_td_compare	bit	1'b0 => disable.	Configuration to enable comparison of TLP Digest field, which is generated by VIP. (TLP Header field)

Table 4-3 Fields of TLP in Test Suite Configuration (Continued)

Field Name	Type	Default	Description
enable_sb_dut_rx_ep_compare	bit	1'b0 => disable.	Configuration to enable comparison of TLP Error Poisoning field, which is generated by VIP. (TLP Header field)
enable_sb_dut_tx_ep_compare	bit	1'b0 => disable.	Configuration to enable comparison of TLP Error Poisoning field, which is generated by DUT. (TLP Header field)
enable_sb_dut_rx_attr_id_order_compare	bit	1'b0 => disable.	Configuration to enable comparison of TLP ID Based ordering attribute field, which is generated by VIP. (TLP Header field)
enable_sb_dut_tx_byte_enable_compare	bit	1'b0 => disable.	Configuration to enable comparison of byte_enable field, which are generated by VIP. (TLP Header field). It's also controlling byte_count field.
enable_sb_dut_rx_reserved_bit_compare	bit	1'b0 => disable.	Configuration to enable comparison of reserved field, which are generated by VIP. (TLP Header field)
enable_sb_dut_rx_byte_count_compare	bit	1'b0 => disable	Configuration to enable comparison of TLP reserved field, which is generated by VIP
enable_sb_dut_rx_lower_address_compare	bit	1'b0 => disable	Configuration to enable comparison of TLP reserved field, which is generated by VIP
enable_sb_dut_rx_cid_compare	bit	1'b0 => disable	Configuration to enable comparison of TLP reserved field, which is generated by VIP

Because of the current VIP-IIP AXI based interface, some of fields cannot populate at the Application BFM side. Therefore, comparison of the following fields are disabled. The remaining fields of TLPs are compared.

- Inbound TLP: IDO (attr_id_order), TD, EP, BYTE_ENABLE (FBE and LBE), byte_count
- Outbound TLP: TAG, EP, BYTE_ENABLE (FBE and FBE)

Currently, the architecture of Scoreboard can be understood in two different modes.

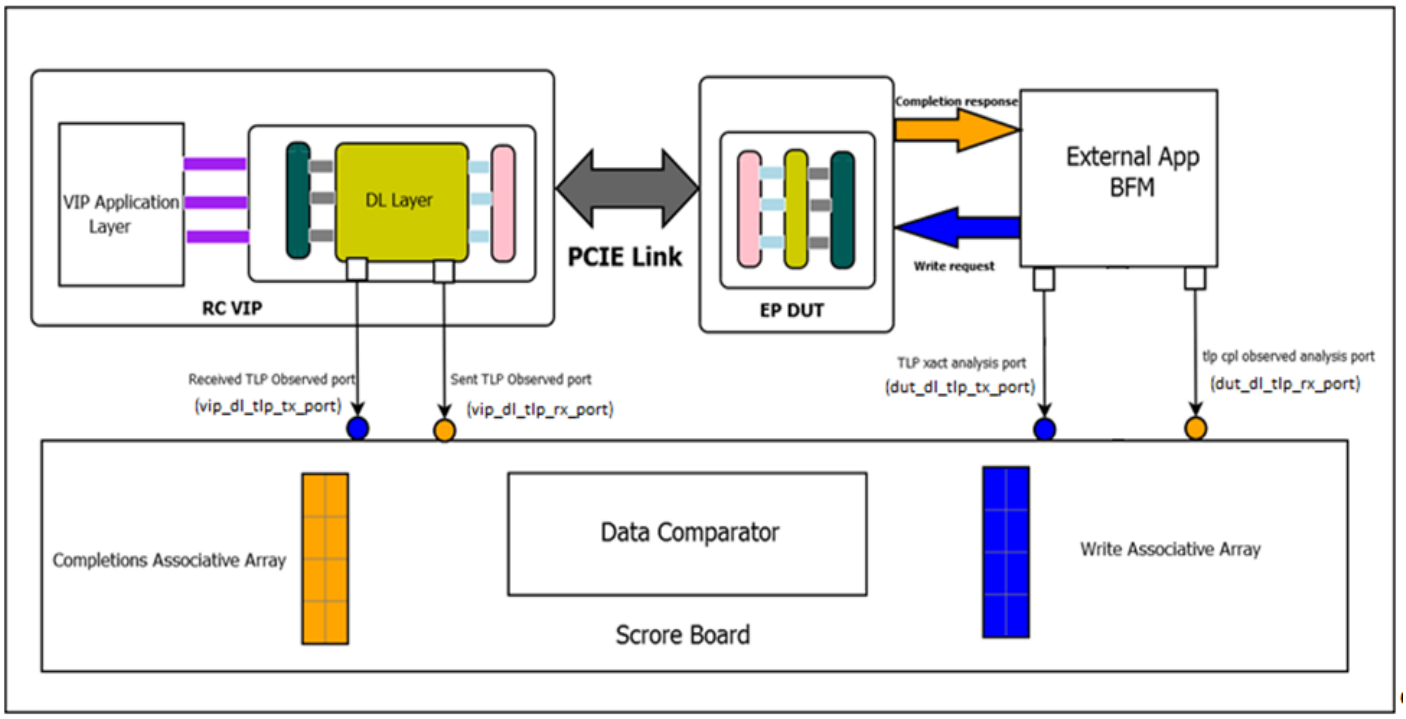
- VIP-DUT Mode
- VIP-VIP Mode

4.9.1 VIP-DUT Mode

Attention It is recommended to enable the Scoreboard so that errors are flagged in case any TLP compare failed. In case of miscompares on a DUT's TLP receive path, then the incorrectly filled TLP object received from the External Application BFM cannot be sent to the Application Agent for further processing.

VIP-DUT (EP) mode can be understood using Outbound traffic as an example. Refer to the following illustration Figure 4-11.

Figure 4-11 VIP-DUT (EP) Mode



For outbound write requests from an EP, store the data into a write associative array indexed with the transaction address through the analysis port available at External Application BFM. Once this request reaches the the RC VIP, you get it from a received TLP observed analysis port. Next, compare this against the data stored in the associative array.

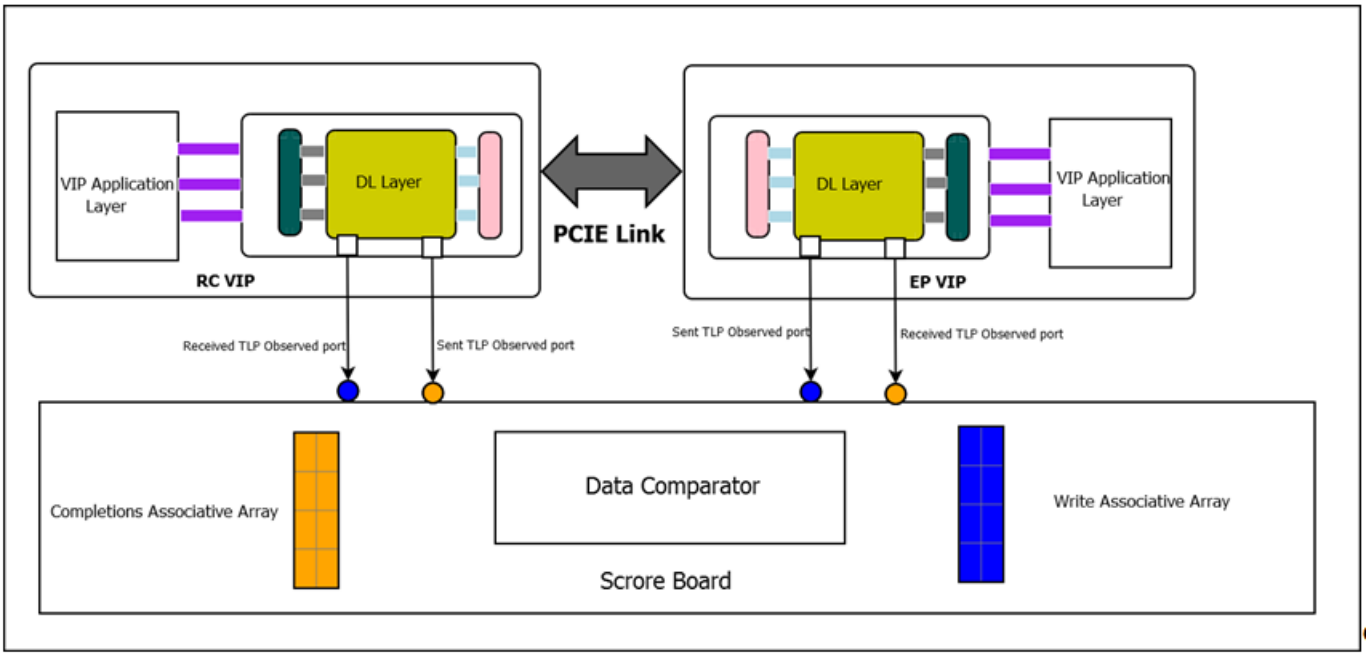
Once a read response is generated by the VIP, then you get it from the sent TLP observed analysis port. Next, store it into the completion associative array indexed with the transaction address.

Once this completion is received by an External Application BFM, pass it to the Scoreboard through a TLP CPL observed analysis port. Compare it against the data stored in completion associative array.

4.9.2 VIP-VIP Mode

This mode can be understood with Outbound traffic as an example. Consult the following illustration.

Figure 4-12 VIP-VIP Mode



For outbound write requests from the EP VIP, store the data into a write associative array indexed with the transaction address through analysis port available at in DL layer of EP VIP. Once this request reaches the RC VIP, get it from received a TLP observed analysis port, and compare this against the data stored in the associative array.

Once a read response is generated by the VIP, get it from sent TLP observed analysis port. Next, store it into the completion associative array indexed with the transaction address.

Once this completion is received by an EP VIP, pass it to Scoreboard through a TLP CPL observed analysis port. Compare it against the data stored in completion associative array.

4.10 Configuration

4.10.1 Configuration Data Classes

When any of the PCIe Test Suite testbench test cases are executed, the `build_phase()` of the test case base class first creates a top level configuration data object instance named `cfg`. This is an instance of the `svt_pcie_test_suite_configuration` class, and it serves as the entry point for setting up the UVM test environment's configuration.

- ep_cfg

  Initial configuration of the PCIe EP.

- rc_cfg

  Initial configuration of the PCIe Root Complex.

- app_sys_cfg

Container class which holds the configurations of the Application Master and Application Slave components. By default, this is of type `svt_axi_system_configuration`. It is expected to have sub-configuration object instances arrays named `master_cfg` and `slave_cfg`. Furthermore, it is expected that `master_cfg[0]` corresponds to the configuration of the DUT Application Slave interface (that is, the interface through which Application side sends PCIe TLP information), and it is expected that `master_cfg[1]` corresponds to the configuration of the DUT Application DBI Slave interface (that is, the interface through which DUT register/memory writes take place) and it is expected that `slave_cfg[0]` corresponds to the configuration of the DUT Application Master (DMA) interface (that is, the interface the DUT uses to access system memory).

In addition to the PCIe interface and Application interface configuration data objects, the Test Suite configuration class defines a few additional control fields, and a number of methods used to streamline the process of setting up the PCIe configurations. For more details about `svt_pcie_test_suite_configuration` class, see the HTML class reference.

4.10.2 Test Case Configuration

As noted above, the test case base class creates the configuration object and sets up the full initial configuration which typically involves calling one of the utility functions of the `cfg` instance (that is, one of the `svt_pcie_test_suite_configuration::setup_*_defaults()` functions). Then each test case if necessary may tweak the configuration to set up the desired details.

4.10.2.1 Runtime Configuration Control

The testbench *vcs_run_options* file can be edited to add runtime plusargs that sets configuration variables. This allows the same test case (set up with a default configuration) to be run with different configuration variations without modification or duplication of the test case. An example of a configuration setting that can be applied with this approach is as follows:

```
+pcie_cfg.pl_cfg.supported_speeds=32'h0000_0002
```

4.10.3 DUT Configuration

With respect to the configuration of the DUT itself, there are two types of configuration—hard configuration and soft configuration.

Hard configuration is normally bounded by the RTL itself, and fixed at the time the DUT instance is created (for example, as built-in to the DUT RTL structure, or as defined by RTL configuration parameters passed to the DUT instantiation).

Soft configuration may be specified and applied at runtime (for example, based on test case configuration data, and programmed into the DUT via the Application interface before initiating PCIe traffic).

4.10.3.1 DUT Hard Configuration

The hard configuration of the DUT is defined before compile time, based on how the DUT RTL is built. Therefore, the properties of the hard configuration must be reflected in any PCIe Test Suite test case intended to run with that DUT. To support this requirement, the testbench *sim_build_options* file must be edited to add the necessary configuration settings to match the DUT hard configuration.

4.10.3.2 DUT Soft Configuration

The soft configuration of the DUT should occur in the `configure_phase()` of the UVM testbench execution. With respect to the PCIe protocol, this phase must be executed before any PCIe traffic is initiated.

This soft configuration can be performed in the `configure_phase()` implementation of your External Application BFM.

The DUT PCIe EP soft configuration must be programmed based on testbench `cfg.ep_cfg` settings, within the bounds and capabilities determined by the DUT hard configuration.

Test case configuration details are set up (potentially randomized) with knowledge of, and within the bounds of the hard configuration.

For example, a soft configuration can be `expected_link_speed`.

4.10.4 DUT PCIe Endpoint Enumeration

The enumeration sequence in the Test Suite is used to discover all functions and to program BAR registers in all functions. The enumeration features are as follows:

- Support for all BAR types (I/O, Mem32, and Mem64).
- BAR support for a number of functions supported by the DUT.
- The maximum number of functions (excluding VFs) supported by the EP DUT is configurable by the attribute `max_num_functions_supported` of class `svt_pcie_test_suite_configuration`.
- The base address configured in each bar is mutually exclusive and can be selected in two different ways:
  - Random Mode: Base address is selected randomly. This is the default mode.
  - Incremental Mode: Base address is selected as incremental contiguous memory locations. To enable this mode, you must set the test suite configuration, `enable_incremental_bar_allocation` to 1. This ensures that there is no unused address space. (which may be the case with randomized allocation). This should be enabled if it is required to enable a large number of VFs or PFs requiring huge memory space. In this mode, you can also control the absolute starting and last address in memory space which can be used during BAR address allocation process for all the functions. You need to set the following test suite configurations to control the absolute address:
    - starting_bar_address_32
    - starting_bar_address_64
    - last_bar_address_32
    - last_bar_address_64
- Basic capability support for all functions.

- Extended capability support for all functions.

Note

- For a Non-ARI multi-function device, PCIe Test Suite does not support device enumeration when PFs are non-sequential.
- For an ARI multi-function device, by default all the functions starting from function 0 upto (`max_num_functions_supported`-1) are enumerated sequentially. You can also select to enumerate the selective non sequential functions. These functions are enumerated through the linked list of function numbers read from `'Next Function Number'` field from ARI Capability Register. You must set the `svt_pcie_test_suite_configuration` attribute, `'enable_selective_function_enumeration'` to 1 to enable this feature. You must set the value for `SVT_PCIE_TEST_SUITE_MAX_FUNC_SUPPORTED_BY_DUT` macro as (`highest_numbered_function_in_device` + 1). For example, consider if the ARI device has functions 0, 1, 5, 7, and 9, then you must set the value of `SVT_PCIE_TEST_SUITE_MAX_FUNC_SUPPORTED_BY_DUT` macro to 10. This is required to maintain the compatibility with the existing Test Suite architecture which was originally for sequential function numbers. Once this feature is enabled, all Test Suite tests will run only with the selective functions in the ARI device.
- To use DUT as a single function device, you must define the following in the compile file:

```
+define+SVT_PCIE_TEST_SUITE_MAX_VIR_FUNC_SUPPORTED_BY_DUT=0
+define+SVT_PCIE_TEST_SUITE_MAX_FUNC_SUPPORTED_BY_DUT=1
```

- You can program any random Bus and Device Number that you want EP DUT to capture during simulation. Non-zero Device Number should only be configured for non-ARI device.
- To program bus and device number, you need to configure `ep_cfg.driver_cfg[0].requester_id` and `ep_cfg.target_cfg[0].completer_id` fields accordingly.

  Here, `ep_cfg.driver_cfg[0].requester_id[15:8]` and `ep_cfg.target_cfg[0].completer_id[15:8]` are the bus numbers and `ep_cfg.driver_cfg[0].requester_id[7:3]` and `ep_cfg.target_cfg[0].completer_id[7:3]` are the device numbers for non-ARI device.
- You can also use the `'set_bus_numbers'` utility in `svt_pcie_test_suite_configuration` class to program the bus numbers for the RC and EP instances. Note that the bus numbers for RC and EP instances should be different.

Base addresses programmed in PF/VF BAR registers of DUT can be read via a call to public static member `get_bar_base_address()` of `svt_pcie_ts_device_system_virtual_base_sequence` class. For more details, see the HTML class reference.

Example:

bit [63:0] address = svt_pcie_ts_device_system_virtual_base_sequence class::get_bar_base_address(pf_number, vf_number, bar_num, is_vf);

PF and VF BAR type information determined from DUT's configuration space is available via `non_virtual_bar_present` and `virtual_bar_present` public static members of `svt_pcie_ts_device_system_virtual_base_sequence` class. For more details, see the HTML class reference.

On the basis of discovered BAR values, the `driver_app` and tlp base sequences/extended transaction classes constrain the address, length and transaction type of a transaction initiated by the Root complex.

- svt_pcie_config_space_status

  This is a Register model of endpoint PCIe Device configuration space in which image of endpoint Device's RO registers is updated by enumeration sequences. Whenever the configuration space of an endpoint device is searched for a basic or extended capability header through enumeration

sequences, it is also saved in the configuration space object for faster access in subsequent calls during the simulation.

- Reconfigurable Switches

  There is a attribute `max_num_functions_supported` of class `svt_pcie_test_suite_configuration` which need to be redefined by user as per number of functions in the endpoint device. The default value of this define is eight.

- Enumeration sequences

  Table 4-4 provides details about the enumeration phases and features, and the details of the sequence implementing the phase.

Table 4-4 Enumeration phases and features

Enumeration phase	Feature	Sequence Name	Sequence collection Filename
Function discovery	Read Vendor Id and Device Id	svt_pcie_ts_device_system_virtual_function_discovery_sequence	svt_pcie_ts_device_system_virtual_sequence_collection.sv
BAR configuration	Read all BAR and reconfigure them	svt_pcie_ts_device_system_virtual_bar_read_sequence	svt_pcie_ts_device_system_virtual_sequence_collection.sv
Base Capability	Reads config space of DUT to determine base address of the requested base capability structure.	svt_pcie_ts_device_system_virtual_base_cap_read_sequence	svt_pcie_ts_device_system_virtual_sequence_collection.sv
Extended Capability	Reads config space of DUT to determine base address of the requested extended capability structure.	svt_pcie_ts_device_system_virtual_ext_cap_read_sequence	svt_pcie_ts_device_system_virtual_sequence_collection.sv

4.10.4.1 Enable Memory and IO Space Decoding

Default during enumeration sequence Memory and IO space decoding is enabled before BAR configuration is completed. But if you want Memory and IO space decoding must be enabled once BAR configuration is completed then you must set the test suite configuration `enable_gen4_added_imp_note` variable from base test.

In base test during `creat_cfg()` subroutine, you must set this test suite configuration variable `enable_gen4_added_imp_note` (by default, it is set to zero).

```
/**
* This enable test suite env to behave as per added implementation note in
* PCIe Spec Revision 4.0 Version 0.7 Section 7.5.2.1.
* Default its set with zero , in case user want this need to enable it.
*/
cfg.enable_gen4_added_imp_note = 1'b0;
```

4.10.5 Customization of Test Suite Enumeration Flow (Available for EP DUT Only)

In the test suite, you can use the following methods for Enumeration flow:

- Test suite provided Enumeration flow (Default)

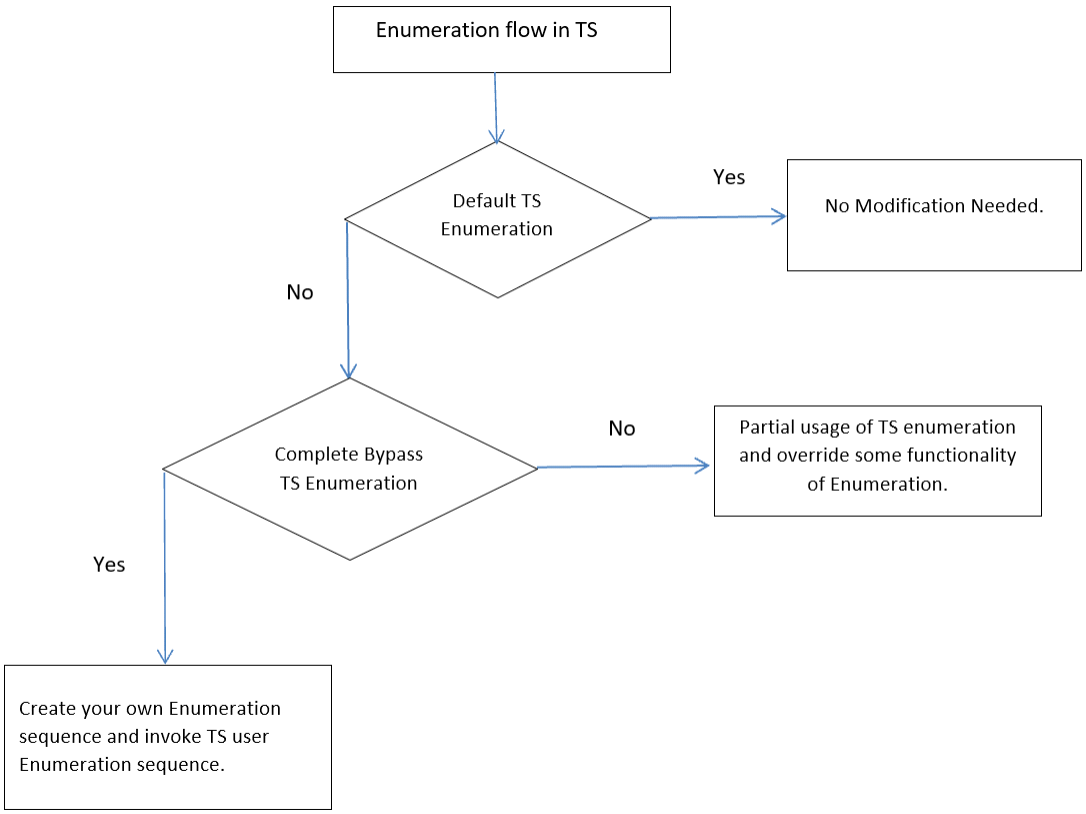
- Customization of Enumeration flow

This section discusses the method to customize the test suite Enumeration flow as per your requirement and how these changes will impact all test suite test cases.

Customization of Enumeration flow is shown in Figure 4-13.

- Either you can completely bypass test suite Enumeration flow and put your own enumeration flow.
- Override the test suite Enumeration process for some specific requirements—such as
  - Initialize the PCIe BAR Register with user-specific Address range.
  - Configure SRIOV Capability Register as per you requirement.
  - By default, enable/disable the reporting and logging mechanism during Enumeration.

Figure 4-13 Flow Chart to Hook Enumeration Sequence with Test Suite



4.10.5.1 Completely Bypass Test Suite Enumeration Flow

You can completely override (by using UVM factory override mechanism) the existing primitive sequence `svt_pcie_device_system_virtual_activate_link_sequence` (also known as, Activate link sequence, for more details of this sequence, see HTML class reference documentation) by putting your own piece of primitive sequence, so that you cannot completely bypass test suite Enumeration flow and pull in your required Enumeration flow. This is required because Activate link sequence is widely used in the test suite complex sequences, so replacing via UVM factory override is the only way to bypass complete Enumeration. When you use your own Enumeration flow, the test suite TL status attributes will not be populated correctly, so many test cases may fail with this flow. To avoid this, when your Enumeration

process is completed, you must invoke `svt_pcie_ts_device_system_virtual_user_enumeration_sequence` (for more details, see Details of Test Suite User Enumeration Sequence). This user Enumeration sequence will populate all the test suite TL status attributes with details—that is, BAR base address, Address range and so on.

The TL status attributes can be populated by setting the following test suite configuration attributes:

- Auto-tracing of configuration space and update TL status attributes
  - `enable_enumeration` – Disable it to bypass test suite Enumeration flow.
  - `ts_bypass_enumeration_populate_tl_status` – Populate TL stats flow, when you bypass test suite Enumeration flow.
- Manually updating Bar programming and update TL status attributes
  - `enable_enumeration` - Disable it to bypass test suite Enumeration flow.
  - `ts_bypass_enumeration_populate_tl_status` – Populate TL status flow when you bypass test suite Enumeration flow.
  - `ts_bypass_pcie_bar_programming` – Enable this only if you do not want to rely on PCIe BAR programming—that is, to provide the address ranges manually.
  - `ts_user_defined_starting_bar_addr` – If you want to provide Address range manually, then initialize this with your starting address. For more details, see class `svt_pcie_test_suite_configuration` in the HTML class reference documentation.
  - `ts_user_defined_bar_range` – If you want to provide address range manually, then you must initialize this with address depth for respective starting address. For more details, see class `svt_pcie_test_suite_configuration` in the HTML class reference documentation.

Note While using this flow for EP DUT, do not run the test cases when the `SVT_PCIE_ENABLE_ENUMERATION` switch is enabled.

4.10.5.2 Partial Usage of Test Suite Enumeration Overriding Some Operations

This section discusses the steps to use test suite Enumeration process and to enumerate the device with user-specific value like programming the BAR with user-specific address ranges, and programming the PCIe configuration space as per your requirements.

To use this flow, override the user Enumeration sequence `svt_pcie_ts_device_system_virtual_user_enumeration_sequence` and put the appropriate algorithmic code to enumerate the device in the specific method and sequence will be invoked post test suite Enumeration process.

You can explicitly program the BAR values and then TL status attributes will be updated. Set the following test suite configuration attributes:

- `enable_enumeration` – Enable test suite Enumeration flow.
- `ts_bypass_enumeration_populate_tl_status` – Populate TL stats flow when you bypass test suite Enumeration flow.
- `ts_bypass_pcie_bar_programming` – Enable this for explicit use defined in PCIe BAR programming—that is, if you want to manually provide the address ranges.
- `ts_user_defined_starting_bar_addr` – If you want to provide Address range manually, then you need to initialize this with your starting address. For more details, see class `svt_pcie_test_suite_configuration` in the HTML class reference documentation

- `ts_user_defined_bar_range` –If you want to provide Address range manually, then you need to initialize this with address depth for respective starting address. For more details, see class `svt_pcie_test_suite_configuration` in the HTML class reference documentation.

Note If you want to provide address range manually, then make sure to initialize it for all supported functions—that is, each function must have some memory range to target.

4.10.5.3 Test Suite Configuration Variables

Table 4-5 shows the list of configuration that need to be configured as the per requirements.

Table 4-5 Test Suite Configuration Variables

Configuration Variable	Description
`ts_bypass_enumeration_populate_tl_status`	Set this if you want to completely or partially skip the test suite Enumeration sequence and want to populate some of the TL Status attributes. This can be done by auto-tracing of configuration space or manual update.
`ts_bypass_pcie_bar_programming`	This configuration variable must be set if you do not rely on PCIe BAR programming for Address range selections. Set this if you want to provide address range manually.
`ts_user_defined_starting_bar_addr`	This test suite configuration variable is valid only if `ts_bypass_pcie_bar_programming` test suite configuration variable is set. This attribute replicates the starting address of respective function and BAR (Total Index[Func * BAR] and supposed to be initialized in incremental order). For more details, see class `svt_pcie_test_suite_configuration` in the HTML class reference documentation.
`ts_user_defined_bar_range`	This test suite configuration variable is valid only if `ts_bypass_pcie_bar_programming` test suite configuration variable is set. This attribute replicates the address depth of respected function and BAR (Total Index [Func * BAR] and supposed to be initialized in the incremental order). For more details, see class `svt_pcie_test_suite_configuration` in the HTML class reference documentation.

4.10.5.4 Details of Test Suite User Enumeration Sequence

The `svt_pcie_ts_device_system_virtual_user_enumeration_sequence` sequence in the test suite user Enumeration sequence is used for

- Creating your Enumeration sequence by extending it.
  - It provides a platform in the form of method where you can put your appropriate piece of code to enumerate the device.
  - It contains some read only method. It is not recommended to use this, functionality of this method to read the PCIe configuration space once programming of PCIe configuration space is done and populate the test suite TL status attribute.

- If you are using your own enumeration sequence (bypass the test suite Enumeration flow), then you need to call this sequence at last so that it will populate the test suite TL status attributes.

Methods Present in Sequence

Following are the list of methods that are present in the sequence:

Table 4-6 Sequence Methods

`initialize()`	This method will fetch the user-specific shared information. You can override this method.
`target_rid_initialize()`	This method will do the initialization for target rid for each supported functions. You can override this method.
`pf_bar_programming()`	It is a user-controllable method. If the user enumeration sequence is extended from this sequence, then the BAR programming algorithmic code should be present in this.
`pf_programmed_bar_read()`	It is a read only method. Control is not provided to the user, it will read the Programmed BAR Register for all supported functions (PFs) and then perform the initialization process for test suite TL status attributes related to memory addresses.
`sriov_ext_cap_programming()`	It is a user-controllable method. If the user enumeration sequence is extended from this sequence, then the code for programming of SRIOV Extended Capability Structure should be present in this.
`sriov_ext_cap_read()`	It is a read only method. It will read the content of SRIOV Extended Capability Structure and initialized the test suite TL status variables related to SR-IOV.
`vf_programmed_bar_read()`	It is a read only method and it is getting called during `sriov_ext_cap_read()`. It will read the programmed VF BARs Register and then perform the initialization process of all test suite TL status variables related to VF memory address.

4.10.6 Setting Up Test Suite/VIP Configurations

This section discusses some of the configuration methods for VIP and test suite configuration classes.

- Non User-Controllable
  - Default values in VIP/test suite configuration classes (usually fetches values from compile-time macros).
  - `create_cfg` method of `pcie_unified_base_test`
- User-Controllable
  - `create_cfg` method of `cust_base_test` (compilation required again after reconfiguration)
  - `create_cfg` method of `actual_test`: Scenario-specific settings required for a particular test are done here (compilation required again after reconfiguration).
  - Name value pair file (*.txt* file) (runtime option - compilation not required after reconfiguration)

    Following format is supported for settings done through *.txt* file:

```
cfg[0].ep_cfg.pcie_cfg.tl_cfg.remote_max_payload_size=4096
cfg[0].ep_cfg.pcie_cfg.tl_cfg.remote_max_read_request_size=4096
```

```
cfg[0].seq_cfg.tune_local_tx_preset_8g=1
```

Note

- Do not insert space between the property name and the value.

  ```
  cfg[0].ep_cfg.pcie_cfg.tl_cfg.remote_max_payload_size = 4096
  ```

- Radix of the value specified is determined by the type of variable/property. For example: for an int type of variable, value is considered to be in decimal, whereas for bit vectors, value is considered to be in hex.

  ```
  //Here 4096 is considered in decimal as attribute
  remote_max_payload_size is of type int
            cfg[0].ep_cfg.pcie_cfg.tl_cfg.remote_max_payload_size=4096
  //Here 10 is considered in hex as attribute length_tlp_packet_upstream
  isoftypebit[9:0](bitvector)
  cfg[0].seq_cfg.length_tlp_packet_min_upstream=10
  ```

- Any scalar rand variable loaded via file should have its rand mode turned off, this is essential to ensure any subsequent randomization on the object does not override user input but instead aligns other attributes to the desired value(s).

  ```
  //For eg. If user is turning off the rand_mode from the base test or
  actual test create_cfg method
  cfg.ep_cfg.pcie_cfg.tl_cfg.remote_max_payload_size.rand_mode(0);
  ```

- For more details, see *tb_dut_pcie/tests/reference_txt_file.txt.*

- SVT plusargs file (*.scr* file) (Runtime option - compilation not required after reconfiguration)

  Following format is supported for settings done through *.scr* file:

  ```
  +cfg[0]=ep_cfg.pcie_cfg.pl_cfg.link_width:1,rc_cfg.pcie_cfg.pl_cfg.link_width:
  1,ep_cfg.pcie_cfg.pl_cfg.expected_link_width:1,rc_cfg.pcie_cfg.pl_cfg.expected
  _link_width:1,ep_cfg.pcie_cfg.pl_cfg.supported_link_width_vector:1,rc_cfg.pcie
  _cfg.pl_cfg.supported_link_width_vector:1
  ```

- User-created instance of `all_links_cfg` (runtime option - compilation not required after reconfiguration)
  - You can create your own instance of `all_links_cfg` of type `svt_pcie_test_suite_unified_configuration`. This anchor instance holds the configuration properties of all the links. You can populate the elements of `all_links_cfg.cfg[i]` to set the link- i specific configurations.
  - Handle for this user-populated `all_links_cfg` instance can then be passed to the test suite environment through:

    ```
    uvm_config_db#(svt_pcie_test_suite_unified_configuration)::set(this,"*","us
    er_created_all_links_cfg",all_links_cfg));
    ```

  - If no instance for `all_links_cfg` is passed, then a new instance is created for `all_links_cfg` which acquires the default values for all the properties.

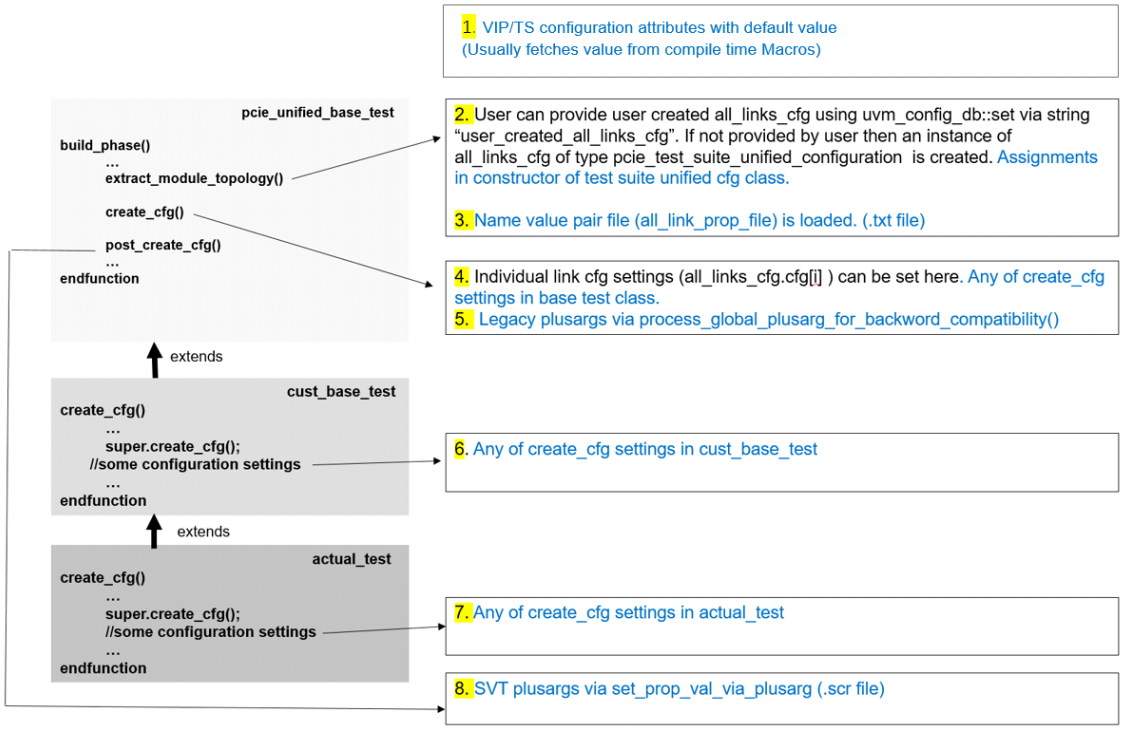
Note

If you want to provide your own instance of `all_links_cfg` with all the properties setup, then you can bypass the `create_cfg` methods and can control all the settings through this `all_links_cfg` handle. Any of the configurations can be changed at runtime without recompilation through this instance of `all_links_cfg`.

4.10.7 Precedence Flow to Overwrite VIP/Test Suite Configuration Attributes

VIP/Test suite configuration attributes can be overwritten from multiple places. The following diagram shows the precedence flow:

- The settings from higher precedence/point number overrides the lower precedence/point number settings.



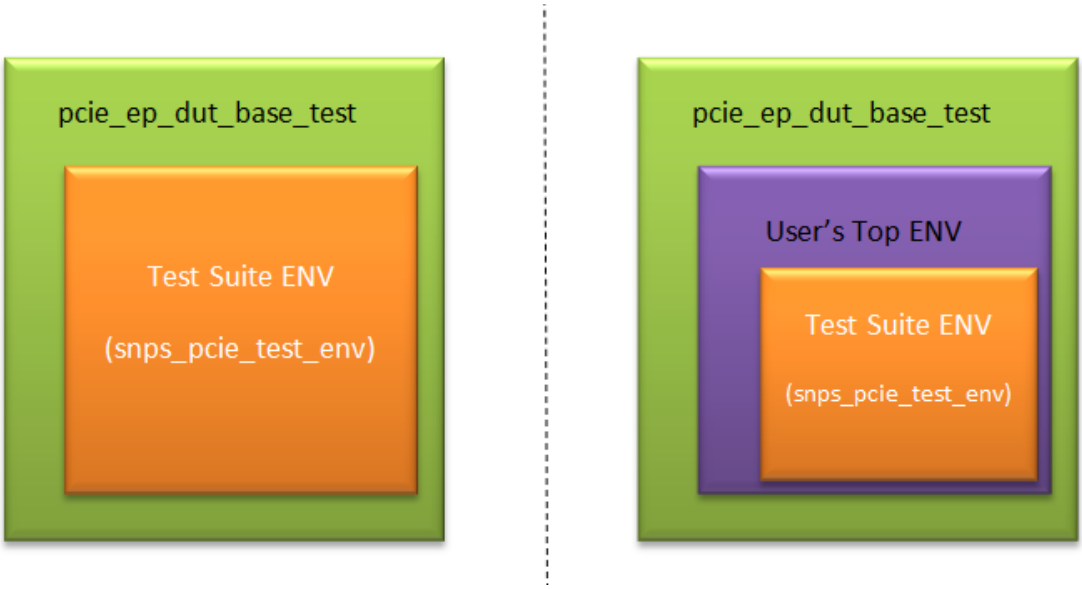
Note A *.txt* file can also be provided through *scr* file by `+svt_pcie_test_suite_all_link_prop_file=<relative path and name of the .txt file>`. But the precedence of *txt* file still remains at position 3 and can be overridden by settings from higher point numbers (4–8).

4.11 Test Suite Environment as Sub-Env in User's Top Env

The default use model of the Test Suite is for the base test to instantiate the Test Suite ENV class (`snps_pcie_test_env`) directly so that it becomes the top-level ENV. However, an alternate use model is to instantiate the top-level ENV as a sub-ENV in an existing top level ENV. Note that Figure 4-14 shows the use model view for environment construction. The default use model contrasts the use model where the "top

ENV" instantiates the Test Suite ENV (Right).

Figure 4-14 Use Model View for Environment Construction



The following attributes are in the `svt_pcie_test_suite_configuration` class to facilitate this use model:

```
/**
   * Flag that enables the top level test suite ENV object to be instantiated
   * as a sub-ENV of a user's environment.  When this flag is set to 1 then the
   * tests provided by the test suite do not construct an instance of the test
   * suite test ENV.  It is then the responsibility of the testbench integrator
   * to ensure that the 'env' reference that is in the base test is assigned
   * once their top-level ENV class is constructed.
   *
   * The default value for this property is 0 meaning that the tests provided
   * by the test suite will construct the test suite test ENV class as a top-level
   * ENV.
   */
  bit enable_integration_in_user_environment = 0;

  /**
   * Instance path to the test suite test ENV class.  The path should be relative
   * to the base test scope.  This can be used to provide this path if the
   * #enable_integration_in_user_environment property is used to integrate the
   * test suite test ENV class as a sub-ENV of a user's top-level ENV.
   *
   * The default value of this is "env" which matches the instance path used
   * when #enable_integration_in_user_environment is 0.
   */
  string path_to_pcie_test_suite_env = "env";
```

This string variable is overridden with `SVT_PCIE_TEST_SUITE_SNPS_ENV` inside SNPS Base test.

The default value of macro is env. You can override this macro with the required hierarchy.This will replace all instances of env with the string passed to macro.

By default, as shown in the previous code, the class `enable_integration_in_user_environment` is '0' and `path_to_pcie_test_suite_env` is set to "env". This matches the handle name in the base test.

To enable this use model, the following steps are required by the testbench integrator:

- Set the property `enable_integration_in_user_env` to '1' and set the macro `SVT_PCIE_TEST_SUITE_SNPS_ENV` to the correct hierarchical path in the top-level ENV in your extended base test (see section 6.2)

```
virtual function void create_cfg();

    // Create the configuration object.
    cfg = svt_pcie_test_suite_configuration::type_id::create("cfg",this);

   // Update the hierarchical path to SNPS test suite env
   // and set the enable_integration_in_user_environment = 1
   cfg.enable_integration_in_user_environment = 1;
```

Set the macro `SVT_PCIE_TEST_SUITE_SNPS_ENV` to `top_env.snps_ts_env`.

- Construct the top-level Test Suite ENV as a sub-ENV of the top-level ENV in the build-phase.

```
function void pcie_ep_user_top_env::build_phase(uvm_phase phase) ;

  ...

   //Construct the SNPS test suite environment
  snps_ts_env = snps_pcie_test_env::type_id::create("snps_ts_env", this);

  ...
```

- Create an instance of the user top-level ENV within your extended base test and construct this during the `build_phase()`.

```
class pcie_ep_dut_base_test extends uvm_test;

   /**
    * Top level Env (acting as customer's top env)
    */
  pcie_ep_user_top_env  top_env;

...
...
...

virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    ...
    //Construct top level environment
    top_env = pcie_ep_user_top_env::type_id::create("top_env", this);
```

- Within the `connect_phase()` of your extended base test, "env" instance in the base test must be assigned with a reference to the `snps_pcie_test_env`—that is, in your top-level ENV.

```
function void connect_phase(uvm_phase phase);
   super.connect_phase(phase);

   //Update the handle of test suite Env
   env = top_env.snps_ts_env;
   ...
   ...
```

- Update the path of Test Suite ENV in `test_top` module when setting virtual interface setting through configDB.

```
module test_top;
…
initial begin
    uvm_config_db#(virtual `SVT_PCIE_TEST_SUITE_PCIE_EP_TOP_IF)::set(uvm_root::get(),
"uvm_test_top.top_env.snps_ts_env", "ep_pcie_if", ep_pcie_if);
```

4.12 Test Suite test_top as Child Module in User's test_top

You can override SNPS `test_top` module with any name by overriding macro `SVT_PCIE_TEST_SUITE_SNPS_TEST_TOP` to achieve below functionality. By default, the value of this macro is `test_top`.

- To invoke SNPS `test_top` as child module inside user test_top.

  Here, user has the option to invoke `run_phase` from SNPS top module using parameter `SVT_PCIE_TEST_SUITE_INVOKE_RUN_PHASE`.

  Default value of parameter is 1, which implies that the `run_phase` is invoked from SNPS top module.

- To rename SNPS `test_top` module

4.13 Test Pass/Fail Message in Simulation Transcript

Following message gets printed in the transcript if test is passed:

```
SvtTestEpilog: Passed
```

Following message gets printed in the transcript if the test fails:

```
SvtTestEpilog: Failed
```

You must update the '`dut_capabilities`' configuration properties according to the DUT's implementation. If a test is run which exercises an optional feature not implemented by the DUT (as determined from the `dut_capabilties` configuration or DUT's configuration space), then it will fail with a warning similar to following:

```
`uvm_warning("build_phase", "OptionalFeatureNotSupportedByDUT : Test not run as ATS is
not supported by DUT as determined from the dut_capabilities configuration.");
```

The keyword "`OptionalFeatureNotSupportedByDUT`" will be present in all such warnings.

4.14 Regression Test List

The following table lists regression tests.:

Table 4-7 Regression Test List

Regression List	Description
pcie_ep_common_tests.sv	Common tests for Gen2/Gen3/Gen4 specification.
pcie_ep_gen2_tests.sv	Gen2 spec specific tests
pcie_ep_gen3_tests.sv	Gen3 spec specific tests

Table 4-7 Regression Test List

Regression List	Description
pcie_ep_gen4_tests.sv	Gen4 spec specific tests

These files are present in *env* directory under testbench area:

- tb_dut_ep_gen3_* have all the test list files.
- tb_dut_ep_gen2_* have *pcie_ep_common_tests.sv* and *pcie_ep_gen2_tests.sv*.
- tb_dut_ep_gen4_* have all the test list files.

4.15 Error/Normal Scenarios

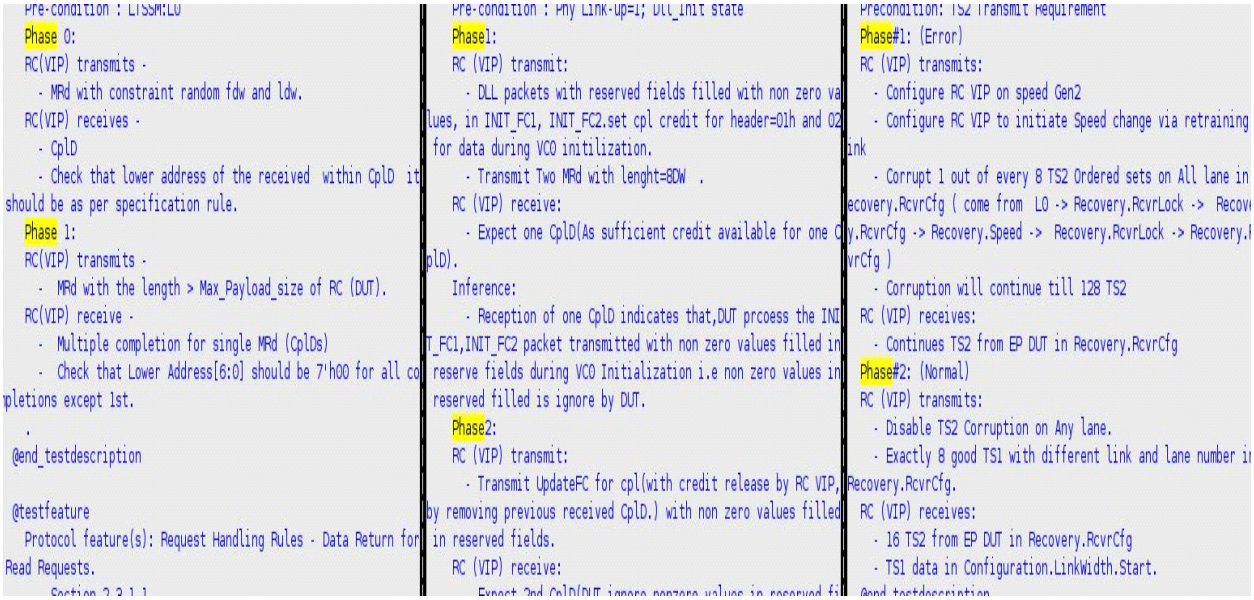
Test Suite covers both type of scenarios: normal and error.

Naming conventions:

- All *ts.*_error.sv* cases are errors while all other cases are normal scenarios.

Strategy in error scenarios:

- Validate normal traffic after error injection during phase1 of test is complete to verify behavior of the DUT is as expected when the error condition is removed.



4.16 Check ‘x’ or ‘z’ on Signal

This feature checks that the signals at the interface are not driven to 'x' or 'z' state after reset de-assertion. The feature is command line controllable. It is disabled by default, it can be enabled in the Unified TB (`tb_dut_pcie`) by adding a macro `+define+SVT_ENABLE_XCHECK=1` in the compile flow.

This check is used to check the case when a signal is not connected or if the signal is wrongly driven to tristate.

It is implemented on the principle that once Unified VIP is out of power on reset (issued via "reset" node) it can check all applicable (based on `svt_pcie_device_configuration` content) interface signals for known values.

In case of signal driven to x or z, the error signature will be as follows.

```
[Unknown Value Observed by X Checker instance] offending signal is connected to
interface instance
test_top.spd_1.mpipe.port0.pipe_if.u_xcheck_pipe_if_rx_eq_in_progress_0.unnamed$$_1.
Please refer to instance name for further details.
```

In the above example message, the VIP is complaining about interface signal `pipe_if_rx_eq_in_progress_0`.

- Limitation
  - This check is not applicable for conventional TBs.

4.17 User-Specific Error Demotions in a Separate File

Perform the following steps for error demotions in a separate file:

- Create a new error demoter class and extend it from `svt_err_catcher`.

  Snapshot of the class

```
class svt_pcie_error_demoter extends svt_err_catcher;

  /** Factory registration.*/
  `uvm_object_utils(svt_pcie_error_demoter)

  function new(string name = "svt_pcie_error_demoter");
    super.new(name);
  endfunction

  function void catch_error();
      //Statement for error demotion
      add_message_text_to_demote("/Received non-IDL token/");
  endfunction

endclass
`endif
```

- Include that file in the package so that this file gets compiled.
- Instantiate the newly created class in user-specific base test in build phase.

```
// Done inside tests
 /** Create response catcher class and append it to callback pool. */
 demoter = reg_model_warning_demoter::type_id::create({get_full_name(), ".demoter"},this);
 uvm_report_cb::add(null, demoter);
end
```

4.18 Verification Planner Usage

4.18.1 Back-Annotation of Test Pass/Fail Results (From VDB)

The Synopsys Verification Planner tool provides a mechanism for back-annotating the test pass/fail results to a test plan from VDB. Perform the following procedure to back-annotate the test results:

1. Modify the `sim_build_options` to define the `SVT_PCIE_TEST_SUITE_ENABLE_PASS_FAIL_COVERAGE`.

   For example, add the command in `sim_build_options`.

   ```
   +define+SVT_PCIE_TEST_SUITE_ENABLE_PASS_FAIL_COVERAGE
   ```

2. Modify the `sim_build_options` to include a VCS include directory for the *CoverMeter.vh* file.

   For example, add the command in `sim_build_options`.

   ```
   +incdir+${VCS_HOME}/include/
   ```

3. Modify the `sim_run_options` with the following options (only when functional coverage/check rule coverage is not enabled).

   ```
   -cm_test $scenario -cm_name $scenario -cm_stats all -cm_dir mysimv.vdb
   ```

   Note This will create a vdb with the name "mysimv.vdb".

4. Use the `hvp` annotate command to generate the back-annotated sub-plan file:

   ```
   hvp annotate -plan top_plan_name.xml -dir "<vdb_name>.vdb"
   ```

Example 4-1 Generating a Back-Annotated XML Test Plan

The following command generates a back-annotated test plan name *pcie_ts_dut_ep_toplevel_test_plan.xml*, using a `vdb` named `mysimv.vdb`:

```
hvp annotate -plan pcie_ts_dut_ep_toplevel_test_plan.xml
-dir "mysimv.vdb"
```

4.18.2 Merging the VDBs

Use the following command to merge multiple VDBs to a single VDB:

```
$(VCS_HOME)/bin/urg -lca -dir <VDB_1.vdb VDB_2.vdb …. VDB_N.vdb> -dbname combined
```

By using the above command, `VDB_1.vdb, VDB_2.vdb ….. VDB_N.vdb` are merged to a single VDB named `combined`. For more information, use the following command:

```
urg -help
```

4.19 Test Selection and Tuning

New information in the test header has been added to ease the process of test selection and achieving overall regression stability. It will help user to determine under what conditions a test is valid. You can

utilize it and create a desired test list according to your requirement and DUT capabilities. Also, set up a TCF with appropriate configurations for each test.

Note Currently, it is only done in few cases of EP/RC/PHY mode.

- Following new information will be present in each test case inside `@testcfg` and `@end_testcfg`:

  `@testcfg`

  - `TS_EXCLUSION_REASON`:
    - `TS_EX_LINK_WIDTH`: <*Ex. Test is not applicable on x1*>
    - `TS_EX_SPEED`: <*Ex. Test is not applicable to Gen1/2/3 speed*>
    - `TS_EX_OPTIONAL_FEATURE`: <*Ex. RCB rule checking/Byte enable check/…. Etc*>
    - `TS_EX_INTERFACE`: <*Ex. Not applicable on PIPE*>
    - `TS_EX_SNPS_IIP_TEST`: <*Ex. Not applicable for third party DUT*>
  - `TS_OPTIONAL_FEATURE_CONFIG_SPACE`: <*Provide config space optional feature information. If not present, then mark N/A* >
  - `TS_OPTIONAL_FEATURE_DUT_CAPABILITY`: <*Provide attribute name. If not present, then mark N/A* >
  - `TS_APP_BFM_DEPENDENCY`: <*provide attribute name which is used in uvm_config_db. If not present, then mark N/A*>
  - `VIP_CFG_DEPENDENCY`: <*Any timer/Any specific VIP configuration. If not present, then mark N/A* >
  - `TS_CFG_DEPENDENCY`: <*Scoreboard/enumeration disable enable. If not present, then mark N/A*>
  - `TS_SEQ_CFG_DEPENDENCY`: <*Any dependent sequence configuration. If not present, then mark N/A*>
  - `TS_TXT_FILE_DEPENDENCY`: <*Test need to run with any txt file, if not present, then mark N/A*>

  `@end_testcfg`

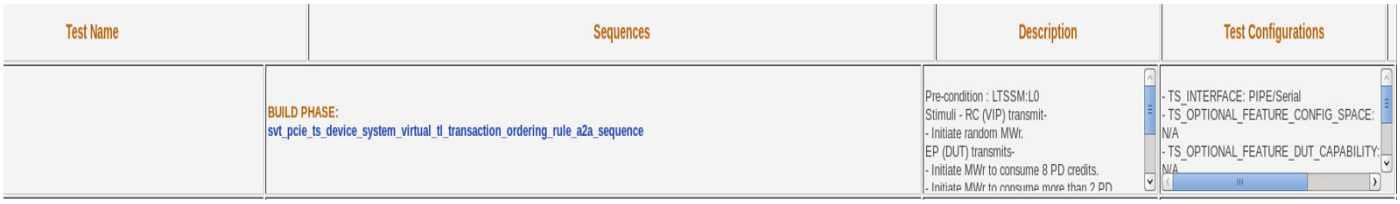
- Examples for using keywords for test selection (before running the tests)
  - If you want to extract link width exclusion information, then use the following command:

    ```
    grep -r "TS_EX_LINK_WIDTH" tests/ts.*-ep.sv | grep -v "N/A"
    ```

    Output
    - *tests/ts.gen2_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_corruption_error-ep.sv*: `TS_EX_LINK_WIDTH`: Test is not applicable for x1 link width.
    - *tests/ts.gen2_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_link_lane_mismatch_error-ep.sv*: `TS_EX_LINK_WIDTH`: Test is not applicable for x1 link width
  - If you want to extract test cases that have to be excluded on specific link width like x4, then use following command:

    ```
    grep -r "TS_EX_LINK_WIDTH" tests/ts.*-ep.sv | grep -v "N/A" | grep "x4"
    ```
  - Similar `grep` command can be used for speed related exclusion (`TS_EX_SPEED`), optional feature related exclusion (`TS_EX_OPTIONAL_FEATURE`), or any VIP configuration dependency (`VIP_CFG_DEPENDENCY`) related information and so on.
- Output of `@testcfg` and `@end_testcfg`

- Output is shown in HTML reference documentation in the following format.



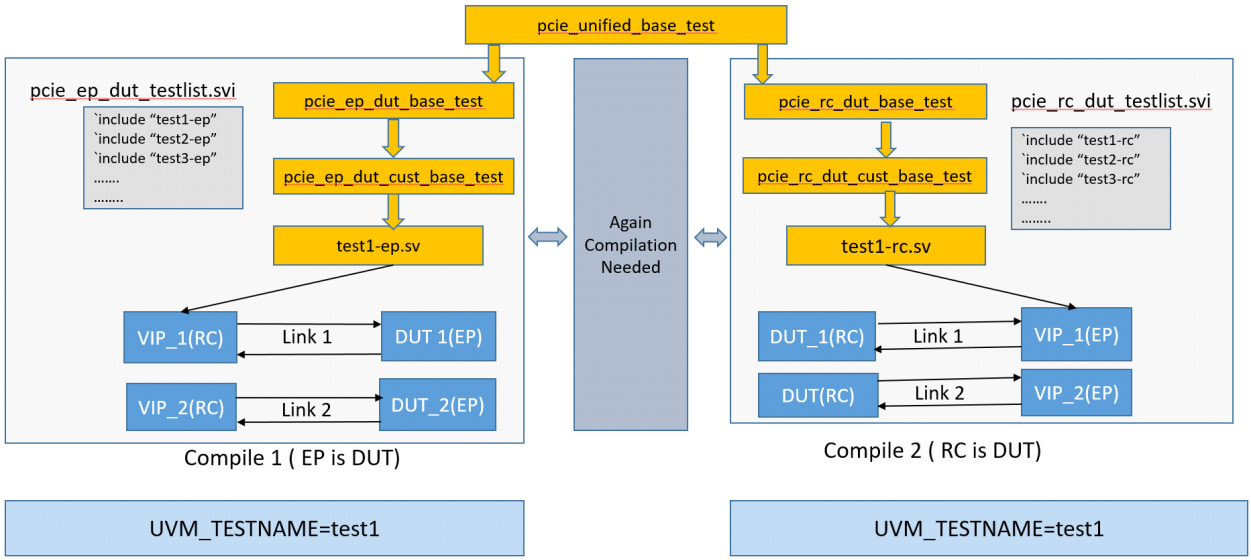
4.20 Concurrent DUT Roles

The PCIe test suite has been enhanced to support both RC and EP DUT modes with single compilation. This feature allows the RC DUT and EP DUT to operate on two or more independent links in a single simulation.

- Without Concurrent Mode (existing implementation)
  - Without concurrent DUT mode support, if you want to switch from EP DUT to RC DUT mode or vice-versa, then you have to compile again (tests, sequence, callbacks) and all links in multi-link setup to use the same DUT role.
  - Same test on each link

    The test selection naming convention is as follows:
    - For EP DUT, `UVM_TESTNAME=test1`
    - For RC DUT, `UVM_TESTNAME=test1`

Figure 4-15 Without Concurrent Mode - Same Tests on Each Link



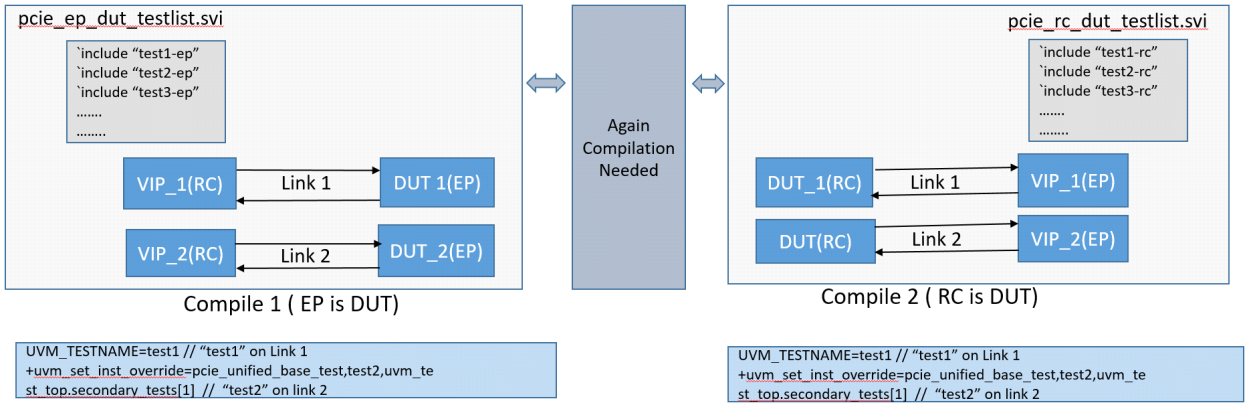
- Different tests on each link

  Use the following mechanism to run separate tests on first link and second link:

  `UVM_TESTNAME=test1` *// "test1" will be on link 1*

```
+uvm_set_inst_override=pcie_unified_base_test,test2,uvm_test_top.secondary_tes
ts[1] // "test2" will be on link 2
```

Figure 4-16 Without Concurrent Mode - Different Tests on Each Link



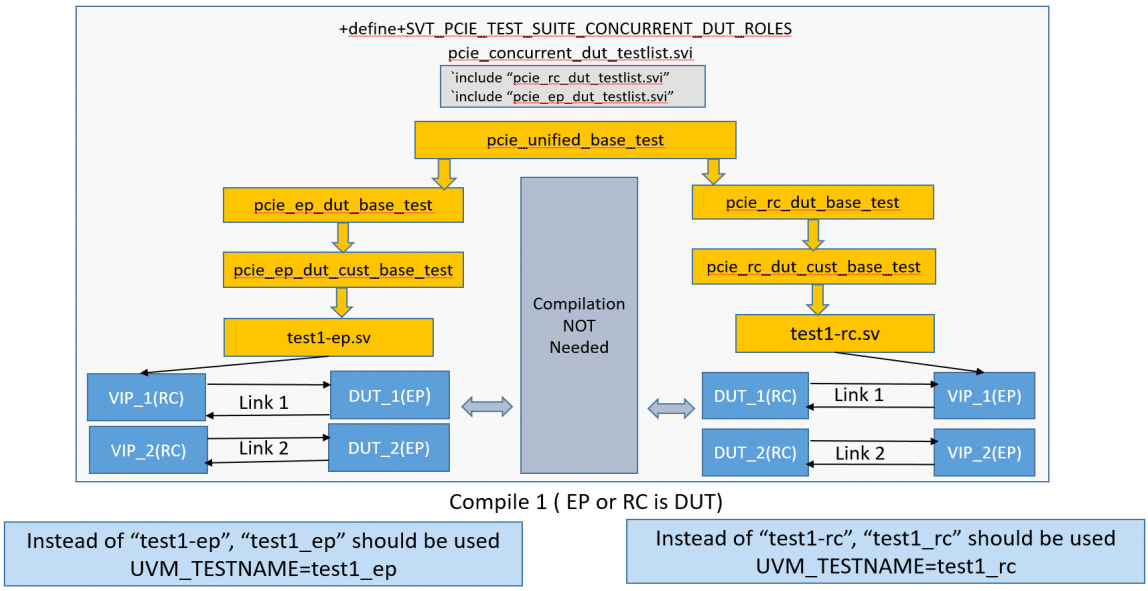
- With Concurrent Mode

  The modified tests used by this feature are available in *pcie_concurrent_dut_testlist.svi* file (*tb_dut_pcie/pcie_concurrent_dut_testlist.svi*). These files were modified to support concurrent DUT operation so that a single simulation can be used to simulate an EP DUT and RC DUT on independent links.

  To use this feature, perform the following steps:

1. Define the `SVT_PCIE_TEST_SUITE_CONCURRENT_DUT_ROLES` macro.
2. Include the tests listed in *tb_dut_pcie/env/pcie_concurrent_dut_testlist.svi* during compilation.
3. Modify the simulation launch scripts to pass `UVM_TESTNAME=test1_ep` or `UVM_TESTNAME=test1_rc` (instead of `UVM_TESTNAME=test1`). If you have defined `SVT_PCIE_TEST_SUITE_CONCURRENT_DUT_ROLES` and invoked any of the test suite RC DUT/EP DUT tests using `UVM_TESTNAME=test1`, the you may encounter errors stating that the desired test name is not found (that is, the trailing `_rc/_ep` in the string passed to `UVM_TESTNAME` is mandatory) This rule is applicable only to the test suite tests provided with the installation and it is not applicable for user-written tests.

Figure 4-17 With Concurrent Mode



To concurrently simulate DUT RC and EP ports, compile the testbench with all links such that all Full Stack VIP instances are set as Endpoints and all Application layer only VIP instances (that is, the ones driving DUT Non-PCIe interface) are set as RC (or vice-versa) and use the plusarg `svt_pcie_ts_flip_device_roles=32 bit hex_value`.

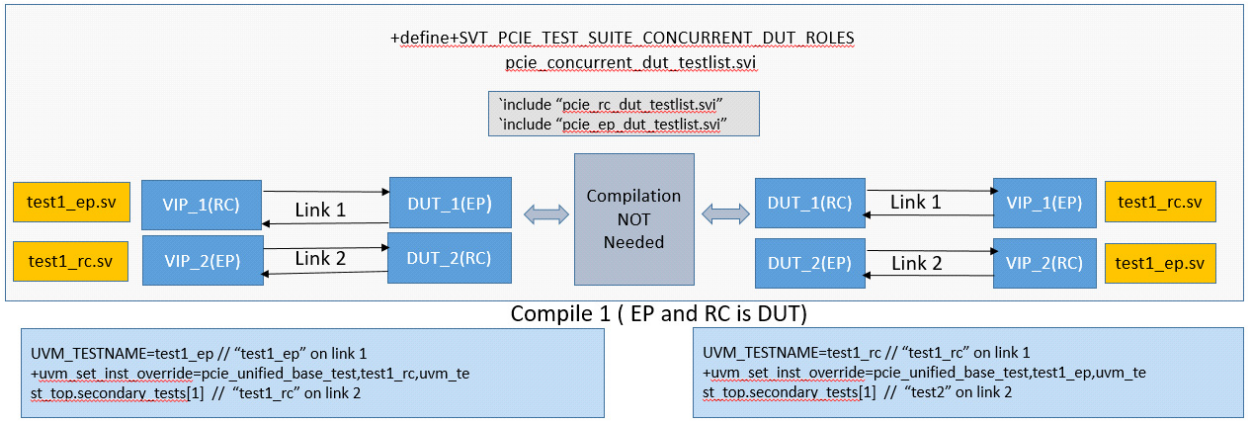
Each bit of the plusarg `svt_pcie_ts_flip_device_roles` acts as a control to retain (or flip) the compile time roles of VIP instances associated with that link. Value of 1'b1 indicates the role of the VIP instances associated with that link will flip and value 1'b0 indicates the role of the VIP instances will retain the roles assigned at compile time.

For example, if `svt_pcie_ts_flip_device_roles=32'h0000_000C,` then VIP instances associated with links 0 and link 1 will retain the roles assigned at compile time and VIP instances associated with link 2 and link 3 will flip the roles at runtime.

Make sure to invoke the correct test type (EP DUT or RC DUT test) on the link after accounting for the compile time role settings and the applicable role flips.The test suite tests follow a naming convention (`*_rc` indicate RC DUT tests and `*_ep` indicate EP DUT tests), relying on this convention the base test tries to check for consistency across the test type invoked and the DUT/VIP role settings. It does this check via method `pcie_unified_base_test::check_test_name`.

```
UVM_TESTNAME=test1_ep  // "test1_ep" will be on link 1

+uvm_set_inst_override=pcie_unified_base_test,test1_rc,uvm_test_top.secondary_tests[1]
// "test1_rc" will be on link 2
```



4.21 TLP Bypass Support (Only for SNPS EP IIP Core)

The PCIe test suite has been enhanced to support the DWC PCIe EP IIP configuration in which configuration access on the wire is passed to the target interface instead of being processed internally. These requests are routed to the Application layer VIP via the test suite scoreboard. By default, the Application layer VIP generates completions with successful status and random payload if it is a read access. To alter the completion response, use VIP callbacks.

To enable this feature, set the bit during the `build_phase` execution in the user base test.
`svt_pcie_test_suite_configuration::enable_dwc_ep_config_bypass`

For example, `cfg. enable_dwc_ep_config_bypass = 1;`

Currently, this feature supports Type 0 configuration requests, but Type 1 configuration requests will receive an Unsupported Request (UR) warning. This message can be demoted.

4.22 Selecting Secondary Tests

The PCIe test suite has been enhanced to support the test type run on secondary links without using the command line inputs from the user.

New method for selecting secondary tests is as follows:

```
virtual function string
pcie_unified_base_test::randomly_selected_test_name(`SVT_PCIE_TEST_SUITE_CONFIGURATION_TYPE link_cfg);
```

The above method serves as a customizable hook. The existing implementation of this method is a prototype used only for demonstration and you must override this implementation to suit your DUT needs. This new feature is turned off by default. To enable this feature, perform either of the following steps:

- Pass the command-line plusarg `svt_pcie_ts_random_test_type_for_secondary_test=1`.

  OR

- Add `random_test_type_for_secondary_test=1` in the base test (extension of `pcie_unified_base_test`) before invoking the `super.build_phase()`.

The function returns a string containing the desired test name which in turn is used to set an instance override before creating a secondary test instance. If this feature is enabled, this function will be invoked after preceding each secondary test instance creation.

Note Combining this feature with the existing command-line option based secondary test selection is not recommended and may yield undefined results.

5 Integrating EP, RC and PHY

Starting O-2018.06-3 release, DUT integration and Application BFM sections have been moved to the following guides:

- **EP/RC DUT**
  - VC Verification IP PCIe Test Suite EP/RC DUT Integration Guide

    $DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/doc/pcie_test_suite_ep_rc_integration_guide.pdf
- **PHY DUT**
  - VC Verification IP PCIe Test Suite PHY DUT Integration Guide

    $DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/doc/pcie_test_suite_phy_integration_guide.pdf
- **External Application BFM**
  - VC Verification IP PCIe Test Suite Reference External Application BFM

    $DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/doc/pcie_test_suite_app_bfm_reference_guide.pdf

6
Using the Test Suite: EP DUT

This chapter discusses the following topics:

6.1 Command-Line Options for Demo Mode

6.1.1 EP VIP as DUT in SERDES mode

```
gmake tl_directed_traffic-ep
USE_SIMULATOR=vcsvlog
SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_ep_vip_serial.f
SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_vip.f
```

6.1.2 EP VIP as DUT in PIPE mode

```
gmake tl_directed_traffic-ep
USE_SIMULATOR=vcsvlog
SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_ep_vip_pipe.f
SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_vip.f
```

6.2 Extended Base Test Use Model

You must extend the Test Suite's base test class to add implementation specific customizations. Following are the steps:

1. Create a new file by taking reference from *pcie_ep_dut_cust_base_test.sv*.

It is not recommended to extend this class, use it only as a reference.

2. Add the following two macros in the user defines file (*svt_pcie_test_suite_user_defines.svi*):
   - `SVT_PCIE_TEST_SUITE_EP_DUT_TEST_FILE` with the name of this new file.
   - `SVT_PCIE_TEST_SUITE_EP_DUT_TEST` with the name of this new class.

   In this class, you must set the following configurations:

- Set the configuration properties for link partner VIP based on DUT requirement—timer configuration, link width, and speed.

  For example, when EP is DUT, the cfg configurations are required for RC VIP.

```
cfg.rc_cfg.pcie_cfg.dl_cfg.min_num_tx_initfc1_p = 15
```

- All relevant properties in the `dut_capabilities` configuration object can be programmed as per DUT specific implementation.

  For example, if EP is DUT with ATS feature support, it can be set as shown below:

  ```
  cfg.ep_cfg.pcie_cfg.dut_capabilities.enable_ats_support
  ```

- Error Demotion
- You must reimplement `pcie_unified_base_test::set_env_override` method in this class. This method is invoked in `pcie_unified_base_test` at an appropriate point before the `build_phase`.
- You can add your own custom application to application agent.

  This class also serve as placeholder for all configuration settings and 'workarounds' specific to your implementation. You must add the following as per your implementation:

  - If the DUT has limitations on outbound traffic and is not able to transmit TLPs 'as is', then a callback similar to `update_app_xact_for_32bit_axi_callback` (take reference from *pcie_env_callback_collection.sv*) must be developed and added to TLP Mapper Component's callback pool (take reference from *pcie_ep_dut_cust_base_test.sv*). If there is no limitation, then the callback should not be added.
  - If the DUT is not able to calculate a `completer_id` for outbound completions, then it must use a callback similar to `completer_id_calculator_callback`.
  - If the DUT requires completion to be generated from the target application present in Application Agent, then it may want to look at configuration properties available for the target application. For example, '< min | max>_read_cpl_data_size_in_bytes' properties can be set to configure how split completions should be generated.
  - Adjust the Scoreboard switches available in the Test Suite configuration.

Note Even if the Scoreboard is disabled, the configuration settings corresponding to inbound TLPs must be set as per your Application BFM's limitations (if any).

  - Set other Test Suite configuration properties.
  - All relevant properties in the `dut_capabilities` configuration object must be set here as per your specific implementation.

6.3 Simulation Timeout and Usage

The test execution is controlled by a global timeout attribute and is terminated when the timer expires. The default value of test timeout is 1200000ns. For a specific test run the default value can be overridden by command line argument SVT_PCIE_TEST_TIMEOUT= *timeout-value-in-ns*.

6.4 MFVC Tests Behavior

The MFVC tests are applicable only if DUT supports MFVC or VC capability. The enumeration sequence discovers MFVC or VC capability support in the DUT, and these tests are executed only if the DUT advertises support for MFVC or VC capability during enumeration.

6.5 Fast Simulation Mode

To facilitate faster simulation of tests, the Test Suite provides a mechanism to set up scaled down values of specification timers and the number of ordered sets communicated during the link initialization phase. This assumes that the DUT also supports the fast simulation mode and the timers can be configured accordingly. Also, it is required to match the RC VIP configuration attributes to be same as of DUT values.

Table 6-1 provides the details of replaceable macro defined in svt_pcie_test_suite_defines.svi (present in the sverilog/include/ directory in the install tree) for fast simulation mode.

Table 6-1 Replaceable macro defined in svt_pcie_test_suite_defines.svi for fast simulation mode

`SVT_PCIE_TEST_SUITE_FAST_SIM_1MS svt_pcie_device_configuration	PCIe VIP RC (When EP as DUT) This macro defines the scaled down value of 1milisec timer for fast simulation mode in ns. (Default value is1024ns)

6.5.1 Configuration of LTSSM State Timers for Fast Simulation Mode

To configure all LTSSM timers, based on the scale factor in the DUT, set the value to the `SVT_PCIE_TEST_SUITE_FAST_SIM_1MS` macro, in ns (default value is 1024). All the timer values are calculated internally (for example, `` `define SVT_PCIE_TEST_SUITE_FAST_SIM_12MS 12 * `SVT_PCIE_TEST_SUITE_FAST_SIM_1MS ``).

Based on the macros defined for different timeouts, the LTSSM timer attributes in the `svt_pcie_test_suite_configuration` class of RC VIP are set up.

6.5.2 Configuration of Erroneous Counts of TS1 or TS2 Ordered Sets

In the PL sequences, erroneous TS1 or TS2 counts are used and can be controlled based on each LTSSM state timeout condition. The attributes are:

- `num_ts_os_error_2ms`
- `num_ts_os_error_12ms`
- `num_ts_os_error_24ms`
- `num_ts_os_error_32ms`
- `num_ts_os_error_48ms`

The above attributes can be updated in the base test (applicable for all PL layer sequences).

Table 6-2 shows the replaceable defines for the erroneous count order sets and their default values.

Table 6-2 Replaceable Defines for the Erroneous Count Order Sets

`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_2MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 2ms timeout condition. (Default value is 10)
`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_12MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 12ms timeout condition (Default value is 50)

Table 6-2 Replaceable Defines for the Erroneous Count Order Sets

`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_24MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 24ms timeout condition (Default value is 100)
`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_32MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 32ms timeout condition (Default value is 100)
`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_48MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 48ms timeout condition (Default value is 100)

6.6 SRIOV

The default behavior of SRIOV in the test suite is as follows:

- SRIOV feature is enabled (to disable, you have to pass the `SVT_PCIE_DISABLE_SRIOV` command-line switch).
- Number of Physical function is eight. (it can be controlled with attribute `max_num_functions_supported` of class `svt_pcie_test_suite_configuration`).
- Number of Virtual functions are calculated during enumeration. But for connection at Application BFM side with the DUT, it is controlled with macro `` `SVT_PCIE_TEST_SUITE_MAX_VIR_FUNC_SUPPORTED_BY_DUT `` (its default value is 256).

6.6.1 Configuring SRIOV Capabilities in Physical Functions

The RC VIP randomly configures a supported page size in SRIOV extended capability structures of all Physical Functions (PF). The maximum page size that will be set by VIP while setting up SRIOV registers is controlled via `max_page_size_set` field in the test Suite configuration. The actual page size that will be set by VIP is randomized, constrained to this maximum value and the supported page sizes of PF determined from its configuration space. For example, if the actual memory size of any of the VF BARs is lower than an advertised supported page size. In such cases, this property must be set as per the highest advertised supported page size value smaller or equal to the smallest VF BAR aperture across all PFs.

SRIOV tests randomly configure the SRIOV extended capability structure and enable Virtual Functions (VF) in all existing Physical Functions (PF). By default, the enumeration (for all tests that invoke it) also does this in the `svt_pcie_device_system_virtual_activate_link_sequence`, if enumeration is enabled and `SVT_PCIE_DISABLE_SRIOV` command-line switch is not provided. You can use `svt_pcie_ts_device_system_virtual_sriov_vf_enable_sequence` to configure the SRIOV extended capability structure in all PFs randomly according to their respective capabilities and user-configurable attributes in test suite configuration. This sequence also updates members of the test suite status class (`svt_pcie_ts_tl_status`) which are used by enumeration constraints.

Enumeration programs VF BARs either with random address values (by default) or with contiguous address locations (if the incremental mode of BAR allocation is enabled by setting the test suite configuration, `enable_incremental_bar_allocation` to 1). Enumeration also ensures that all VF BAR ranges are exclusive. If the device supports a large number of VFs and has very large BAR registers, then there is a chance that the memory space might run out during enumeration. In this case, if random mode of BAR address allocation is selected, then the test exits with a constraint error and you must configure SRIOV related parameters like `min_num_vfs_per_pf`, `max_num_vfs_per_pf`, and `max_num_vfs_overall` in the test suite configuration. When DUT supports large number of VFs, it is advisable to set the test suite

configuration, `enable_incremental_bar_allocation` to 1. This setting ensures that there is no unused address space which may be the case with randomized allocation. If the memory space runs out even during incremental BAR allocation mode, then no VFs are enabled thereafter. Note that based on the specification, it is not a valid DUT configuration if it supports so many VFs that the required memory space cannot be allocated for all of them.

6.6.2 Exercising Virtual Functions

All memory requests from RC will also be able to target the memory space of VFs randomly. Similarly, most tests that initiate a transaction from EP will randomly select a requester from enabled PFs and VFs. Whenever a test needs to read a register from configuration space of a VF whose routingID >= 16'h0100—that is, the VF resides at a bus-number higher than the device's captured bus-number, then a Type1 CfgRd/Wr request will be issued from RC VIP, as dictated by SRIOV specification.

6.6.3 Bus Master Enable for Virtual Functions

By default, enumeration does not enable the Bus Master Enable (BME) bit in any VF even though it is required as per specification. This is because enumeration time can increase significantly if there are a large number of enabled VFs. If the EP DUT implements the functionality of the BME bit for VFs in the core instead of imposing the requirement on application layer, then you have the following two options:

- Set `enable_vf_enumeration` in test suite configuration.

  OR

- Enable BME from backdoor in your application whenever a test issues a request to be transmitted from VF.

6.7 ATS

Following are the primitive sequences developed for ATS, as part of the TLP sequence collection. All of them are blocking on successful transmission/receipt of TLP by RC. All ATS tests use these sequences. It is also required to drop all implicitly generated CplDs by the target layer (via callback) of RC VIP as translation completions should be explicitly started from the complex sequences.

- svt_pcie_ts_tlp_ats_mem_request_sequence: For memory requests which need to use the AT field.
- svt_pcie_ts_tlp_translation_completion_sequence: For translation completion required in response to a translation request by EP.
- svt_pcie_ts_tlp_invalidate_request_sequence: For random Invalidation Request request message.
- svt_pcie_ts_tlp_invalidate_completion_sequence: For Invalidation Completion required in response to invalidation request by RC VIP (only used to explicitly transmit from EP VIP in VIP-VIP mode).
- svt_pcie_ts_tlp_page_request_message_sequence: For random page request message.
- svt_pcie_ts_tlp_prg_response_message_sequence: For PRG response message in response to page request by EP DUT.

In case the EP DUT core implements its own internal TAG management, then the actual value used needs to be copied back into the outbound TLP transaction object so that the ATS complex sequences have access to that value when explicitly starting translation completions.

6.8 Multi-Function

The driver_app and tlp base sequences/ extended transaction classes have constraints which in case of a memory/IO transaction from RC randomize it so as to target address range of an existing Function (function/PF/VF). The tlp base sequences/ extended transaction classes additionally constrain requester-id field to that of an existing function. All TL tests utilize this infrastructure to randomly target all functions of the EP device.

6.9 Advanced Error Reporting Testing

The Test Suite includes the necessary tools that assists you in developing new Advanced Error Reporting (AER) tests. This is provided in the base sequence named `svt_pcie_ts_device_system_virtual_tl_main_aer_sequence`. The sequence includes all currently supported AER checks and provides 'callbacks' so that you can use this as a method to inject the actual error and add any additional logic if required. Your AER tests must extend this class and implement the required callbacks like `inject_error()` task.

Note These 'callbacks' are not related to the VIP callback feature. They are simply virtual methods that you can override to extend the implementation.

AER mask and severity registers are programmed with random values in each iteration and AER checks are performed after the `inject_error()` user callback returns. A complete account of the injected stimulus and checks performed can be found in the HTML class reference of `svt_pcie_ts_device_system_virtual_tl_main_aer_sequence`.

The Test Suite also delivers a number of AER tests whose test sequences extend from this sequence. Therefore, all those tests are capable of performing all the AER checks listed in the HTML class reference and can also be used as a reference to develop your own AER tests.

Following AER-related primitive sequences are also delivered in case you want to use them.

Table 6-3 AER Primitive Sequences

Sequence	Purpose
svt_pcie_ts_device_system_virtual_is_aer_supported_sequence	AER capability availability for particular function.If AER capability present in function then is_aer_present bit will be set to 1 otherwise is aer_present bit will be set to 0.
svt_pcie_ts_device_system_virtual_aer_register_read_sequence	Used to read 32-bit value in AER register with offset provided for particular function.if read done successfully then sequence updates read_done =1 or read_done = 0.
svt_pcie_ts_device_system_virtual_aer_register_write_sequence	Used to write 32-bit value in AER register with offset provided for particular function. if write done successfully then sequence updates write_done =1 or write_done = 0.
svt_pcie_ts_device_system_virtual_aer_register_compare_sequence	Used to compare bit in AER register with offset provided for particular function.Fatal error if provided values mismatches. or success bit set if value matches.

6.10 Readiness Notification

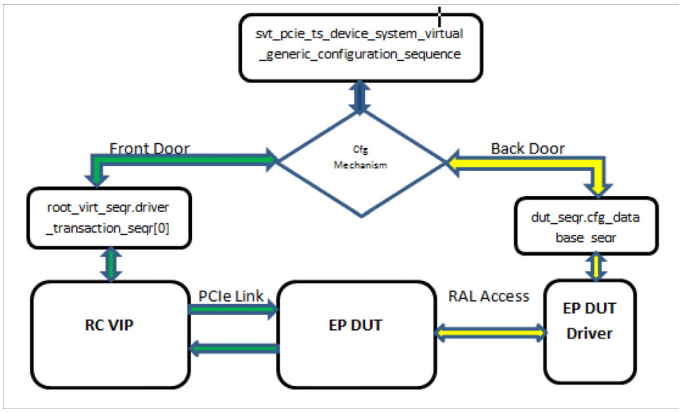
Readiness Notification tests expect DRS/FRS messages from the Endpoint DUT after DRS/FRS events within maximum allowed time. For FRS events, if the expected time values are found in the RN extended capability header then they are used. In all other cases, the values in the 'SVT_PCIE_TEST_SUITE_MAX_DRS_LATENCY_AFTER_DRS_EVENT' and 'SVT_PCIE_TEST_SUITE_MAX_DRS_LATENCY_AFTER_FRS_EVENT' replaceable macro defines are used which the user can set accordingly.

6.11 Backdoor Configuration Access

If backdoor access to the configuration space of DUT is available, you may optionally provide the `SVT_PCIE_ENABLE_BACKDOOR_CFG_UPDATES=1` command-line switch. This will cause configuration accesses to go on `cfg_database_sequencer` of DUT. Its implementation must be provided by DUT driver. This control is available for tests which have been enhanced to use generic configuration access instead of front door CFG TLP transactions directly by using `svt_pcie_ts_device_system_virtual_generic_configuration_read_sequence/svt_pcie_ts_device_system_virtual_generic_configuration_write_sequence` primitive sequences. All enumeration sequences have also been enhanced to use generic configuration accesses as described here.

The following figure shows both front and backdoor access.

Figure 6-1 Front and Backdoor Access



6.12 Secondary PCI Express Extended Capability (SEC)

The current set of test cases verifies the following registers for the DUT as an UPSTREAM device only.

- Secondary PCI Express Extended Capability Header
- Lane Error Status Register
- Optional feature: lane error due to 8b10b decode error, Test case : pl_8b_10b_enc_cfg_error
- Framing error.
- Equalization Control Register

6.13 Custom Applications

The Application Agent allows you to build your own application components and add them to the Application Agent just like existing Driver/Target applications. These applications will be able to receive

inbound TLPs and schedule outbound TLPs through the Application Agent. Note that all outbound TLPs destined for transmission from the DUT need to pass through the TLP Mapper component present in the Application Agent.

The following example demo file includes all the required steps:
$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/pcie_test_suite_custom_tlp_mapper_svt/sverilog/src/vcs/svt_pcie_test_suite_custom_tlp_mapper.svp

The last two steps of this process are shown in the t*b_dut_pcie/env/snps_pcie_link_env.sv* example class.

Following are the required steps:

1. Create a callback class which decides what inbound TLPs (if any) need to be routed to the your custom application for processing instead of the default target: which the tlp_mapper class would otherwise determine. Note that any inbound TLP can only be mapped to 1 application ID by the TLP Mapper.
2. Add TX and RX TLM ports in your custom application to receive inbound TLPs and/or to schedule outbound TLPs.
3. Assign a unique application ID (> 32) to each of your custom applications. Note that first 32 `application_ids` are reserved for internal use.
4. If the custom application also needs to schedule outbound TLPs from the DUT, then add logic to push the transactions on the `tx_tlp_out_port` TLM put port of the custom application.
5. If the custom application also needs to receive inbound TLPs to the DUT, then add an implementation for the put method supporting '`rx_tlp_in_export`' export of the custom application. This method will be called every time the Application Agent receives an inbound TLP mapped to this application and it has the reference of the TLP object of interest.
6. Extend the TLP Mapper to start your custom application, create new ports and to connect with those in the custom application.
7. Create your custom application. Create new ports indexed by the unique `application_id` chosen for your custom application in step 3.
8. Connect the TLM ports created in steps 2 and 7.
9. Create an object of the callback class created in step 1, and add it to the pool as shown below.
10. Include the source file which contained the classes created in steps 1-9.
11. Establish factory override so that the type of `tlp_mapper` object created in Application Agent is that of the class created in step 6.

6.14 DUT Dependent Test Cases

The following table lists DUT dependent cases.

Table 6-4 DUT Dependent Cases

Test cases Name	DUT dependency
gen3_pl_config_complete_to_detect_error gen3_pl_recovery_rcvrcfg_to_detect_speed_ change_1_error gen3_pl_recovery_rcvrcfg_to_detect_ts2_non_ matching_rate_error	The following test cases verify features where the LTSSM state machine needs to loop for 255 times. The idle_to_rlock variable needs to be incremented until the value of 255. Due to this iteration, tests exceed the default test timeout value. As a result, all these following test cases timeout in the regression, and report themselves as failed. You need to increase the default test case timeout values, and run these test cases separately to get the coverage
gen3_pl_recovery_equalization_phase2_to_recovery_speed_ timeout_case1_error	Test expect to initiate preset request from EP DUT in Recovery.Equalization Phase 2 state.

Example 6-1 Example of Dependent Code

```
cfg.ep_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_entry_valid = 16'h7FF;
 cfg.ep_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_table[0]  =
{6'h0C,6'h24,6'h00};
 cfg.ep_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_table[1]  =
{6'h00,6'h28,6'h08};
----------
----------
 cfg.ep_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_table[10] =
{6'h0C,6'h24,6'h00};


 cfg.ep_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_entry_valid = 16'h7FF;
 cfg.ep_cfg.pcie_cfg.pl_cfg.expected_preset_to_coefficients_mapping_table[0]  =
{6'h0C,6'h24,6'h00};
 cfg.ep_cfg.pcie_cfg.pl_cfg.expected_preset_to_coefficients_mapping_table[1]  =
{6'h00,6'h28,6'h08};
----------
----------
 cfg.ep_cfg.pcie_cfg.pl_cfg.expected_preset_to_coefficients_mapping_table[10] =
{6'h0C,6'h24,6'h00};
```

6.15 SRIS Usage

The following sections contains the usage of SRIS and SRNS.

6.15.1 SRIS

SRIS allows a worst case 5600 PPM difference (SSC with 5000 PPM difference + Tx/Rx crystal tolerance 600ppm). Following is the command to enable SRIS:

```
gmake <test_name> SVT_PCIE_SSC_MODE=SSC_ON_PPM_ON  SVT_PCIE_MAX_PPM=<+/- max osc ppm
value>  SVT_PCIE_MIN_PPM=< +/- min osc ppm value >
```

The command line option “SVT_PCIE_SSC_MODE=SSC_ON_PPM_ON” will introduce additional 5000PPM in clock as SSC.

For the oscillator frequency, the test will pick a random PPM value from the range provided by you in `SVT_PCIE_MIN_PPM` and `SVT_PCIE_MAX_PPM`.

The SKIP transmission interval requirement is handled by the VIP internally.

6.15.2 SRNS

The following command line is used to enable SRNS (Tx, Rx Refclk rates differ, allowing a worst case 600 PPM):

```
gmake <test_name> SVT_PCIE_SSC_MODE=SSC_OFF_PPM_ON  SVT_PCIE_MAX_PPM=<+/- max osc ppm
value>  SVT_PCIE_MIN_PPM=< +/- min osc ppm value >
```

The SSC_OFF_PPM_ON parameter enables SRNS mode; that is, there will not be any spread spectrum clocking.

For the oscillator frequency, the test will pick a random PPM value from the range provided by you in `SVT_PCIE_MIN_PPM` and `SVT_PCIE_MAX_PPM`.

The range should be within -600 PPM to +600 PPM which is the allowable tolerance limit for clock. If you want to set a specific/particular crystal PPM, then you must give the same value for min and max PPM command-line options.

For example, if you want to set -500 PPM then, the following must be given.

```
SVT_PCIE_MAX_PPM = -500 & SVT_PCIE_MAX_PPM = -500
```

You provide a value within the specification defined limit of +/-600PPM.

By default, the value will be chosen as +600PPM.

6.16 Loopback Usage

Loopback is applicable to SERDES mode only. As a result, Loopback cases are added in to serdes testbench areas only.

6.17 User Controllable Tolerance for LTSSM Time Outs

You can set the controllable tolerance value for all LTSSM time outs.

You must provide the percentage of the tolerance limit (maximum allowable tolerance is +50% as per specification) in base test as shown below:

```
// Set up the user defined percentage value for tolerance timeout on timout
//(2ms, 12ms, 24ms, 48ms ).
uvm_config_db#(int)::set(this, "env.test_env_seqr",
                      "ltssm_timeout_tolerance_percentage", 50);
```

The default value is set to 50% of the timeout.

6.18 Transaction Ordering

The Transaction Ordering tests utilize the Ordering Application available as part of PCIe SVT VIP to perform Ordering rule related checks on the outbound traffic initiated from DUT. The tests initiate traffic to validate specific transaction ordering rules and the Ordering Application keeps track of all the scheduled TLPs. The application is added as a 'custom application' inside the TLP Mapper component. This is done by

extending it and following the rules as described in section Custom Applications. The application compares the order in which VIP receives TLPs from the DUT with the scheduling order to validate two things:

- That the DUT does not violate any of the transaction Ordering Rules.
- That the DUT does not block TLPs eligible for transmission as per transaction Ordering Rules.

The Ordering Application may also be optionally used to perform Re-Ordering before the DUT. This can be enabled by setting `enable_reordering` property of the `svt_pcie_ordering_app_configuration` class. Its reference is present in the Test Suite configuration object. Also, set other properties of this class to appropriate values for smooth operation.

Note that if the DUT is not able to perform Re-Ordering by itself and the Ordering Application is instead enabled to do so, then the following tests will not be applicable. This is because these tests involve changing the order of outbound completions (or in some cases just holding them back for some time) which is DUT-implementation specific and cannot be guaranteed.

- `tl_transaction_ordering_rule_a5a`
- `tl_transaction_ordering_rule_b5_c5`
- `tl_transaction_ordering_rule_d2a`
- `tl_transaction_ordering_rule_d2b_ro`
- `tl_transaction_ordering_rule_d2b_ido_reqid`
- `tl_transaction_ordering_rule_d3_d4`
- `tl_transaction_ordering_rule_d5`
- `tl_transaction_ordering_rule_stress_test`

Following tests which validate functionality of IDO bit in relation to transaction Ordering are only valid for a DUT with more than 1 requester IDs (for a given bus number) - i.e. a multi-function DUT or EP DUT with SRIOV support - supporting IDO.

- `tl_transaction_ordering_rule_d2b_ido_reqid`
- `tl_transaction_ordering_rule_a2b_ido_reqid`
- `tl_transaction_ordering_rule_b2b_c2b_ido_reqid`

Following tests which validate functionality of RO bit in relation to transaction Ordering are only valid for a DUT with at least 1 function with RO support.

- `tl_transaction_ordering_rule_a2b_ro`
- `tl_transaction_ordering_rule_b2b_c2b_ro`
- `tl_transaction_ordering_rule_d2b_ro`

The following tests require the DUT to be capable of transmitting AtomicOp transactions, in which case `enable_atomic_op_as_requester_support` bit must be set in the `dut_capabilities` configuration object.

- `tl_transaction_ordering_rule_b3_b4_c3_c4`

The PCIe Test Suite currently delivers 17 Transaction Ordering Rule tests and all of their names start with `tl_transaction_ordering_rule_` prefix.

6.19 Equalization Phase2 to Recovery.Speed Transition in VIP-to-IIP Mode

Perform the following steps if you are a SNPS IIP user.

1. In PIPE mode:

a. Invoke preset request from EP DUT.

b. Not required to configure anything as PHY is VIP (taken care in the test itself, additional settings are not required).

2. In SERDES mode:

a. In *IIP_DUT_PCIE_RC_x*.tcl*, you need to set the following variable:
`CX_GEN3_EQ_COEF_CONV_SUPPORTED=1`

b. Changes in *pcie_rc_dut_external_app_bfm.sv*.

```
task
`SVT_PCIE_TEST_SUITE_RC_DUT_EXTERNAL_APP_BFM_TYPE::equalization_setup(svt_pcie_device_c
onfiguration cfg);
  uvm_status_e status;
  uvm_reg_data_t data;
  uvm_reg      reg_obj;

`ifdef SERDES_TB
  if(cfg.pcie_cfg.pl_cfg.min_eq_preset_coeff_validation_delay[0] > 0) begin
    //[23:8] -> Preset Request Vector = 16'h0011
    //[3:0]  -> Feedback Mode.0000b: Direction Change = 4'h0
    write_reg("GEN3_EQ_CONTROL_OFF",32'h00001100, 32'h00FFFF0F);

    //[4:0] -> Minimum Time (in ms) To Remain EQ Master Phase.
    data = 32'h1f;
    write_reg("GEN3_EQ_FB_MODE_DIR_CHANGE_OFF",data, 32'h0000001F);
  end
  `endif

endtask: equalization_setup
```

c. Invoke preset request from EP DUT.

Perform the following steps if you are a Non-SNPS IIP user.

1. In PIPE mode:

- Invoke preset request from EP DUT.
- Not required to configure anything as PHY is VIP (taken care in the test itself, additional settings are not required).

2. In SERDES mode:

- Fire preset request from EP DUT.
- Configure EP evaluation timer same as phase2 timeout value so that EP continues evaluating coefficients and timeout will occur.

6.20 Changing the Link Width (Target Link Width Option)

You can use the following command-line option to control the target link width.

```
SVT_PCIE_TARGET_LINK_WIDTH
```

It can be passed with the value “1/2/4/8/12/16/32” The Link width for RC and EP will be configured according to the passed value. But the passing value should be equal to or less than the value of `SVT_PCIE_MAX_LINK_WIDTH`. The following table shows the relationship between `SVT_PCIE_TARGET_LINK_WIDTH` and `SVT_PCIE_MAX_LINK_WIDTH`.

SVT_PCIE_MAX_LINK_WIDTH	SVT_PCIE_TARGET_LINK_WIDTH	Final Link-up
Any possible value from : 4/16/32 Example: 32	1/2/4/8/12/16/32	Link-up will be based on Target link width value
Not given in command line Default is 16)	Any possible value from: 1/2/4//8/12/16	Link-up will be based on Target link width value
Any possible value from: 4/16/32.	Not given in command line	Link-up will be based on Max link width value
Value less then Target link width Example: 4	16	Fatal Error

In this release, `SVT_PCIE_TARGET_LINK_WIDTH` option support is provided only to PIPE TB area.

6.21 Verdi Spec Linking Feature Usage

PCIe Test Suite delivers the test plans based on Verdi specification linking feature. The tests verifying the DUT PCIe protocol features are linked to the specification document. You can open plans in Verdi and view the mapping specification in Verdi GUI.

The plans also support back annotation of test pass/fail metric based on the regression results.

6.21.1 PCIe Test Suite Plan Deliverables

6.21.1.1 Directory

The plans are available at the following location in the installation directory:

$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/version/doc/VerificationPlans/

It is recommended to save a copy of these plans in your local directory before loading the plans into Verdi Coverage tool.

6.21.1.2 Naming Convention for Plans

6.21.1.2.1 Sub-plans

Sub-plans are created by copying the corresponding template and renaming as mentioned below.

```
< Protocol >_dut_<dut_type>_<subplan_name>_<plan_category>_subplan
```

Where,

Protocol = name of the VIP

dut_type = type of dut. Example: ep | rc

plan_category = test

subplan_name = can be one of following as per the level at which a sub plan is present in test plan hierarchy:

At Level2: <<spec name>_<chapter _name>>

Example:

```
pcie_dut_ep_pci_express_base_spec_r3_1_power_management_test_subplan.hvp
```

At Level3: <<chapter _name>_<functionality>>

Example:

```
pcie_dut_ep_power_management_test_subplan.hvp
```

Note You can organize sub-plans in a hierarchy. For example, a sub-plan can be created for each chapter/layer in the specification and each of these sub-plans may have multiple sub-plans available for a specific functionality.

- Use name as `spec_name` and layer name if the sub-plan is for a specific layer.
- Use `layer_name` and functionality name if the sub-plan represents a specific functionality described by the layer. This sub-plan may be the sub-plan of layer-wise sub-plan.

Examples:

```
pcie_dut_ep_pci_express_base_spec_r3_1_transaction_layer_test_subplan
pcie_dut_ep_transaction_layer_aer_test_subplan
pcie_dut_ep_transaction_layer_flow_control_test_subplan
```

6.21.1.2.2 Top Level Plan Naming Convention

A top level plan is the plan which includes more than one sub-plans. These sub-plans might have been linked to different specifications.

Each plan category will have a top level plan.

```
< protocol>_dut_<dut_type>_<plan_category>_toplevel_plan
```

Where,

protocol = pcie

dut_type = ep | rc

plan_category = test

Example:

```
pcie_dut_ep_test_toplevel_plan
```

6.21.2 Requirements to Setup the Verdi Spec Linking Flow

- Verdi tool Supported: verdi_2015.09 onwards
- Download the PCIE specification documents from the PCIE-SIG with the version mention below:
  - PCIe Gen3: PCI Express® Base Specification Revision 3.1, October 8, 2014
  - PCIe Gen4: PCI Express® Base Specification Revision 4.0 Version 0.5 February 6, 2015
  - SRIOV (Single Root I/O Virtualization and Sharing Specification Revision 1.1)
  - ATS (Address Translation Services Revision 1.1)
- Test pass/Fail Results text file inclusion.

- The Verdi tool need the pass/fail status from the Regression results. Use the regression results to produce the test status information in a text file. Use the first line of the text file as it is, following that is the test file status information. In the text file, HVP metric = test is the first line, the test is a in-built Verdi metric of type enum with possible values of pass, fail, unknown.

```
HVP metric = test
test_case_name = pass|fail|unknown|...
```

Example:

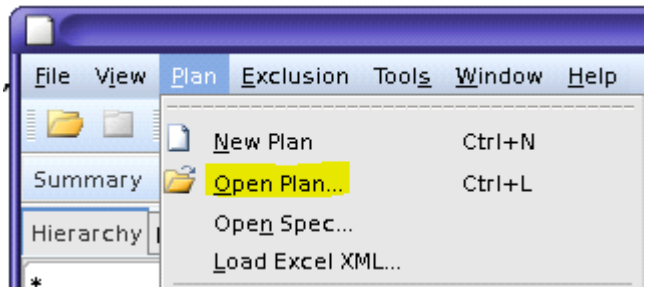
```
HVP metric = test
dl_ack_collapsed = pass
dl_ack_nack_latency_timer = fail
```

6.21.3 Commands and Guidelines to View the Test Suite with Specification

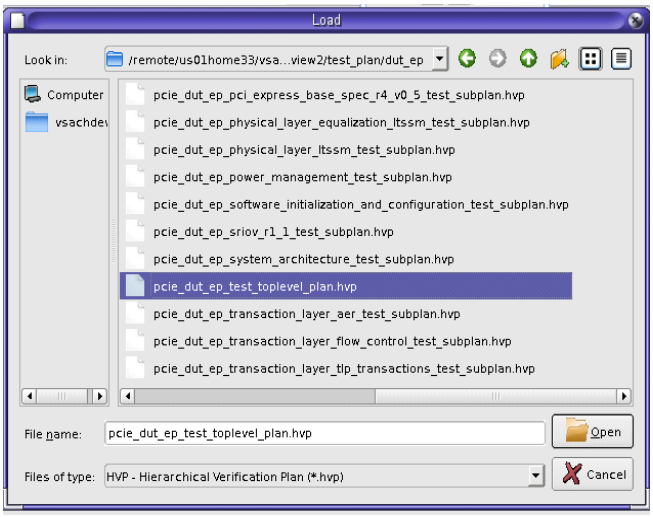
- Invoke Verdi
  - To invoke Verdi Coverage, you need to add the `-cov` option to the Verdi command line.

```
> verdi -cov &
```

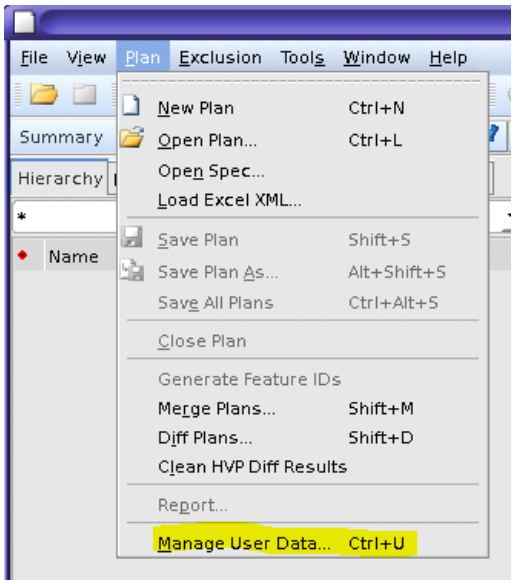
- Select the Open Plan option from the Plan menu as shown in the following figure.



- The Load dialog box appears. To import the Hierarchical Verification plan (.hvp) file, navigate to the directory and click Open after selecting the plan file. The following figure shows selection of EP top level *.hvp* file (*pcie_dut_ep_test_toplevel_plan.hvp*).

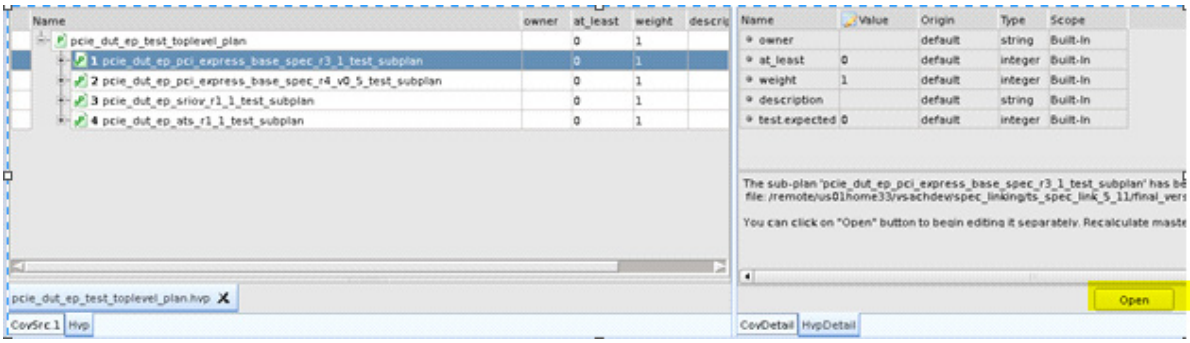


- Select the Manage User Data option from the Plan menu to load the regression result text file (Pass/Fail status information).

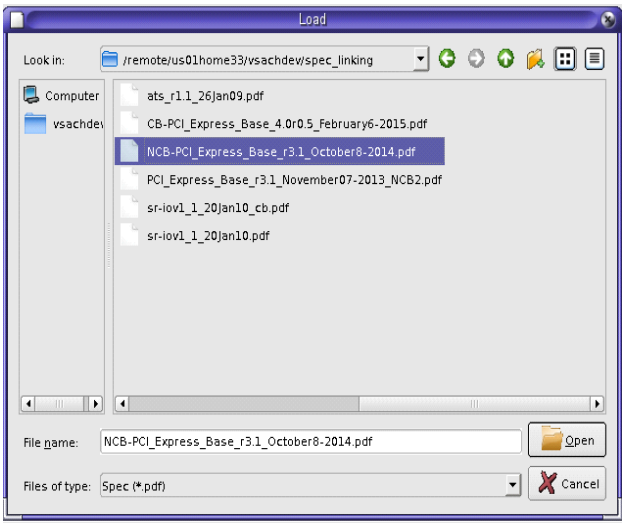


  - The Load/Unload User Data dialog box appears. To import the regression results file, click the Load button. Click OK after selecting the text file.

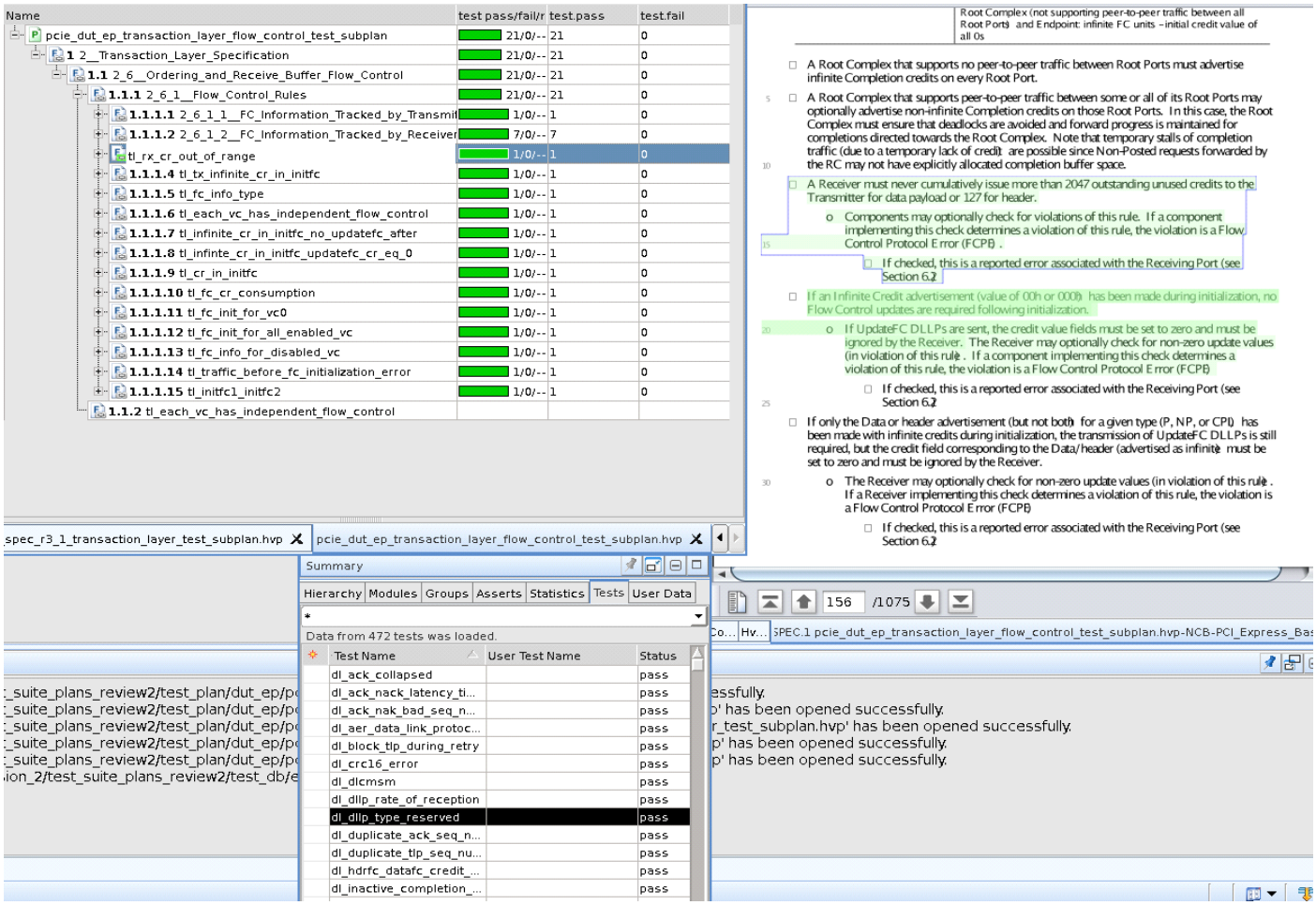
- In the Hvp dialog box, select the *.hvp* file you want to load from the spec linked tool and click Open.



  - The Load dialog box appears. Select the specification version you want to load and click Open to open the Verdi spec linked tool.



  - The following figure shows the complete view of Verdi spec linked tool which includes the Test back annotation results (Summary), HVP sub-plan and specification which covers the attributes for the respective test (highlighted in green).



For more details, refer to Verdi Coverage User Guide and Tutorial.

6.22 Enabling EIEOS Pattern for Gen4

Added support for new 16G EIEOS pattern:

- To enable 16G EIEOS pattern for Gen4, set the `SVT_PCIE_TEST_SUITE_ENABLE_EIEOS_16G_0000_FFFF` macro to 1. This will invoke VIP to transmit and expect new 16G EIEOS pattern. By default, it is set to 0.

```
`define  SVT_PCIE_TEST_SUITE_ENABLE_EIEOS_16G_0000_FFFF 1
```

6.23 Test Suite Specific Command-Line Options

The following table lists command-line options.

Attention The Enumeration feature is not supported for demo (VIP-VIP) mode. As a result, you can not use the switch `SVT_PCIE_ENABLE_ENUMERATION` in demo mode. Using `SVT_PCIE_ENABLE_ENUMERATION`, you can enable or disable enumeration for an EP DUT in VIP-IIP mode.

.

Table 6-5 Test Suite Specific Command Line Option

Command Line Option	Corresponding plusarg	Description	Interface type	Test case Applicable
SVT_PCIE_MAX_LINK_ WIDTH =4		Select (x1/x2/x4/x8/x12/x16/x32) link_width configuration (Default is x4)	PIPE/Serial	All Test
SVT_PCIE_ENABLE_ TRANSACTION_ LOGGING=1	+svt_pcie_enable_transaction_logging	To enable transaction logging and Symbol logging	PIPE/Serial	All Test
SVT_PCIE_ENABLE_PA=1	+svt_pcie_enable_pa	Enable protocol analyzer xml generation	PIPE/Serial	All Test

SVT_PCIE_ENABLE_COVERAGE=1	+svt_pcie_enable_coverage	Enables the coverage. This is a five-bit variable and each layer corresponds to unique layer.
		• [4] bit when set to 1'b1 enables coverage for LTSSM checks.
		• [3] bit when set to 1'b1 enables coverage in TL Layer.
		• [2] bit when set to 1'b1 enables coverage in DL Layer.
		• [1] bit when set to 1'b1 enables coverage in PL Layer.
		• [0] bit when set to 1'b1 enables coverage for PIPE.
		For example,
		`SVT_PCIE_ENABLE_COVERAGE = 5'b00001 // To enable PIPE coverage`
		`+ svt_pcie_enable_coverage = 5'b00001 // To enable PIPE coverage`
			PIPE/Serial	All Test

SVT_PCIE_ENABLE_ ENUMERATION=1	+svt_pcie_enable_enumeration	To run enumeration sequence only in DUT mode, not applicable for demo mode.	PIPE/Serial	All Test (All TL, DL and few PL)
SVT_PCIE_ENABLE_ BACKDOOR_CFG_ UPDATES=1	+svt_pcie_enable_backdoor_cfg_updates	This option will cause tests to initiate cfg_database_service_transactions for doing read/write operations on DUT's configuration registers instead of frontdoor CfgRd/Wr on PCIe Link.	PIPE/Serial	All Test*

Table 6-5 Test Suite Specific Command Line Option (Continued)

Command Line Option	Corresponding plusarg	Description	Interface type	Test case Applicable
SVT_PCIE_LINK_SPEED=<GEN1|GEN2|GEN3|GEN4|GEN5>	+svt_pcie_link_speed=<GEN1|GEN2|GEN3|GEN4|GEN5>	Force a test to run on gen1/gen2/gen3/gen4/gen5 speed **Note**: Gen5 is applicable only for serial mode.	PIPE/Serial	Specific cases**
SVT_PCIE_TEST_TIMEOUT		To set the test stop time (default 1200000ns)	PIPE/Serial	All Test
SVT_PCIE_ENABLE_POLARITY_INVERSION	+svt_pcie_enable_polarity_inversion	To enable the polarity inversion feature	Serial	All Test
SVT_PCIE_SSC_MODE=<SSC_OFF_PPM_ON|SSC_ON_PPM_ON>	+svt_pcie_ssc_mode<SSC_OFF_PPM_ON|SSC_ON_PPM_ON>	To enable the SSC mode SSC_OFF_PPM_ON: Provides Positive or Negative ppm(+/- 600) will be randomly chosen SSC_ON_PPM_ON: Provides Positive or Negative ppm(+/- 600) will be randomly chosen and default 0 to -5000ppm for SSC.	Serial	All Test
SVT_PCIE_DISBALE_SRIOV=1		If the DUT supports SRIOV but it is desired that TL layer tests not target VFs randomly, then 'SVT_PCIE_DISBALE_SRIOV=1' command line switch should be provided. If this switch is provided then the activate_link sequence will not enable VFs in the SRIOV extended capability structure(s).	PIPE/Serial	All TL cases
SVT_PCIE_ENABLE_EXTERNAL_APP_BFM=1		Enables the External Application BFM Use Model	PIPE/Serial	All tests
SVT_PCIE_ENABLE_LANE_SKEW=1		User can enable lane to lane skew insertion feature in random lanes with spec. defined allowable limit.	PIPE/Serial	All tests
SVT_PCIE_TARGET_LINK_WIDTH=1/2/4/8/12/16/32	+svt_pcie_target_link_width	Whatever value is passed to this command line option, the RC and EP is configured in that link width. But, that value should be equal to or less than SVT_PCIE_MAX_LINK_WIDTH. If not, then test case will exit with FATAL Error	PIPE	All tests

Table 6-5 Test Suite Specific Command Line Option (Continued)

Command Line Option	Corresponding plusarg	Description	Interface type	Test case Applicable
SVT_PCIE_PIPE_WIDTH	+svt_pcie_pipe_width	To enable the different PIPE width configuration. (Possible value 0, 1, and 2). to SVT_PCIE_PIPE_WIDTH to Value 0, 1 & 2 means PIPE_WIDTH of 8, 16 & 32 respectively. In demo mode this pipe width fixed for both side EP as well as RC side, while in DUT mode, this pipe width set for VIP. By default, PIPE width is constant in X4, and its dynamic for X16 mode.	PIPE	All tests
SVT_PCIE_TS_AXI_WIDTH		To Enable AXI interface with either 128-bit or 32-Bit. Value <32|128> to this option, corresponds to width AXI Bit.	PIPE/Serial	All tests
SVT_PCIE_TEST_SUITE_PIPE_SPEC_VER_4_2_PCLK_INPUT		To enable PIPE Spec 4.2 with PCLK as PHY Input.	PIPE	All tests
SVT_PCIE_TEST_SUITE_PIPE_SPEC_VER_4_3		To enable PIPE 4.3 support in Test Suite	PIPE	All tests
SVT_PCIE_ENABLE_10_BIT_TAGS		To enable 10-bit tag support from VIP.	PIPE/Serial	All Tests
SVT_PCIE_MAX_PPM	+svt_pcie_max_ppm	This introduces the maximum oscillator ppm in Tx Bit clk.Testbench will choose a value between svt_pcie_max_ppm and svt_pcie_min_ppm randomly and introduce that ppm shift in Tx bit clk.	Serial	PL cases
SVT_PCIE_MIN_PPM	+svt_pcie_min_ppm	This introduces the minimum oscillator ppm in Tx Bit clk.Tes bench will choose a value between svt_pcie_max_ppm and svt_pcie_min_ppm randomly and introduce that ppm shift in Tx bit clk.	Serial	PL cases

Table 6-5 Test Suite Specific Command Line Option (Continued)

Command Line Option	Corresponding plusarg	Description	Interface type	Test case Applicable

SVT_PCIE_ENABLE_CHECK_COVERAGE_EP	+svt_pcie_enable_check_coverage_ep	Enables the check coverage for Endpoint VIP. This is a 4-bit variable and each bit corresponds to unique layer.
		• [3] bit when set to 1'b1 enables check coverage in TL Layer.
		• [2] bit when set to 1'b1 enables check coverage in DL Layer.
		• [1] bit when set to 1'b1 enables check coverage in PL Layer.
		• [0] bit when set to 1'b1 enables check coverage for PIPE.
		For example,
		`SVT_PCIE_ENABLE_CHECK_COVERAGE_EP = 4'b0001 // To enable PIPE check coverage for EP`
		`+ svt_pcie_enable_check_coverage_ep = 4'b0001 // To enable PIPE check coverage for EP`
			PIPE/Serial	All Tests

SVT_PCIE_ENABLE_CHECK_COVERAGE_RC	+svt_pcie_enable_check_coverage_rc	Enables the check coverage for Root VIP. This is a 4-bit variable and each bit corresponds to unique layer.
		• [3] bit when set to 1'b1 enables check coverage in TL Layer.
		• [2] bit when set to 1'b1 enables check coverage in DL Layer.
		• [1] bit when set to 1'b1 enables check coverage in PL Layer.
		• [0] bit when set to 1'b1 enables check coverage for PIPE.
		For example,
		`SVT_PCIE_ENABLE_CHECK_COVERAGE_RC = 4'b0001 // To enable PIPE check coverage for RC`
		`+ svt_pcie_enable_check_coverage_rc = 4'b0001 // To enable PIPE check coverage for RC`
			PIPE/Serial	All Tests

Table 6-5 Test Suite Specific Command Line Option (Continued)

Command Line Option	Corresponding plusarg	Description	Interface type	Test case Applicable

SVT_PCIE_ENABLE_CHECK_COVERAGE_PASS	+svt_pcie_enable_check_coverage_pass	Enables Pass only check coverage when check coverage is enabled. This is a 4-bit variable and each bit corresponds to unique layer.
		• [3] bit when set to 1'b1 enables Pass check coverage in TL Layer.
		• [2] bit when set to 1'b1 enables Pass check coverage in DL Layer.
		• [1] bit when set to 1'b1 enables Pass check coverage in PL Layer.
		• [0] bit when set to 1'b1 enables Pass check coverage for PIPE.
		For example,
		`SVT_PCIE_ENABLE_CHECK_COVERAGE_PASS = 4'b0001 // To enable PIPE Pass check coverage`
		`+ svt_pcie_enable_check_coverage_pass = 4'b0001 // To enable PIPE Pass check coverage`
			PIPE/Serial	All Tests

SVT_PCIE_ENABLE_CHECK_COVERAGE_FAIL	+svt_pcie_enable_check_coverage_fail	Enables Fail only check coverage when check coverage is enabled. This is a four-bit variable and each bit corresponds to unique layer.
		• [3] bit when set to 1'b1 enables Fail check coverage in TL Layer.
		• [2] bit when set to 1'b1 enables Fail check coverage in DL Layer.
		• [1] bit when set to 1'b1 enables Fail check coverage in PL Layer.
		• [0] bit when set to 1'b1 enables Fail check coverage for PIPE.
		For example,
		`SVT_PCIE_ENABLE_CHECK_COVERAGE_FAIL = 4'b0001 // To enable PIPE Fail check coverage`
		`+ svt_pcie_enable_check_coverage_fail = 4'b0001 // To enable PIPE Fail check coverage`
			PIPE/Serial	All Tests

** These tests are: • For tl_*/pl_*/dl_* cases, all Gen1/Gen2/Gen3/Gen4 speed is applicable • For gen2_pl_* cases, Gen2, Gen3, and Gen4 speed applicable • For gen3_pl_* cases, only Gen3/Gen4 speed applicable • For gen4_pl_* cases, only Gen4 speed applicable • For gen5_* cases, only Gen5 speed applicable				

Example:

```
gmake <test_case_name> USE_SIMULATOR=<simulator_type> SVT_PCIE_MAX_LINK_WIDTH=<4|16|32>
```

6.24 Partition Compile Support

Partition Compile is a VCS MX feature that allows you to compile portions of the design getting significantly faster turnaround times during the iterative process of compile and recompile.

With Partition Compile, you specify partitions in your design that you expect to revise and recompile often. VCS MX recompiles only the modified partitions.

The PCIe Test Suite provides a partition compile configuration file (*pc.optcfg*) which has partitions defined for the Synopsys Test Suite environment. You can add adjust partitions for their DUT to fine tune the partitioning.

Following are the Synopsys provided partitions.

```
pc.optcfg for serial testbenches(tb_dut_*_serdes):
    partition package svt_pcie_test_suite_uvm_pkg;
    partition package svt_pcie_uvm_pkg;
    partition cell pciesvc_device_serdes_x4_model_8g;
    partition cell pciesvc_device_serdes_x16_model_8g;
    partition cell pciesvc_device_serdes_x32_model_8g;
    partition cell pciesvc_device_application_model;
    partition cell <Top DUT module name>;


pc.optcfg for serial testbenches(tb_dut_*_pipe):
    partition package svt_pcie_test_suite_uvm_pkg;
    partition package svt_pcie_uvm_pkg;
    partition cell pciesvc_device_mpipe_x4_model_8g;
    partition cell pciesvc_device_spipe_x4_model_8g;
    partition cell pciesvc_device_mpipe_x16_model_8g;
    partition cell pciesvc_device_spipe_x16_model_8g;
    partition cell pciesvc_device_mpipe_x32_model_8g;
    partition cell pciesvc_device_spipe_x32_model_8g;
    partition cell pciesvc_device_application_model;
    partition cell <Top DUT module name>;
```

You can run tests in the partition compile mode using the `USE_SIMULATOR=vcspcvlog` command-line switch.

6.25 Link Width Dependent Tests

Table 6-6 shows the Link Width dependent tests.

Table 6-6 Link Width Dependent Tests

S.No.	Test Name	Not applicable for Link Width
1	gen3_pl_recovery_equalization_phase0_to_phase1_case1_error	1
2	gen3_pl_recovery_equalization_phase0_to_recovery_speed_timeout_case1_error	1
3	gen3_pl_recovery_equalization_phase1_to_rcvrlock_case1_error	1

Table 6-6 Link Width Dependent Tests

4	gen3_pl_recovery_equalization_lw_down_error	1
5	gen3_pl_recovery_equalization_phase3_to_rcvrlock_case1_error	1
6	gen4_pl_recovery_equalization_lw_down_error	1
7	gen4_pl_recovery_equalization_phase0_to_phase1_case1_error	1
8	gen4_pl_recovery_equalization_phase0_to_recovery_speed_timeout_case1_error	1
9	gen4_pl_recovery_equalization_phase1_to_rcvrlock_case1_error	1
10	gen4_pl_recovery_equalization_phase3_to_rcvrlock_case1_error	1
11	pl_config_linkwidth_start_to_config_linkwidth_accept_upconfigure_link_width_down_random	1
12	pl_polling_active_to_polling_compliance_partial_lanes_timeout_error	1
13	gen2_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_corruption_error	1
14	gen2_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_link_lane_mismatch_error	1
15	gen3_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_corruption_error	1
16	gen3_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_link_lane_mismatch_error	1
17	pl_l0_to_recovery_linkwidth_change_error	1
18	pl_config_linkwidth_start_to_config_linkwidth_accept_upconfigure_link_width_down_error	1
19	pl_config_linkwidth_start_to_config_linkwidth_accept_upconfigure_link_width_up_error	1
20	gen4_pl_logidle_corruption_on_multilane_in_configuration_idle_error	1
21	gen4_pl_logidle_corruption_on_multilane_in_recovery_idle_error	1
22	gen4_pl_logidle_corruption_on_multilane_in_l0_error	1
23	pl_config_lanenum_accept_to_config_lanenum_wait_link_width_down_case2_error	< 4
24	gen3_pl_multi_sdp_in_symbol_time_error	<= 4
25	pl_polling_active_to_polling_config_compliance_bit_0_timeout_error	1
26	pl_polling_active_to_polling_config_loopback_bit_1_timeout_error	1
27	pl_polling_active_to_polling_config_ts2_timeout_error	1
28	gen3_pl_idle_followed_by_no_idle_next_lane_error	1

6.26 Support for PIPE 4.3 and PIPE 4.2

The testbench supports the following topologies:

- PCLK as PHY Output (Normal mode)

- PCLK as PHY Input

Follow these steps to enable the topologies.

6.26.1 PCLK as PHY Output (Normal mode)

6.26.1.1 To Enable PIPE 4.2

This is the default mode in Test Suite. No changes or addition command-line options are required. All signals under "*PCIESVC_PIPE_SPEC_VER_GTR_4_0*" macro should be connected. This was already present in earlier release. There is no change in it.

6.26.1.2 To Enable PIPE 4.3

a. Command-line option:

You must pass the following command-line option:

```
SVT_PCIE_PIPE_SPEC_VER_4_3=1
```

b. DUT connection: In PIPE 4.3 mode, PowerDown is converted from a 3 bit to a 4 bit signal, and an extra port "AsyncPowerChangeAck" is added. You must connect these with the EP (MAC) DUT with following way.

Following is an example which shows an X16 connection with a DUT. Similarly it can be connected with a X16 and x32 Link width as well.

Example 6-2 X16 connection

```
`ifdef PCIESVC_PIPE_SPEC_VER_4_3
wire [3:0] endpoint0_powerdown;
wire endpoint0_async_power_change_ack;
`else
wire [2:0] endpoint0_powerdown;
`endif
//VIP connection
pcie_mpipe_x16_interconnect pcie_mpipe_x16_interconnect(
.reset (test_top.reset),
......
......
.powerdown (endpoint0_powerdown),
`ifdef PCIESVC_PIPE_SPEC_VER_4_3
.async_power_change_ack (endpoint0_async_power_change_ack),
`endif
 // DUT connection:
`ifdef PCIESVC_PIPE_SPEC_VER_4_3
assign
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_powerdow
n = <MAC output4-bit powerdown dignal>;
```

```
assign
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_async_po
wer_change_ack = <MAC output AsyncPowerChangeAck >
`else
assign
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_powerdow
n = <MAC output 3-bit powerdown dignal>;
`endif
```

6.26.2 PCLK as PHY Input

6.26.2.1 To Enable PIPE 4.2 PCLK as PHY Input

a. Command-line option:

You must pass the following command-line option:

```
SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT=1
```

b. DUT connection:

- ✧ All signals which is defined under “PCIESVC_PIPE_SPEC_VER_GTR_4_0”, should be connected.
- ✧ For PIPE 4.2 PCLK as PHY input, PCLK has become the input to the PHY and also there are two more signals “PclkChangeOk” (output of PHY) and “PclkChangeAck” (Input of PHY) added. Therefore, the connection will be as follows:

The following code shows a X16 connection:

Example 6-3 X16 connection

```
`ifdef SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT
      wire              endpoint0_pclk_change_ok;
      wire              endpoint0_pclk_change_ack;
`endif // ifdef SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT

`ifdef PCIESVC_PIPE_SPEC_VER_GTR_4_0
      wire              endpoint0_rx_eq_in_progress_0;
      wire              endpoint0_rx_eq_in_progress_1;
    ......
    ......
      wire              endpoint0_rx_eq_in_progress_15;
      wire [5:0]        endpoint0_lf_0;
      wire [5:0]        endpoint0_fs_0;
      wire [5:0]        endpoint0_lf_1;
      wire [5:0]        endpoint0_fs_1;
    ......
    ......
      wire [5:0]        endpoint0_lf_15;
      wire [5:0]        endpoint0_fs_15;
`else
      wire [5:0]        endpoint0_lf;
      wire [5:0]        endpoint0_fs;
`endif
// VIP connection
   pcie_mpipe_x16_interconnect pcie_mpipe_x16_interconnect(
            .reset                (test_top.reset),
```

```
`ifdef SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT
                            .pipe_clk_0           (endpoint0_pipe_clk),
                            .pipe_clk_1           (endpoint0_pipe_clk),
    ......
    ......
                            .pipe_clk_15          (endpoint0_pipe_clk),
                            .pclk_change_ok       (endpoint0_pclk_change_ok),
                            .pclk_change_ack      (endpoint0_pclk_change_ack),
`else
                           .pipe_clk             (endpoint0_pipe_clk),
`endif // ifdef SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT
              .......
              .......

`ifdef PCIESVC_PIPE_SPEC_VER_GTR_4_0
.rx_eq_in_progress_0  (endpoint0_rx_eq_in_progress_0),
.rx_eq_in_progress_1  (endpoint0_rx_eq_in_progress_1),
              .......
              .......
.rx_eq_in_progress_15  (endpoint0_rx_eq_in_progress_15),
.lf_0  (endpoint0_lf_0),
.fs_0  (endpoint0_fs_0),
.lf_1  (endpoint0_lf_1),
.fs_1  (endpoint0_fs_1),
              .......
              .......
.lf_15  (endpoint0_lf_15),
.fs_15  (endpoint0_fs_15),
`else
 .lf   (endpoint0_lf),
 .fs   (endpoint0_fs),
`endif

// DUT connection

`ifdef PCIESVC_PIPE_SPEC_VER_GTR_4_0
assign
{dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_rx_eq_in_progress_0,
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_rx_eq_in_progress_1,
. . . . .
            . . . . .
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_rx_eq_in_progress_15}
= <MAC output with PIPE 4.2 RxEqInProgress signal>;

assign
{dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_fs_0,
 dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_fs_1,
              . . . . .
              . . . . .
 dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_fs_15 }
  =<MAC output PIPE 4.2 FS signal>;

assign
{dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_lf_0,
 dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_lf_1,
              . . . . .
              . . . . .
 dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_lf_15}
  = <MAC output PIPE 4.2 LF signal>;
```

```
`else
    assign
     dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_fs

      = <MAC output FS signal>;
    assign
     dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_lf

      = <MAC output LF signal>;
`endif
```

6.26.2.2 To Enable PIPE 4.3 PCLK as PHY Input

a. Command-line options:

   You must pass the following command-line option:

   ```
   SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT=1
   SVT_PCIE_PIPE_SPEC_VER_4_3=1
   ```

b. DUT connection: All connection for 4.2 PCLK as PHY input, it will require all PIPE 4.3 connections to be added.

   The following codes shows a X16 connection.

Example 6-4 X16 connection

```
`ifdef PCIESVC_PIPE_SPEC_VER_4_3
   wire [3:0] endpoint0_powerdown;
   wire endpoint0_async_power_change_ack;
`else
   wire [2:0] endpoint0_powerdown;
`endif
//VIP connection
pcie_mpipe_x16_interconnect pcie_mpipe_x16_interconnect (
       .reset (test_top.reset),
       ......
       ......
       .powerdown (endpoint0_powerdown),
`ifdef PCIESVC_PIPE_SPEC_VER_4_3
       .async_power_change_ack (endpoint0_async_power_change_ack),
`endif
// DUT connection:
`ifdef PCIESVC_PIPE_SPEC_VER_4_3
   assign dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_powerdown =
<MAC output4-bit powerdown dignal>;
   assign
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_async_power_change_ack  =
<MAC output AsyncPowerChangeAck >
`else
   assign dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_powerdown =
<MAC output 3-bit powerdown dignal>;
`endif
```

6.27 Configuration Space Tests

In the current release, Synopsys provides the following configuration space test cases to verify different attributes of PCIe configuration space inside DUT.

1. *ts.tl_cfg_capability_check.sv*: This test case reads out the whole configuration space from the DUT, and verifies whether all PCIe capabilities implemented by the DUT are present once inside a function.
2. *ts.tl_cfg_valid_range_test.sv*: This test reads out the whole configuration space from the DUT and verifies that no register and field advertise any reserved value as per the PCIe specification.

Note These tests must be run just after activating the PL-DL linkup.

6.28 Application Error Reporting Interface Test (Only for SNPS IIP)

Test: tl_application_error_reporting

- You must set the `APP_RETURN_ERR_EN` configuration parameter to enable the Application Error Reporting Interface in IIP TCL (configuration file).
- From Application BFM, you must invoke the `application_error_reporting_handler` task when the TLP is received at its application interface.

6.28.1 Configuring the *env* to use the Application Error Reporting

- Changes in DUT specific interface file (*rtl_dut_specific_if.svi*).

  You must declare the following DUT-specific interface signals:

```
// Application Error Reporting.
logic app_hdr_valid;
logic [127:0] app_hdr_log;
logic [11:0] app_err_bus;
logic [7:0] app_err_func_num;
logic app_err_advisory;
logic app_err_vfunc_active;
logic [7:0] app_err_vfunc_num;
```

- Changes in DUT Wrapper file (*pcie_dut_sv_wrapper.sv*)
  - DUT-specific signals:

```
bit            dut_app_hdr_valid  ;
bit [127:0]    dut_app_hdr_log  ;
bit [11:0]     dut_app_err_bus  ;
bit [NF-1:0]   dut_app_err_func_num  ;
bit            dut_app_err_advisory  ;
bit            dut_app_err_vfunc_active;
bit [NVF-1:0]  dut_app_err_vfunc_num  ;
```

  - Connect all Application-specific error signals to DUT specific interface:

```
assign dut_app_hdr_valid             = dut_specific_if.app_hdr_valid;
assign dut_app_hdr_log               = dut_specific_if.app_hdr_log;
assign dut_app_err_bus               = dut_specific_if.app_err_bus;
assign dut_app_err_func_num          = dut_specific_if.app_err_func_num;
assign dut_app_err_advisory          = dut_specific_if.app_err_advisory;
assign dut_app_err_vfunc_active      = dut_specific_if.app_err_vfunc_active;
assign dut_app_err_vfunc_num         = dut_specific_if.app_err_vfunc_num;
```

  - Connection with DUT:

```
`ifdef APP_RETURN_ERR_EN
    .app_hdr_valid                (dut_app_hdr_valid),
    .app_hdr_log                  (dut_app_hdr_log),
`ifdef CX_TLP_PREFIX_ENABLE
    .app_tlp_prfx_log             (dut_app_tlp_prfx_log),
`endif  // CX_TLP_PREFIX_ENABLE(dut_
    .app_err_bus                  (dut_app_err_bus),
    .app_err_advisory             (dut_app_err_advisory),
    .app_err_func_num             (dut_app_err_func_num),
`ifdef CX_SRIOV_ENABLE
    .app_err_vfunc_num            (dut_app_err_vfunc_num),
    .app_err_vfunc_active         (dut_app_err_vfunc_active),
`endif // CX_SRIOV_ENABLE
`endif // APP_RETURN_ERR_EN
```

- Changes in Application BFM file (*pcie_ep_dut_external_app_bfm.sv*)
  - You must invoke the `application_error_reporting_handler` method to drive the Application Error Reporting interface signals.
  - The `application_error_reporting_handler` method must be invoked after an inbound TLP is completely received at the application interface.
  - In SNPS *env*, the `process_tlp_requests` subroutine is invoked during the task method.
  - The `process_tlp_requests` subroutine is available only with AXI Interface. In case of native interface, you must call the `application_error_reporting_handler` subroutine (similar to `process_tlp_request`), because `process_tlp_request` will be bypassed due to the `SVT_PCIE_TEST_SUITE_EXCLUDE_APP_INTERFACE` subroutine.

6.29 Limit Transfer Size Per-BAR (Optional, DUT Specific Feature)

If DUT has a limitation of maximum transfer size per-BAR basis, then you can control this by the following settings:

- You must enable configuration variable and then set the transfer size per-BAR basis from the base test.
- If configuration is enabled, then all test sequences follow the constraint and transfer size (length) will be selected as per address range lies within BAR limit. Example/use model shown below.

  List of test suite configuration variable that must be set in base test.

  - During creat_cfg() subroutine, you must set this test suite configuration variable `enable_transfer_size_limit_per_bar` (by default, it is set to zero).
  - Once the above mentioned configuration variable is set, then `limit_transfer_size_per_bar_basis`() subroutine is called where test suite configuration multi-dimensional fixed arrays (bit [9:0] `min_transfer_range_per_bar`[6], `max_transfer_range_per_bar`[6]) are configured to limit the transfer size (in terms of DW) per-BAR basis.

    Control variables declared in test suite configuration class are as follows:

```
/**
 * Configuration to enable transfer size limit per BAR basis.
 * User can control by setting variable from base test.
 */
bit enable_transfer_size_limit_per_bar;
```

```
/**
 * The variables specify minimum and maximum transfer size in DW per bar basis.
 * Any TLP payload cannot exceed this value in size if its configured.
 * E.g let's say BAR_1 support upto max transfer size 16 DW (64B) then
 * min_transfer_range_per_bar [1] = 1  and
 * max_transfer_range_per_bar [1] = 16.
 */
bit [9:0] min_transfer_range_per_bar[6];
bit [9:0] max_transfer_range_per_bar[6];
```

Code snippet of subroutine `limit_transfer_size_per_bar_basis`() from base test is as follows (you can limit the transfer size as per there requirement).

```
virtual function void limit_transfer_size_per_bar_basis();

  begin
    // BAR0 Max transfer size limit to 3DW.
    cfg.min_transfer_range_per_bar[0] = 1;
    cfg.max_transfer_range_per_bar[0] = 3;

    // BAR1  Max transfer size limit to 7DW.
    cfg.min_transfer_range_per_bar[1] = 1;
    cfg.max_transfer_range_per_bar[1] = 7;

    // BAR2  Max transfer size limit to 11DW.
    cfg.min_transfer_range_per_bar[2] = 1;
    cfg.max_transfer_range_per_bar[2] = 11;

    // BAR3  Max transfer size limit to 15DW
    cfg.min_transfer_range_per_bar[3] = 1;
    cfg.max_transfer_range_per_bar[3] = 15;

    // BAR4 Max transfer Size limit to 19DW
    cfg.min_transfer_range_per_bar[4] = 1;
    cfg.max_transfer_range_per_bar[4] = 19;

    // BAR5   Max transfer size limit to 24DW
    cfg.min_transfer_range_per_bar[5] = 1;
    cfg.max_transfer_range_per_bar[5] = 24;
  end
endfunction // limit_transfer_size_per_bar_basis
```

If you want to transmit memory transaction with more than the transfer size limit, then you need to disable the constraint, and then the TLP will be split into multiple TLPs as per configured transfer size limit. (for example, if a test was supposed to do a write/read for a certain transfer size (example, 64B), but transfer size limited to a smaller value (example, that is 32B), then primitive sequence will break up those accesses into two sets of transactions each of 32B to cover the same range.) As per requirement `transfer_size_per_bar` constraint must be disabled/off from sequence (*.transfer_size_per_bar.constraint_mode(0)). Constraint present in the following primitive sequences and in address generator class.

- `svt_pcie_ts_driver_app_transaction_mem_wr_sequence`
- `svt_pcie_ts_address_generator`

There are some length-specific error scenario or `Max_Payload_Size` validation cases for which this configuration setting is not applicable (for any error scenario related to length, if you have enabled it, then you must disable this feature by setting test suite status variable `enable_transfer_size_limit_per_bar` to zero or else you will receive constraint failure).

```
i. tl_error_msg_tc
ii.tl_split_cpld_max_payload_size
iii.tl_rx_max_payload_size_error
iv.tl_ari_rx_max_payload_size_error
v. gen3_pl_skip_in_tlp_payload_after_scrambling
```

There are some exception primitive sequences which are used for the error injection, for which this feature is disabled. Wherever these primitive sequences are used this feature in not applicable.

```
svt_pcie_ts_driver_app_transaction_exception_sequence
svt_pcie_ts_driver_app_mem_request_exception_sequence
```

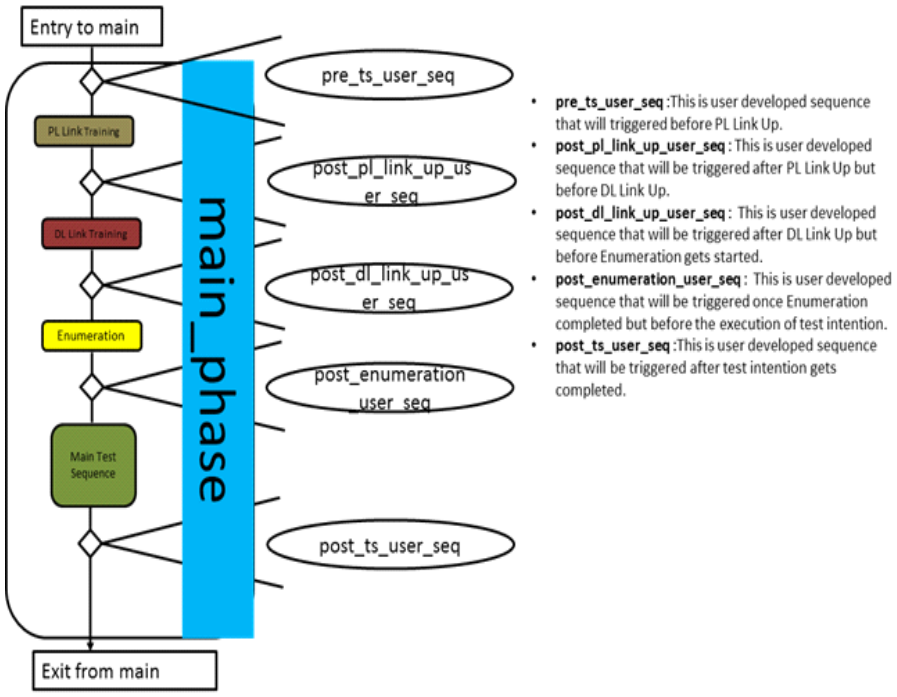
List of test cases which are using above mentioned primitive sequences:

```
i. dl_aer_tlp_lcrc_error
ii.gen3_pl_frame_parity_verify_error
iii.gen3_pl_length_for_fcrc_error
iv.tl_aer_tlp_ecrc_error
v. tl_aer_first_error_pointer_update_for_multiple_error
vi.dl_tlp_seq_number_error
vii.dl_rx_trans_nullified_tlp
viii.gen2_dl_invalid_tlp_with_decode_error
ix.dl_duplicate_tlp_seq_number_error
x. gen2_dl_tlp_missing_end_framing_error
xi.gen3_pl_multi_sdp_in_symbol_time_error
xii.gen3_pl_frame_parity_verify_error
xiii.gen3_pl_length_for_fcrc_error
xiv.pl_pmg_d1_tlp_other_than_cfg_response_ur
xv.pl_pmg_pci_pm_d3_l1_pm_request_ack_dllp_corrupt_error
xvi.tl_td_set_corrupt_ecrc_error
xvii.tl_tx_cpl_rsp_ur_error
xviii.tl_unlock_msg
xix.tl_atomicop_operand_size_support_mismatch_error
```

6.30 Customizing Verification Code in Test Suite Sequences

This section describes the steps to customize verification code with test suite sequences. Figure 6-2 describes the architectural level diagram.

Figure 6-2 Architecture



Note Test *tl_sample_test_to_invoke_user_sequence* is provided for reference.

6.30.1 Configuration Variables for User Sequences

Table 6-7 shows the list of configuration variables required to configure as per your requirements.

Table 6-7 User Sequence Controllable Test Suite Configuration Variables

Test Suite Configuration	Description
enable_user_sequence	Configuration variable that should be enabled through base test, if you want to run user sequences on test sequences.
enable_pre_ts_user_seq	Enable user sequence to run on test sequences before PL Link Up.
enable_post_pl_link_up_user_seq	Enable user sequence to run on test sequences after PL Link Up but before DL Link Up.
enable_post_dl_link_up_user_seq	Enable user sequence to run on test sequences after DL Link Up but before Enumeration gets started.
enable_post_enumeration_user_seq	Enable user sequence to run on test sequences after Enumeration gets completed, but before execution of test intention.

Table 6-7 User Sequence Controllable Test Suite Configuration Variables

Test Suite Configuration	Description
enable_post_ts_user_seq	Enable user sequence to run on test sequences after test intention gets completed.

For more details about the configuration variables, see HTML class reference documentation:
$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/doc/pcie_test_suite_svt_uvm_ts_reference/pcie_test_suite_svt_dut_ep_uvm_ts_reference/html/class_2570.html

6.30.2 Configuring the cust_base_test to Hook User-Defined Sequences

Perform the following steps to configure the `cust_base_test` to hook user-defined sequences:

1. You must create your sequences with your verification code and use it as shown below:

   File: *pcie_ep_dut_cust_base_test.sv*

```
 /**
* This is user developed sequence that will triggered before PL Link Up.
* To Do : User  need to replace sequence
* svt_pcie_device_system_virtual_user_controllable_sequence with their
* developed user sequence, if they want to hook this sequence with TS sequences.
*/
 svt_pcie_device_system_virtual_user_controllable_sequence  pre_ts_user_seq;

 /**
* This is user developed sequence that will be triggered after PL Link Up but before DL Link Up.
* To Do : User  need to replace sequence svt_pcie_device_system_virtual_user_controllable_sequence
* with their developed user sequence, if they want to hook this sequence with TS sequences.
*/
 svt_pcie_device_system_virtual_user_controllable_sequence post_pl_link_up_user_seq;

 /**
* This is user developed sequence that will be triggered after DL Link Up but before Enumeration gets started.
* To Do : User  need to replace sequence svt_pcie_device_system_virtual_user_controllable_sequence
*  with their developed user sequence, if they want to hook this sequence with TS sequences.
*/
 svt_pcie_device_system_virtual_user_controllable_sequence  post_dl_link_up_user_seq;

 /**
* This is user developed sequence that will be triggered once Enumeration completed but before the execution of
* test intention.
* To Do : User  need to replace sequence svt_pcie_device_system_virtual_user_controllable_sequence
* with their developed user sequence, if they want to hook this sequence with TS sequences.
*/
 svt_pcie_device_system_virtual_user_controllable_sequence post_enumeration_user_seq;

/**
* This is user developed sequence that will be triggered after test intention gets completed.
* To Do : User  need to replace sequence svt_pcie_device_system_virtual_user_controllable_sequence
*  with their developed user sequence, if they want to hook this sequence with TS sequences.
*/
 svt_pcie_device_system_virtual_user_controllable_sequence  post_ts_user_seq;

/**
```

```
 * This sequence is TS virtual base sequence and it will cast with pre_ts_user_seq
 * and then shared it with TS virtual base sequence.
 */
  svt_pcie_ts_device_system_virtual_base_sequence base_pre_ts_user_seq;

  /**
 * This sequence is TS virtual base sequence and it will cast with post_pl_link_up_user_seq
 * and then shared it with TS virtual base sequence.
 */
  svt_pcie_ts_device_system_virtual_base_sequence base_post_pl_link_up_user_seq;

 /**
 * This sequence is TS virtual base sequence and it will cast with post_dl_link_up_user_seq
 * and then shared it with TS virtual base sequence.
 */
  svt_pcie_ts_device_system_virtual_base_sequence base_post_dl_link_up_user_seq;

  /**
 * This sequence is TS virtual base sequence and it will cast with post_enumeration_user_seq
 * and then shared it with TS virtual base sequence.
 */
  svt_pcie_ts_device_system_virtual_base_sequence base_post_enumeration_user_seq;

  /**
 * This sequence is TS virtual base sequence and it will cast with post_ts_user_seq
 * and then shared it with TS virtual base sequence.
 */
  svt_pcie_ts_device_system_virtual_base_sequence base_post_ts_user_seq;
```

2. Enable test suite configuration variable as per your requirement.

   For example, if you want to run user sequence once DL link is up. You must set the test suite configuration variable `enable_user_sequence` and `enable_post_dl_link_up_user_seq` to 0.

   File: *pcie_ep_dut_cust_base_test.sv*

```
function void build_phase(uvm_phase phase);

    super.build_phase(phase);
    . . . . .
 . . . . .
    . . . . .

// If user wants to hook their sequences with TS sequences then test suite
  // configuration variable enable_user_sequence need to be set.
if (cfg.enable_user_sequence) begin
      //=============================================================================
   // Requirement specific Configuration settings if prime configuration variable enable_user_sequence is enabled.
   //=============================================================================
      cfg.enable_pre_ts_user_seq = 1'b1;
      cfg.enable_post_pl_link_up_user_seq = 1'b1;
      cfg.enable_post_dl_link_up_user_seq = 1'b1;
      cfg.enable_post_enumeration_user_seq = 1'b1;
      cfg.enable_post_ts_user_seq = 1'b1;
      set_user_base_sequence();
end
. . . . .
```

```
      . . . . .
      . . . . .
endfunction
```

3. The `set_user_base_sequence` method will be invoked during `build_phase` when configuration variable `enable_user_seqeunce` is set. It is used is to create user-specific sequences, then `cast` it to virtual base sequence and then share the information with test suite virtual base sequence.

❖ Usage

- ✦ If you want to run customize verification code on a single test case. For example, test *tl_sample_test_to_invoke_user_sequence* (it is a demo case created to validate user sequences hook up).
- ✦ Attach the user sequences `pre_ts_user_seq, post_pl_link_up_user_seq, post_dl_link_up_user_seq, post_enumeration_user_seq, post_ts_user_seq` using `uvm_config_db` set method from `cust_base_test` during method `set_user_base_sequence` (for details, see the following example).
- ✦ If you want to run customize verification code on all test suite cases (Limitation describes the cases on which this hook will not work.), then during `uvm_config_db` call in place of `<inst_name>` need to use method to set every test sequences.
- ✦ All user sequences is an extension of `svt_pcie_ts_device_system_virtual_base_sequence`.

File: *pcie_ep_dut_cust_base_test.sv*

```
/**
  * This method will be invoked if this 'enable_user_sequence' test suite configuration variable is set.
  * It will first create user specific sequences then cast it to virtual base seqeunce and finally
  * shared information with TS virtual base sequence.
  */
function void set_user_base_sequence();
begin
       // Shared configuration setting information with TS virtual base sequence.
       uvm_config_db#(bit)::set(this,{cfg.path_to_pcie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"enable_user_sequence",cfg.enable_user_sequence);
       uvm_config_db#(bit)::set(this,{cfg.path_to_pcie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"enable_pre_ts_user_seq",cfg.enable_pre_ts_user_seq);
       uvm_config_db#(bit)::set(this,{cfg.path_to_pcie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"enable_post_pl_link_up_user_seq",cfg.enable_post_pl_link_up_user_seq);
       uvm_config_db#(bit)::set(this,{cfg.path_to_pcie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"enable_post_dl_link_up_user_seq",cfg.enable_post_dl_link_up_user_seq);
       uvm_config_db#(bit)::set(this,{cfg.path_to_pcie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"enable_post_enumeration_user_seq",cfg.enable_post_enumeration_user_seq);
       uvm_config_db#(bit)::set(this,{cfg.path_to_pcie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"enable_post_ts_user_seq",cfg.enable_post_ts_user_seq);


       // Shared test default sequence name with TS virtual base sequence.
```

```
      uvm_config_db#(string)::set(this,{cfg.path_to_pcie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
"},"default_test_seq_name","svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_u
ser_sequence");

   // create method for user sequences.
      pre_ts_user_seq =
svt_pcie_device_system_virtual_user_controllable_sequence::type_id::create("pre_ts_user_
seq");
      post_pl_link_up_user_seq =
svt_pcie_device_system_virtual_user_controllable_sequence::type_id::create("post_pl_link
_up_user_seq");
      post_dl_link_up_user_seq =
svt_pcie_device_system_virtual_user_controllable_sequence::type_id::create("post_dl_link
_up_user_seq");
      post_enumeration_user_seq =
svt_pcie_device_system_virtual_user_controllable_sequence::type_id::create("post_enumera
tion_user_seq");
      post_ts_user_seq =
svt_pcie_device_system_virtual_user_controllable_sequence::type_id::create("post_ts_user
_seq");

   // Calling $cast method.
      void'($cast(base_pre_ts_user_seq,pre_ts_user_seq));
      void'($cast(base_post_pl_link_up_user_seq,post_pl_link_up_user_seq));
      void'($cast(base_post_dl_link_up_user_seq,post_dl_link_up_user_seq));
      void'($cast(base_post_enumeration_user_seq,post_enumeration_user_seq));
      void'($cast(base_post_ts_user_seq,post_ts_user_seq));

   // Shared information with TS virtual base sequence.

uvm_config_db#(svt_pcie_ts_device_system_virtual_base_sequence)::set(this,{cfg.path_to_p
cie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"pre_ts_user_seq",base_pre_ts_user_seq);

uvm_config_db#(svt_pcie_ts_device_system_virtual_base_sequence)::set(this,{cfg.path_to_p
cie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"post_pl_link_up_user_seq",base_post_pl_link_up_user_seq);

uvm_config_db#(svt_pcie_ts_device_system_virtual_base_sequence)::set(this,{cfg.path_to_p
cie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"post_dl_link_up_user_seq",base_post_dl_link_up_user_seq);

uvm_config_db#(svt_pcie_ts_device_system_virtual_base_sequence)::set(this,{cfg.path_to_p
cie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"post_enumeration_user_seq",base_post_enumeration_user_seq);

uvm_config_db#(svt_pcie_ts_device_system_virtual_base_sequence)::set(this,{cfg.path_to_p
cie_test_suite_env,
".test_env_seqr.svt_pcie_ts_device_system_virtual_tl_sample_test_to_invoke_user_sequence
*"},"post_ts_user_seq",base_post_ts_user_seq);
    end
  endfunction:set_user_base_sequence
```

6.30.3 Limitation

Hook up will not work for test suite cases that are neither using PL link up sequence (`svt_pcie_device_system_virtual_enable_link_up_sequence`) nor activate link sequence (`svt_pcie_device_system_virtual_activate_link_sequence`).

6.31 Precision Time Measurement (PTM)

The PCIe test suite has been enhanced with new test cases to support PTM protocol verification. The newly added test cases are applicable only when PTM feature is supported by controller DUT (EP/RC). The added functionality tracks the timestamps for sending and receiving PTM messages which are used for sending PTM `ResponseD` (for RC VIP) and for calculating PTM context (for EP VIP) using the DL and PL callbacks.

Note The naming convention followed for the test cases is *ts.tl_ptm**.

- Extend a sequence class with `svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence` as the base class and override the base class with derived class. This is required if you want to override/implement some tasks (which are DUT implementation specific) of the base class. Refer to the following steps to determine whether overriding is required. For more details, see comments provided in this base class.
- Set the following DUT capability attributes before running any test case:

1. `enable_ptm_context_read`: This bit indicates whether a calculated PTM context value by EP DUT can be accessed by test cases. Default value is 1. Set this attribute to 0 in case PTM context cannot be accessed.
2. `enable_ptm_context_valid_bit_read`: This bit indicates if the test can access the bit which specifies if PTM context of EP DUT is currently valid or not. Default value is 1. Set this attribute to 0 in case PTM context valid/invalid information bit cannot be accessed.
3. `enable_ptm_context_update_trigger_through_pcie_cfg_wr`: This bit indicates if software or test can trigger EP DUT (PTM Requester) to request for PTM Master time (initiate PTM requests) through PCIe `CFG_WR` mechanism. Enable this if EP DUT has some Vendor Specific Capability in its PCIe configuration space through which it can be triggered to update its PTM context. Default value is 1.

Note If you set this bit to 0 (which implies that DUT will be triggered through some non-PCIe DUT specific interface), then you must override/implement task `trigger_ptm_context_refresh` in derived class extended from `svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence`.

4. `enable_ptm_context_read_through_pcie_cfg_rd`: This bit indicates if software or test can read the PTM context value of EP DUT (that is the calculated PTM Master time at t1') through PCIe `CFG_RD` mechanism. Enable this if EP DUT has some Vendor Specific Capability in its PCIe configuration space through which test can access the PTM context. Default value is 1.

Note If you set this bit to 0 (which implies PTM context will be read through some non-PCIe DUT specific interface), then you must override/implement task `ptm_context_read` in derived class extended from `svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence`.

5. `enable_ptm_context_valid_bit_read_through_pcie_cfg_rd`: This bit indicates if software or test can read the PTM context valid/invalid bit through PCIE `CFG_RD` mechanism. Enable this if EP DUT has some Vendor Specific Capability in its PCIe Configuration space through which test can access the information that PTM context of EP DUT is currently valid or not. Default value is 1.

Note If you set this bit to 0 (which implies PTM context valid bit will be read through some non-PCIe DUT specific interface), then you must override/implement task `ptm_context_valid_bit_read` in derived class extended from `svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence`.

6. `enable_ptm_current_t1_t4_timestamp_read`: This bit indicates if software or test can read the current t1/t4 timestamps noted by PTM Requester (EP DUT). Default value is 1.
7. `enable_ptm_current_t1_t4_timestamp_read_through_pcie_cfg_rd`: Software or test can read the current t1/t4 timestamps noted by PTM Requester (EP DUT) through PCIe configuration space access. Enable this if EP DUT has some Vendor Specific Capability in its PCIe configuration space through which test can access the t1/t4 timestamps. Default value is 1.

Note If you set this bit to 0 (which implies that t1/t4 timestamp will be read from non-PCIe DUT specific interface), then you must override/implement task `ptm_current_t1_timestamp_read` and `ptm_current_t4_timestamp_read` in derived sequence class extended from `svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence`.

If `enable_ptm_context_read=0`, then the following test cases cannot be run.

- `tl_ptm_dialogs-ep`
- `tl_ptm_update_t1_on_request_replay-ep`
- `tl_ptm_update_t4_on_duplicate_response-ep`

If `enable_ptm_context_valid_bit_read=0 && enable_ptm_context_read==0`, then the following test case cannot be run.

- `tl_ptm_context_invalidation-ep`

❖ If DUT has some Vendor Specific Enhanced Capability (VSEC) for PTM, then for VSEC, extend the `svt_pcie_test_suite_configuration_class` and re-implement function `get_vendor_specific_reg_off_or_field_indices`. This function is used to get the values of all register offsets or field indices for PTM VSEC. Using this function, you can specify different register offsets/field indices values for

- Different links
- DUT modes
- Functions within the DUT (PF/VF number)

Set the values for following attributes in the function:

- `vsec_ptm_base_addr`
- `vsec_ptm_context_read_lower_32_reg_off`
- `vsec_ptm_context_read_upper_32_reg_off`
- `vsec_ptm_req_issue_control_reg_off`

- vsec_ptm_req_issue_control_bit_field
- vsec_ptm_context_valid_bit_reg_off
- vsec_ptm_context_valid_bit_field
- vsec_ptm_t1_timestamp_read_lower_32_reg_off
- vsec_ptm_t1_timestamp_read_upper_32_reg_off
- vsec_ptm_t4_timestamp_read_lower_32_reg_off
- vsec_ptm_t4_timestamp_read_upper_32_reg_off

Note It is recommended to avoid using the replaceable defines for PTM VSEC which were used in previous releases and you must use this function only. Currently, these defines are retained for backward compatibility but may be removed from the model in the upcoming releases.

Note The *svt_pcie_test_suite_configuration.sv* file includes the information about each attribute.

- Additional optional controls have been added to provide users with better test controllability:

Table 6-8 `svt_pcie_test_suite_sequence_configuration` **Attributes**

Attributes	Description
`link_delay_in_ns_for_ptm`	Specifies the link delay. This attribute is used in PTM test cases. Default value is 30ns. You must extend the `svt_pcie_test_suite_configuration` class and re-implement function `setup_ptm_link_delays` to set the value of this attribute according to your environment.
`stop_ptm_dialogs_on_tlp_replay_or_duplicate`	This is used for the EP DUT. If set to 1, it indicates that upon receiving duplicate PTM TLP or transmitting a replayed PTM TLP: • PTM context of EP DUT invalidated. • EP DUT stops the current PTM dialogs and does not transmit any further PTM request for the ongoing set of dialogs. • To make the PTM context of DUT valid again, it requires another two set of successful PTM dialogs. If set to 0 it indicates that upon receiving duplicate PTM TLP or transmitting a replayed PTM TLP: • EP DUT continues the current set of PTM dialogs. Current PTM dialog remains valid in calculating the PTM context and the DUT will continue to send further PTM requests. Its default value is 0.
`min_ptm_accuracy_range_ns`	Specifies the minimum accuracy range for the calculated PTM Context value. Default minimum accuracy is +-150ns.
`offset_for_ptm_master_time_ns`	Specifies the offset for PTM Master time in ns. Time domain of RC(VIP) is (`($realtime/1ns) + seq_cfg.offset_for_ptm_master_time_ns`). Default value is 50ns.
`enable_ptm_context_invalidation_through_hot_reset`	Specifies if PTM context of EP DUT is invalidated after hot reset. Default value is 1.

Table 6-8 `svt_pcie_test_suite_sequence_configuration` **Attributes**

Attributes	Description
`enable_ptm_context_invalidation_through_link_disable`	Specifies if PTM context of EP DUT is invalidated after disabling and re-enabling the link. Default value is 1.
`enable_ptm_context_invalidation_through_speed_change`	Specifies if PTM context of EP DUT is invalidated after speed change. Default value is 1.
`min_time_between_ptm_req_without_rsp_in_ns`	Specifies the minimum time PTM Requester should wait before issuing another PTM Request without receiving any PTM Response for the precious request. Default value is 100000ns (100us) as per the specification.
`min_time_between_ptm_req_with_rsp_in_ns`	Specifies the minimum time PTM Requester should wait before issuing another PTM Request having received PTM Response for the previous request. Default value is 1000ns (1us) as per the specification.
`insert_ptm_retry_hold_ack_or_send_nak`	This controls the mechanism for replay from VIP or DUT for following tests (`tl_ptm_update_*`). There are two mechanisms for replay: • Holding the ACK and thus replay happens after `replay_timeout`. • Sending NAK. Default value is -1 which randomizes between any of the two mechanisms.
`num_iterations_for_background_traffic`	You can control the number of iterations for background traffic. This is used in API `generate_background_random_tlp_traffic()` which is used in tests `tl_ptm_dialogs-ep`. Default value is 200.
`time_to_wait_between_background_tlps_in_ns`	You can also control how busy link should be and how frequently background TLP packets should be coming on the link. This is used in API `generate_background_random_tlp_traffic()` which is used in `tl_ptm_dialogs-ep`. Default value is 10ns.
min_propagation_delay_for_ptm_master_ns	Indicates the minimum random delay after which the VIP transmits the PTM response when the RC(VIP) receives a PTM request. This attribute is valid only when the VIP is acting as RC. Default value is 100ns.
max_propagation_delay_for_ptm_master_ns	Indicates the maximum random delay after which the VIP transmits the PTM response when the RC(VIP) receives a PTM request. This attribute is valid only when the VIP is acting as RC. Default value is 800ns.

Note

- Currently, the PTM test cases cannot be run with Config bypass feature.
- For running PTM test cases, time precision of user testbench should be smaller than or equal to 1ns value (for example, 100ps, 1ps, and so on.). This is required because all the timing information in PTM protocol is shared in ns and PTM context calculated by EP DUT is expected to be within `min_ptm_accuracy_range_ns` value (which is ~150ns). If time precision is greater than 1ns, then set the `sequence_configuration` attribute `min_ptm_accuracy_range_ns` with a greater value.
- For running PTM test cases, you must program your DUT's TX/RX latencies (if required) correctly for each speed/linkwidth. These latencies are time gap between a packet transmission from device's core to PCIe device's physical boundaries. These latencies are used in determining exact TX/RX timestamps for the DUT. If such latencies exists for the DUT, then you must take care to program these in your environment.

6.32 Gen5 Support

The key Gen5 features supported by the PCIe test suite are as follows:

Feature	Description
SANITY	Basic Gen5 testing
EQ	32GT Equalization
EQ_HIGHEST_RATE	Equalization with highest rate selection feature.
EQ_VIA_LOOPBACK	Equalization transition from Loopback LTSSM state.
FRAMING	Gen5 Framing error tests
LOOPBACK	Gen5 Loopback tests
MODIFIED_TS	Gen5 modified test suite exchange tests
DATA_PARITY_ERR	Gen5 Data Parity error tests
PRECODING	Gen5 Precoding tests
RECOVERY	Gen5 Recovery LTSSM transition tests
REDO_EQ	Gen5 Redo Equalization
REGISTER_CHK	Register Checking test
RX_MARGIN	Rx Margining test
RETIMER	Re timer tests

6.32.1 Mode of Qualification

- Low Pin Count (LPC)
- Serial
- PIPE 4.4.1

6.32.2 Supported Tests

All the test suite tests supported with Gen5 capable device.

- Gen5/Gen4/Gen3/Gen2/Gen1/common tests support with EP DUT
- Retimer support

Complete list of all supported tests are available at the following location:

$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/examples/sverilog/tb_dut_pcie/tests

6.32.3 Gen5 Sequence APIs

To facilitate a new sequence development, APIs have been added into the PCIe test suite product.

The APIs and their documentation is available at:

$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/examples/sverilog/tb_dut_pcie/seq_and_cb/svt_pcie_ts_device_system_virtual_gen5_sequence_collection.sv

While implemented as part of `svt_pcie_ts_device_system_virtual_gen5_sequence_collection`, all the APIs are not limited for Gen5 operations, some of these are generic and can be used for non-Gen5 specific stimulus.

Example 6-5 API Code Sample

```
/* Method to execute speed change from VIP. Should be called in L0.
   * Method will return the control back when as soon as Recovery.Lock is entered from
L0.
   * @param target_rate Specifies the desired rate to speed to.
   */
  extern virtual task execute_speed_change(svt_pcie_pl_status::link_speed_enum
target_rate);

  /* Method to initiate Loopback.Entry from Config.LW.Start
   * @param eq_via_loopback Indicates whether Equalization state will be entered via
Loopback.Entry
   * @param lut Indicates the 'Lane Under Test'. Only applicable if eq_via_loopback is 1
   * @param modified_compliance_in_loopback Indicates whether loopback master will
request slave to send modified compliance pattern. Only applicable if eq_via_loopback is
1.
   * @param device_type indicates which device will be loopback Master(VIP or VIP
instance acting as DUT). VIP instance acting as DUT can be made Loopback Master only in
back-to-back sims and for PHY DUT.
   */
  extern virtual task initiate_loopback_via_config(bit eq_via_loopback=0, int lut=0, bit
modified_compliance_in_loopback=0, device_type_enum device_type=VIP);

  /* Method to initiate Loopback.Entry from Recovery Idle state
   * @param device_type indicates which device will be loopback Master(VIP or VIP
instance acting as DUT). VIP instance acting as DUT can be made Loopback Master only in
back-to-back sims and for PHY DUT.
   */
  extern virtual task initiate_loopback_via_recovery(device_type_enum device_type=VIP);

  /* Method to initiate Loopback exit.
   * @param device_type indicates Loopback Master for which loopback exit to be
initiated. legal values are:
   *        - VIP (Initiate Loopback Exit for VIP acting as loopback Master)
```

```
   *        - DUT (Initiate Loopback Exit for VIP(DUT) acting as loopback Master).
Applicable in b2b sims and for PHY DUT.
   */
  extern virtual task initiate_loopback_exit(device_type_enum device_type=VIP);

  /** Method to enable Equalization setup with Gen5 Eq_via_Loopback cases. If this API
is called, then equalization sequences run in parallel to eq_via_loopback test sequences
which make preset/coefficient requests in Phase2(from RC VIP) and Phase3(from EP VIP).*/
  extern virtual task enable_eq_setup_for_eq_via_loopback();

  /* Method to check the DUT supports for Modified TS */
  extern virtual task check_dut_support_modified_ts();

  /* Method to request Precoding from VIP via 128b/130b EQ TS2 at 16GT/s. Should be
called in L0 or at time0.*/
  extern virtual task request_precoding_at_16gts();

  /* Method to configure VIP to advertise 'No EQ Required'.*/
  extern virtual task configure_vip_for_no_eq_required();
```

To add a new Gen5 test using APIs, create a new sequence extending from `svt_pcie_ts_gen5_base_sequence` and call the appropriate APIs from the body method of the sequence. Register this sequence as the `default_sequence` in the test.

Example 6-6 Sequence Implemented Using APIs

```
task
svt_pcie_ts_device_system_virtual_gen5_pl_clwa_no_8ts2_w_elbc_11b_in_polling_config_sequ
ence::body();
  string method_name = "body";
  registered_seq =
"svt_pcie_ts_device_system_virtual_gen5_pl_clwa_no_8ts2_w_elbc_11b_in_polling_config_seq
uence";
  begin
    super.body();
    begin
      /* Wait until DUT LTSSM is in POLLING_CONFIGURATION */

wait_for_state_at_desired_speed(1,svt_pcie_types::POLLING_CONFIGURATION,svt_pcie_pl_stat
us::SPEED_2_5G);

      //-------------------------------------------------------
      // Transmit the error injection in TS2 from VIP such that 8-TS2s ELBC != 11b on
one lane and 8-TS2 with ELBC == 11b  on rest of the lane
      //-------------------------------------------------------
      //Set mask for active lanes to place user defined TS OS on them
      lane_mask = 32'h0;

      randomize_lut();
      lane_mask[lane_under_test] = 1'b1;

       // Set user defined TS2 as '0' set
      set_lane_mask_for_user_ts(1'b0, lane_mask);

      `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: Starting User TS2
stream...")
```

```
      send_user_ts(1'b0,lane_under_test,8,0,,9'h1F7,9'h1F7,,,8'h00); //  8-user TS on
all Lane except one Lane
      `svt_pcie_ts_note(method_name,$sformatf("Starting User TS2 stream on
Lane=%0d,Transmit the error injection in TS2 from VIP such that 8-TS2s ELBC != 11b on
one lane and 8-TS2 with ELBC == 11b  on rest of the lane",lane_under_test));

      check_dut_support_modified_ts();
      if(dut_support_modified_ts_flag==0)
        `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: DUT did not support
Modified TS1/2, Exiting the test...")
      else
      begin
        `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: BEFORE_WAIT :: Waiting
for DUT to CONFIGURATION_LINKWIDTH_START")
        wait (dut_status.pcie_status.pl_status.ltssm_state ==
svt_pcie_types::CONFIGURATION_LINKWIDTH_START);
        `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: AFTER_WAIT :: DUT
entered CONFIGURATION_LINKWIDTH_START, disabling User TS2 stream now")

        // Disable user defined TS1/2 when DUT enters CONFIGURATION_LINKWIDTH_START
        stop_user_ts(1'b0,lane_under_test,0);
```

7
Using the Test Suite: RC DUT

This chapter discusses the following topics:

7.1 Command-Line Options for Demo Mode

7.1.1 RC VIP as DUT in SERDES mode

```
gmake demo_dl_link_up_test-rc
SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_ep_vip_serial.f
SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_vip.f
USE_SIMULATOR=vcsvlog
```

7.1.2 RC VIP as DUT in PIPE mode

```
gmake demo_dl_link_up_test-rc
SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_rc_vip_pipe.f
SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_vip.f
USE_SIMULATOR=vcsvlog
```

7.2 Extended Base Test Use Model

You must extend the Test Suite's base test class to add implementation specific customizations. Following are the steps:

1. Create a new file by taking reference from *pcie_rc_dut_cust_base_test.sv*.

Note It is not recommended to extend this class, use it only as a reference.

2. Add the following two macros in the user defines file (*svt_pcie_test_suite_user_defines.svi*):
   - `SVT_PCIE_TEST_SUITE_RC_DUT_TEST_FILE` with the name of this new file.
   - `SVT_PCIE_TEST_SUITE_RC_DUT_TEST` with the name of this new class.

   In this class, you must set the following configurations:

- Set the configuration properties for link partner VIP based on DUT requirement—timer configuration, link width, and speed.

  For example, when RC is DUT, the cfg configurations are required for EP VIP.

```
cfg.ep_cfg.pcie_cfg.dl_cfg.min_num_tx_initfc1_p = 15
```

- All relevant properties in the `dut_capabilities` configuration object can be programmed as per DUT specific implementation.

  For example, if RC is DUT with ATS feature, support, it can be set as shown below:

  ```
  cfg.rc_cfg.pcie_cfg.dut_capabilities.enable_ats_support
  ```

- Error Demotion
- You must reimplement the `pcie_unified_base_test::set_env_override` method in this class. This method is invoked in `pcie_unified_base_test` at an appropriate point before the `build_phase`.
- You can add your own custom application to application agent in this class.

  This class also serves as placeholder for all configuration settings and 'workarounds' specific to your implementation. You must add the following as per your implementation:

  - If the DUT has limitations on outbound traffic and is not able to transmit TLPs ‘as is’, then a callback similar to `update_app_xact_for_32bit_axi_callback` (take reference from *pcie_env_callback_collection.sv*) must be developed and added to TLP Mapper Component’s callback pool (take reference from *pcie_rc_dut_cust_base_test.sv*). If there is no limitation, then the callback should not be added.
  - If the DUT is not able to calculate a `completer_id` for outbound completions, then it must use a callback similar to `completer_id_calculator_callback`.
  - If the DUT requires completion to be generated from the target application present in Application Agent, then it may want to look at configuration properties available for the target application. For example, ‘< min | max>_read_cpl_data_size_in_bytes’ properties can be set to configure how split completions should be generated.
  - Adjust the Scoreboard switches available in the Test Suite configuration.

Note Even if the Scoreboard is disabled, the configuration settings corresponding to inbound TLPs must be set as per your Application BFM’s limitations (if any).

  - Set other Test Suite configuration properties.
  - All relevant properties in the `dut_capabilities` configuration object must be set here as per your specific implementation.
  - Similarly, set all the configuration properties available in the VIP as required.

7.3 Simulation Timeout and Usage

The test execution is controlled by a global timeout attribute and is terminated when the timer expires. The default value of test timeout is 1200000ns. For a specific test run the default value can be overridden by command line argument SVT_PCIE_TEST_TIMEOUT= *timeout-value-in-ns*.

7.4 MFVC Tests Behavior

The MFVC tests are applicable only if DUT supports MFVC or VC capability. The enumeration sequence discovers MFVC or VC capability support in the DUT, and these tests are executed only if the DUT advertises support for MFVC or VC capability during enumeration.

7.5 Fast Simulation Mode

To facilitate faster simulation of tests, the Test Suite provides a mechanism to set up scaled down values of specification timers and the number of ordered sets communicated during the link initialization phase. This assumes that the DUT also supports the fast simulation mode and the timers can be configured accordingly. Also, it is required to match the RC VIP configuration attributes to be same as of DUT values.

Table 7-1 provides the details of replaceable macro defined in svt_pcie_test_suite_defines.svi (present in the sverilog/include/ directory in the install tree) for fast simulation mode.

Table 7-1 Replaceable macro defined in svt_pcie_test_suite_defines.svi for fast simulation mode

`SVT_PCIE_TEST_SUITE_FAST_SIM_1MS svt_pcie_device_configuration	PCIe VIP RC (When EP as DUT) This macro defines the scaled down value of 1milisec timer for fast simulation mode in ns. (Default value is1024ns)

7.5.1 Configuration of LTSSM State Timers for Fast Simulation Mode

To configure all LTSSM timers, based on the scale factor in the DUT, set the value to the `SVT_PCIE_TEST_SUITE_FAST_SIM_1MS` macro, in ns (default value is 1024). All the timer values are calculated internally (for example, `` `define SVT_PCIE_TEST_SUITE_FAST_SIM_12MS 12 * `SVT_PCIE_TEST_SUITE_FAST_SIM_1MS ``).

Based on the macros defined for different timeouts, the LTSSM timer attributes in the `svt_pcie_test_suite_configuration` class of RC VIP are set up.

7.5.2 Configuration of Erroneous Counts of TS1 or TS2 Ordered Sets

In the PL sequences, erroneous TS1 or TS2 counts are used and can be controlled based on each LTSSM state timeout condition. The attributes are:

- `num_ts_os_error_2ms`
- `num_ts_os_error_12ms`
- `num_ts_os_error_24ms`
- `num_ts_os_error_32ms`
- `num_ts_os_error_48ms`

The above attributes can be updated in the base test (applicable for all PL layer sequences).

Table 7-2 shows the replaceable defines for the erroneous count order sets and their default values.

Table 7-2 Replaceable Defines for the Erroneous Count Order Sets

Defines	Description
`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_2MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 2ms timeout condition. (Default value is 10)
`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_12MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 12ms timeout condition (Default value is 50)

Table 7-2 Replaceable Defines for the Erroneous Count Order Sets

`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_24MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 24ms timeout condition (Default value is 100)
`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_32MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 32ms timeout condition (Default value is 100)
`SVT_PCIE_TEST_SUITE_NUM_TS_OS_ERROR_48MS svt_pcie_device_configuration	This macro defines the number of erroneous TS1 or TS2 count for LTSSM state that have a 48ms timeout condition (Default value is 100)

7.6 ATS

Following are the primitive sequences developed for ATS, as part of the TLP sequence collection. All of them are blocking on successful transmission/receipt of TLP by RC. All ATS tests use these sequences. It is also required to drop all implicitly generated CplDs by the target layer (via callback) of RC VIP as translation completions should be explicitly started from the complex sequences.

- svt_pcie_ts_tlp_ats_mem_request_sequence: For memory requests which need to use the AT field.
- svt_pcie_ts_tlp_translation_completion_sequence: For translation completion required in response to a translation request by EP.
- svt_pcie_ts_tlp_invalidate_request_sequence: For random Invalidation Request request message.
- svt_pcie_ts_tlp_invalidate_completion_sequence: For Invalidation Completion required in response to invalidation request by RC VIP (only used to explicitly transmit from EP VIP in VIP-VIP mode).
- svt_pcie_ts_tlp_page_request_message_sequence: For random page request message.
- svt_pcie_ts_tlp_prg_response_message_sequence: For PRG response message in response to page request by EP DUT.

In case the EP DUT core implements its own internal TAG management, then the actual value used needs to be copied back into the outbound TLP transaction object so that the ATS complex sequences have access to that value when explicitly starting translation completions.

7.7 Multi-Function

The driver_app and tlp base sequences/ extended transaction classes have constraints which in case of a memory/IO transaction from RC randomize it so as to target address range of an existing Function (function/PF/VF). The tlp base sequences/ extended transaction classes additionally constrain requester-id field to that of an existing function. All TL tests utilize this infrastructure to randomly target all functions of the EP device.

7.8 Advanced Error Reporting Testing

The Test Suite includes the necessary tools that assists you in developing new Advanced Error Reporting (AER) tests. This is provided in the base sequence named `svt_pcie_ts_device_system_virtual_tl_main_aer_sequence`. The sequence includes all currently supported AER checks and provides 'callbacks' so that you can use this as a method to inject the actual error

and add any additional logic if required. Your AER tests must extend this class and implement the required callbacks like `inject_error()` task.

Note These 'callbacks' are not related to the VIP callback feature. They are simply virtual methods that you can override to extend the implementation.

AER mask and severity registers are programmed with random values in each iteration and AER checks are performed after the `inject_error()` user callback returns. A complete account of the injected stimulus and checks performed can be found in the HTML class reference of `svt_pcie_ts_device_system_virtual_tl_main_aer_sequence`.

The Test Suite also delivers a number of AER tests whose test sequences extend from this sequence. Therefore, all those tests are capable of performing all the AER checks listed in the HTML class reference and can also be used as a reference to develop your own AER tests.

Following AER-related primitive sequences are also delivered in case you want to use them..

Table 7-3 AER Primitive Sequences

Sequence	Purpose
svt_pcie_ts_device_system_virtual_is_aer_supported_sequence	AER capability availability for particular function.If AER capability present in function then is_aer_present bit will be set to 1 otherwise is aer_present bit will be set to 0.
svt_pcie_ts_device_system_virtual_aer_register_read_sequence	Used to read 32-bit value in AER register with offset provided for particular function.if read done successfully then sequence updates read_done =1 or read_done = 0.
svt_pcie_ts_device_system_virtual_aer_register_write_sequence	Used to write 32-bit value in AER register with offset provided for particular function. if write done successfully then sequence updates write_done =1 or write_done = 0.
svt_pcie_ts_device_system_virtual_aer_register_compare_sequence	Used to compare bit in AER register with offset provided for particular function.Fatal error if provided values mismatches. or success bit set if value matches.

7.9 Readiness Notification

Readiness Notification tests expect DRS/FRS messages from the Endpoint DUT after DRS/FRS events within maximum allowed time. For FRS events, if the expected time values are found in the RN extended capability header then they are used. In all other cases, the values in the 'SVT_PCIE_TEST_SUITE_MAX_DRS_LATENCY_AFTER_DRS_EVENT' and 'SVT_PCIE_TEST_SUITE_MAX_DRS_LATENCY_AFTER_FRS_EVENT' replaceable macro defines are used which the user can set accordingly.

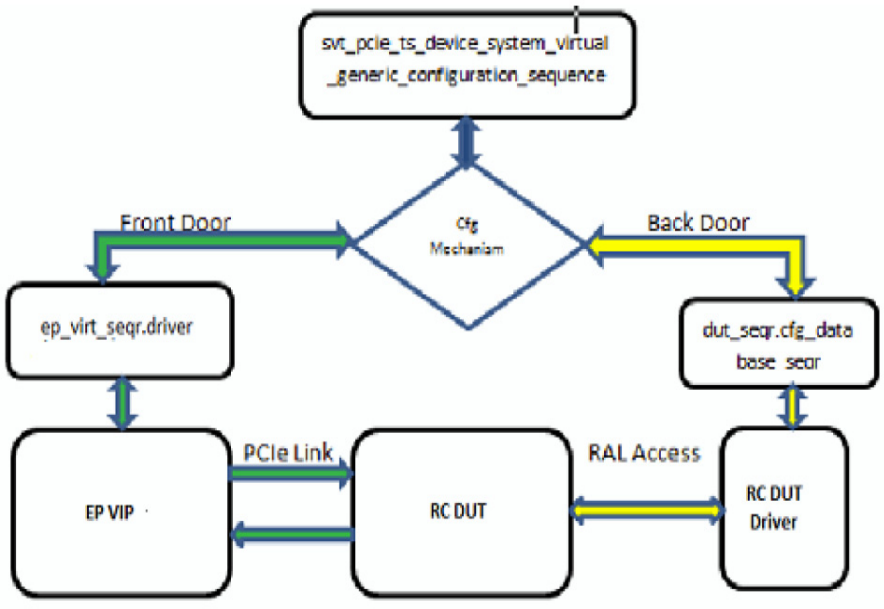
7.10 Backdoor Configuration Access

Backdoor configuration is always valid for RC DUT. This will cause configuration accesses to go on `cfg_database_sequencer` of DUT. Its implementation must be provided by DUT driver. This control is available for tests which have been enhanced to use generic configuration access instead of front door CFG TLP transactions directly by using `svt_pcie_ts_device_system_virtual_generic_configuration_read_sequence/svt_pcie_ts_devi`

`ce_system_virtual_generic_configuration_write_sequence` primitive sequences. All enumeration sequences have also been enhanced to use generic configuration accesses as described here.

The following figure shows both front and backdoor access.

Figure 7-1 Front and Backdoor Access



7.11 Secondary PCI Express Extended Capability (SEC)

The current set of test cases verifies the following registers for the DUT as an UPSTREAM device only.

- Secondary PCI Express Extended Capability Header
- Lane Error Status Register
- Optional feature: lane error due to 8b10b decode error, Test case : pl_8b_10b_enc_cfg_error
- Framing error.
- Equalization Control Register

7.12 Scoreboard

PCIe Test Suite Scoreboard has a uniform scoreboarding architecture for [VIP - VIP/ VIP - DUT] modes, which compares:

- Inbound TLP (Received by DUT EP) with TLP VIP has transmitted.
- Outbound TLP (Transmitted by DUT EP) with TLP VIP has received.

The Test Suite environment provides analysis ports at an external APP BFM and DL layer. The RC VIP provides the write and response data.

Note the following:

- Address based transaction are stored into associative array with addresses as index.
- ID based transaction are stored in to associative array with Requester ID as index.

- Run phase uvm_post_shutdown_phase waits for associative arrays to be empty.

Test suite configuration provides some controlling points. By using those comparison they can be controlled. Table 7-4 shows the controlling points.

Table 7-4 Fields of TLP in Test Suite Configuration

Field name	Type	Default	Description
enable_scoreboard	bit	1'b1 => enable	Enable Scoreboard.
enable_sb_dut_tx_path	bit	1'b1 => enable	Enable Scoreboard in DUT Tx path.
enable_sb_dut_rx_path	bit	1'b1 => enable	Enable Scoreboard in DUT Rx path.
enable_sb_dut_tx_cpl_compare	bit	1'b1 => enable	Configuration to control comparison of completion TLPs generated by DUT.
enable_sb_dut_rx_cpl_compare	bit	1'b1 => enable	Configuration to control comparison of completion TLPs generated by VIP.
enable_sb_dut_tx_cfg_cpl_compare	bit	1'b0 => disable.	Configuration to control comparison of completion TLPs in response of Cfg transactions, generated by DUT.
enable_sb_dut_rx_cpl_no_data_compare	bit	1'b0 => disable.	Configuration to control comparison of received completion with no data TLPs
enable_sb_dut_tx_err_msg_compare	bit	1'b0 => disable.	Configuration to control comparison of error messages transmitted by DUT
check_sb_dut_tx_err_msg	bit	1'b1 => enable.	Configuration control to enable/disable protocol check for unexpected error messages transmitted by DUT.
enable_sb_dut_tx_tag_compare	bit	1'b0 => disable.	Configuration to control comparison of tag field generated by DUT. (TLP Header field)
enable_sb_dut_tx_rid_compare	bit	1'b1 => enable	Configuration to control comparison of requester id field generated by DUT. (TLP Header field)
enable_sb_dut_tx_cid_compare	bit	1'b1 => enable.	Configuration to control comparison of requester id field generated by DUT. (TLP Header field)
enable_sb_dut_rx_td_compare	bit	1'b0 => disable.	Configuration to enable comparison of TLP Digest field, which is generated by VIP. (TLP Header field)
enable_sb_dut_rx_rc_compare	bit	1'b0 => disable.	Configuration to enable comparison of TLP Error Poisoning field, which is generated by VIP. (TLP Header field)
enable_sb_dut_tx_rc_compare	bit	1'b0 => disable.	Configuration to enable comparison of TLP Error Poisoning field, which is generated by DUT. (TLP Header field)

Table 7-4 Fields of TLP in Test Suite Configuration (Continued)

Field name	Type	Default	Description
enable_sb_dut_rx_attr_id_order_compare	bit	1'b0 => disable.	Configuration to enable comparison of TLP ID Based ordering attribute field, which is generated by VIP. (TLP Header field)
enable_sb_dut_tx_byte_enable_compare	bit	1'b0 => disable.	Configuration to enable comparison of byte_enable field, which are generated by VIP. (TLP Header field). It's also controlling byte_count field.
enable_sb_dut_rx_reserved_bit_compare	bit	1'b0 => disable.	Configuration to enable comparison of reserved field, which are generated by VIP. (TLP Header field)
enable_sb_dut_rx_byte_count_compare	bit	1'b0 => disable	Configuration to enable comparison of TLP reserved field, which is generated by VIP
enable_sb_dut_rx_lower_address_compare	bit	1'b0 => disable	Configuration to enable comparison of TLP reserved field, which is generated by VIP
enable_sb_dut_rx_cid_compare	bit	1'b0 => disable	Configuration to enable comparison of TLP reserved field, which is generated by VIP

Because of the current VIP-IIP AXI based interface, some of fields cannot populate at the Application BFM side. Therefore, comparison of the following fields are disabled. The remaining fields of TLPs are compared.

- Inbound TLP: IDO (attr_id_order), TD, EP, BYTE_ENABLE (FBE and FBE), byte_count
- Outbound TLP: TAG, EP, BYTE_ENABLE (FBE and FBE)

Currently, the architecture of Scoreboard can be understood in two different modes.

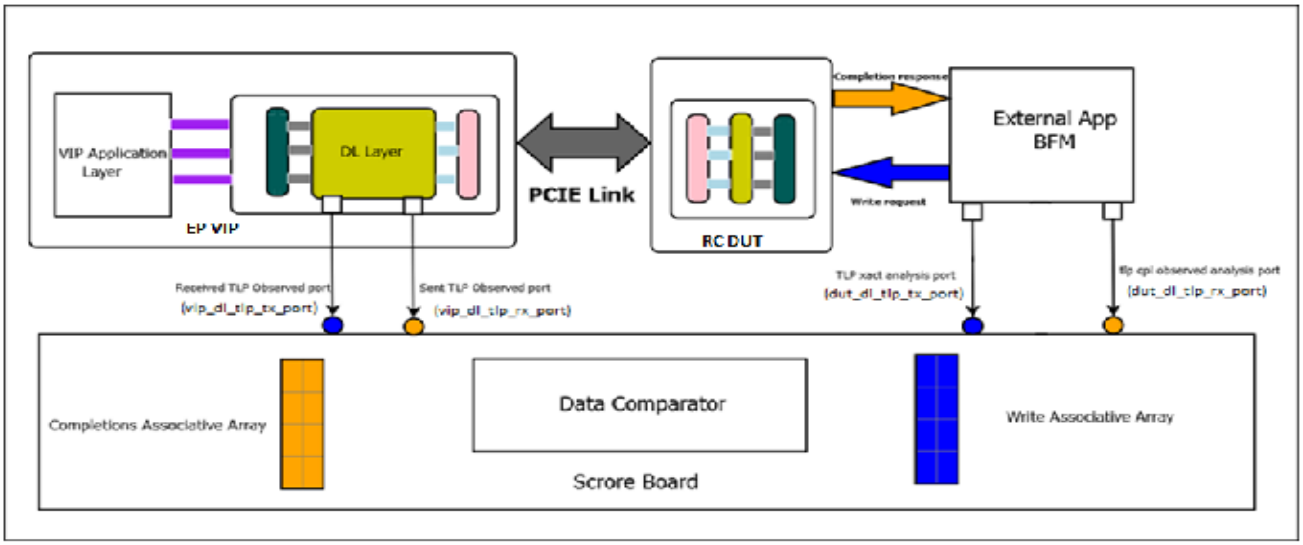
- VIP-DUT (RC) Mode
- VIP-VIP Mode

7.12.1 VIP-DUT (RC) Mode

Attention It is recommended to enable the Scoreboard so that errors are flagged in case any TLP compare failed. In case of miscompares on a DUT's TLP receive path, then the incorrectly filled TLP object received from the External Application BFM cannot be sent to the Application Agent for further processing.

VIP-DUT (EP) mode can be understood using Outbound traffic as an example. Refer to the following illustration Figure 7-2.

Figure 7-2 VIP-DUT (RC) Mode



For outbound write requests from an EP, store the data into a write associative array indexed with the transaction address through the analysis port available at External Application BFM. Once this request reaches the the RC VIP, you get it from a received TLP observed analysis port. Next, compare this against the data stored in the associative array.

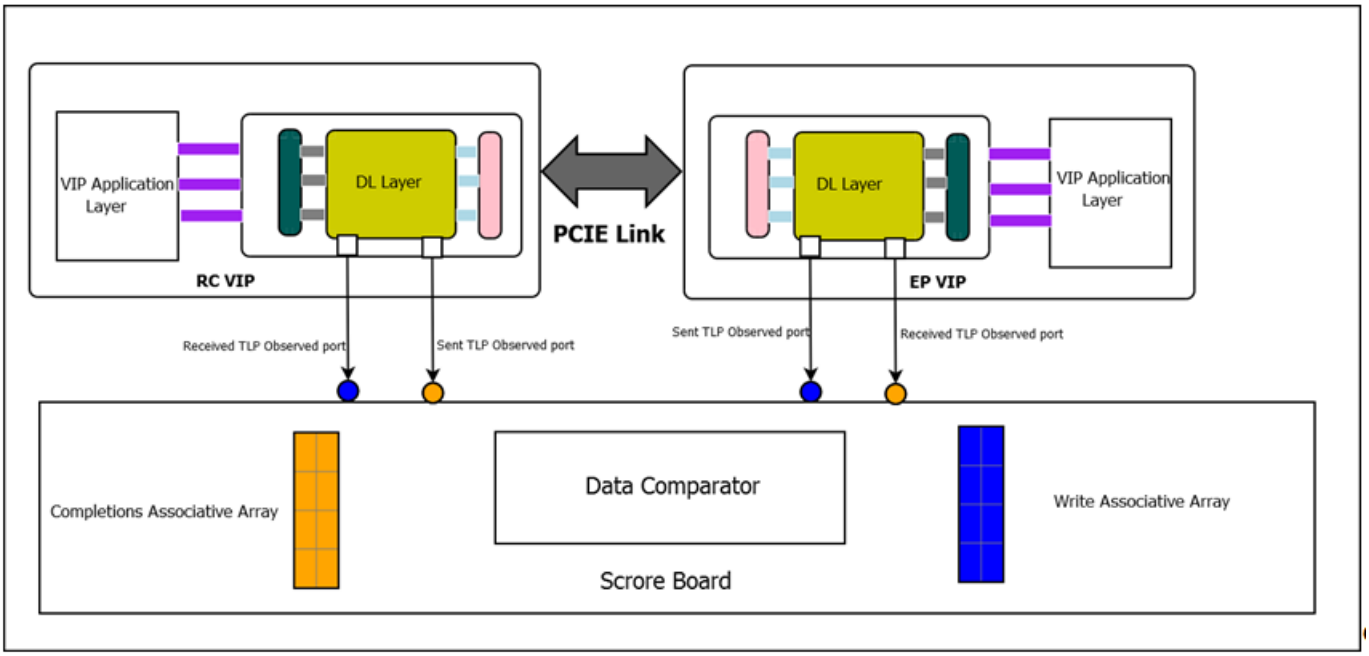
Once a read response is generated by the VIP, then you get it from the sent TLP observed analysis port. Next, store it into the completion associative array indexed with the transaction address.

Once this completion is received by an External Application BFM, pass it to the Scoreboard through a TLP CPL observed analysis port. Compare it against the data stored in completion associative array.

7.12.2 VIP-VIP Mode

This mode can be understood with Outbound traffic as an example. Consult the following illustration.

Figure 7-3 VIP-VIP Mode



For outbound write requests from the EP VIP, store the data into a write associative array indexed with the transaction address through analysis port available at in DL layer of EP VIP. Once this request reaches the RC VIP, get it from received a TLP observed analysis port, and compare this against the data stored in the associative array.

Once a read response is generated by the VIP, get it from sent TLP observed analysis port. Next, store it into the completion associative array indexed with the transaction address.

Once this completion is received by an EP VIP, pass it to Scoreboard through a TLP CPL observed analysis port. Compare it against the data stored in completion associative array.

7.13 Custom Applications

The Application Agent allows you to build your own application components and add them to the Application Agent just like existing Driver/Target applications. These applications will be able to receive inbound TLPs and schedule outbound TLPs through the Application Agent. Note that all outbound TLPs destined for transmission from the DUT need to pass through the TLP Mapper component present in the Application Agent.

The following example demo file includes all the required steps:
$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/pcie_test_suite_custom_tlp_mapper_svt/sverilog/src/vcs/svt_pcie_test_suite_custom_tlp_mapper.svp

The last two steps of this process are shown in the t*b_dut_pcie/env/snps_pcie_link_env.sv* example class.

Following are the required steps:

1. Create a callback class which decides what inbound TLPs (if any) need to be routed to the your custom application for processing instead of the default target: which the tlp_mapper class would otherwise determine. Note that any inbound TLP can only be mapped to 1 application ID by the TLP Mapper.

2. Add TX and RX TLM ports in your custom application to receive inbound TLPs and/or to schedule outbound TLPs.
3. Assign a unique application ID (> 32) to each of your custom applications. Note that first 32 `application_ids` are reserved for internal use.
4. If the custom application also needs to schedule outbound TLPs from the DUT, then add logic to push the transactions on the `tx_tlp_out_port` TLM put port of the custom application.
5. If the custom application also needs to receive inbound TLPs to the DUT, then add an implementation for the put method supporting '`rx_tlp_in_export`' export of the custom application. This method will be called every time the Application Agent receives an inbound TLP mapped to this application and it has the reference of the TLP object of interest.
6. Extend the TLP Mapper to start your custom application, create new ports and to connect with those in the custom application.
7. Create your custom application. Create new ports indexed by the unique `application_id` chosen for your custom application in step 3.
8. Connect the TLM ports created in steps 2 and 7.
9. Create an object of the callback class created in step 1, and add it to the pool as shown below.
10. Include the source file which contained the classes created in steps 1-9.
11. Establish factory override so that the type of `tlp_mapper` object created in Application Agent is that of the class created in step 6.

7.14 DUT Dependent Test Cases

The following table lists DUT dependent cases.

Table 7-5 DUT Dependent Cases

Test cases Name	DUT dependency
gen3_pl_config_complete_to_detect_error gen3_pl_recovery_rcvrcfg_to_detect_speed_ change_1_error gen3_pl_recovery_rcvrcfg_to_detect_ts2_non_ matching_rate_error	The following test cases verify features where the LTSSM state machine needs to loop for 255 times. The idle_to_rlock variable needs to be incremented until the value of 255. Due to this iteration, tests exceed the default test timeout value. As a result, all these following test cases timeout in the regression, and report themselves as failed. You need to increase the default test case timeout values, and run these test cases separately to get the coverage
gen3_pl_recovery_equalization_phase2_to_recovery_speed_ timeout_case1_error	Test expect to initiate preset request from EP DUT in Recovery.Equalization Phase 2 state.

Example 7-1 Example of Dependent Code

```
cfg.rc_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_entry_valid = 16'h7FF;
 cfg.rc_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_table[0]  =
{6'h0C,6'h24,6'h00};
```

```
 cfg.rc_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_table[1]  =
{6'h00,6'h28,6'h08};
----------
----------
 cfg.rc_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_table[10] =
{6'h0C,6'h24,6'h00};


 cfg.rc_cfg.pcie_cfg.pl_cfg.preset_to_coefficients_mapping_entry_valid = 16'h7FF;
 cfg.rc_cfg.pcie_cfg.pl_cfg.expected_preset_to_coefficients_mapping_table[0]  =
{6'h0C,6'h24,6'h00};
 cfg.rc_cfg.pcie_cfg.pl_cfg.expected_preset_to_coefficients_mapping_table[1]  =
{6'h00,6'h28,6'h08};
----------
----------
 cfg.rc_cfg.pcie_cfg.pl_cfg.expected_preset_to_coefficients_mapping_table[10] =
{6'h0C,6'h24,6'h00};
```

7.15 SRIS Usage

The following sections contains the usage of SRIS and SRNS.

7.15.1 SRIS

SRIS allows a worst case 5600 PPM difference (SSC with 5000 PPM difference + Tx/Rx crystal tolerance 600ppm). Following is the command to enable SRIS:

```
gmake <test_name> SVT_PCIE_SSC_MODE=SSC_ON_PPM_ON  SVT_PCIE_MAX_PPM=<+/- max osc ppm
value>  SVT_PCIE_MIN_PPM=< +/- min osc ppm value >
```

The command line option “SVT_PCIE_SSC_MODE=SSC_ON_PPM_ON” will introduce additional 5000PPM in clock as SSC.

For the oscillator frequency, the test will pick a random PPM value from the range provided by you in `SVT_PCIE_MIN_PPM` and `SVT_PCIE_MAX_PPM`.

The SKIP transmission interval requirement is handled by the VIP internally.

7.15.2 SRNS

The following command line is used to enable SRNS (Tx, Rx Refclk rates differ, allowing a worst case 600 PPM):

```
gmake <test_name> SVT_PCIE_SSC_MODE=SSC_OFF_PPM_ON  SVT_PCIE_MAX_PPM=<+/- max osc ppm
value>  SVT_PCIE_MIN_PPM=< +/- min osc ppm value >
```

The SSC_OFF_PPM_ON parameter enables SRNS mode; that is, there will not be any spread spectrum clocking.

For the oscillator frequency, the test will pick a random PPM value from the range provided by you in `SVT_PCIE_MIN_PPM` and `SVT_PCIE_MAX_PPM`.

The range should be within -600 PPM to +600 PPM which is the allowable tolerance limit for clock. If you want to set a specific/particular crystal PPM, then you must give the same value for min and max PPM command-line options.

For example, if you want to set -500 PPM then, the following must be given.

```
SVT_PCIE_MAX_PPM = -500 & SVT_PCIE_MAX_PPM = -500
```

You provide a value within the specification defined limit of +/-600PPM.

By default, the value will be chosen as +600PPM.

7.16 Loopback Usage

Loopback is applicable to SERDES mode only. As a result, Loopback cases are added in to serdes testbench areas only.

7.17 User Controllable Tolerance for LTSSM Time Outs

You can set the controllable tolerance value for all LTSSM time outs.

You must provide the percentage of the tolerance limit (maximum allowable tolerance is +50% as per specification) in base test as shown below:

```
// Set up the user defined percentage value for tolerance timeout on timout
//(2ms, 12ms, 24ms, 48ms ).
uvm_config_db#(int)::set(this, "env.test_env_seqr",
                      "ltssm_timeout_tolerance_percentage", 50);
```

The default value is set to 50% of the time out.

7.18 Transaction Ordering

The Transaction Ordering tests utilize the Ordering Application available as part of PCIe SVT VIP to perform Ordering rule related checks on the outbound traffic initiated from DUT. The tests initiate traffic to validate specific transaction ordering rules and the Ordering Application keeps track of all the scheduled TLPs. The application is added as a 'custom application' inside the TLP Mapper component. This is done by extending it and following the rules as described in section Custom Applications. The application compares the order in which VIP receives TLPs from the DUT with the scheduling order to validate two things:

- That the DUT does not violate any of the transaction Ordering Rules.
- That the DUT does not block TLPs eligible for transmission as per transaction Ordering Rules.

The Ordering Application may also be optionally used to perform Re-Ordering before the DUT. This can be enabled by setting `enable_reordering` property of the `svt_pcie_ordering_app_configuration` class. Its reference is present in the Test Suite configuration object. Also, set other properties of this class to appropriate values for smooth operation.

Note that if the DUT is not able to perform Re-Ordering by itself and the Ordering Application is instead enabled to do so, then the following tests will not be applicable. This is because these tests involve changing the order of outbound completions (or in some cases just holding them back for some time) which is DUT-implementation specific and cannot be guaranteed.

- `tl_transaction_ordering_rule_a5a`
- `tl_transaction_ordering_rule_b5_c5`
- `tl_transaction_ordering_rule_d2a`
- `tl_transaction_ordering_rule_d2b_ro`
- `tl_transaction_ordering_rule_d2b_ido_reqid`
- `tl_transaction_ordering_rule_d3_d4`
- `tl_transaction_ordering_rule_d5`
- `tl_transaction_ordering_rule_stress_test`

Following tests which validate functionality of IDO bit in relation to transaction Ordering are only valid for a DUT with more than 1 requester IDs (for a given bus number) - i.e. a multi-function DUT or EP DUT with SRIOV support - supporting IDO.

- `tl_transaction_ordering_rule_d2b_ido_reqid`
- `tl_transaction_ordering_rule_a2b_ido_reqid`
- `tl_transaction_ordering_rule_b2b_c2b_ido_reqid`

Following tests which validate functionality of RO bit in relation to transaction Ordering are only valid for a DUT with at least 1 function with RO support.

- `tl_transaction_ordering_rule_a2b_ro`
- `tl_transaction_ordering_rule_b2b_c2b_ro`
- `tl_transaction_ordering_rule_d2b_ro`

The following tests require the DUT to be capable of transmitting AtomicOp transactions, in which case `enable_atomic_op_as_requester_support` bit must be set in the `dut_capabilities` configuration object.

- `tl_transaction_ordering_rule_b3_b4_c3_c4`

The PCIe Test Suite currently delivers 17 Transaction Ordering Rule tests and all of their names start with `tl_transaction_ordering_rule_` prefix.

7.19 Equalization Phase3 to Recovery.Speed Transition in VIP-to-IIP Mode

Perform the following steps if you are a SNPS IIP user.

1. In PIPE mode:
   a. Invoke preset request from RC DUT.
   b. Not required to configure anything as PHY is VIP (taken care in the test itself, additional settings are not required).
2. In SERDES mode:
   a. In *IIP_DUT_PCIE_RC_x*.tcl*, you need to set the following variable:
   `CX_GEN3_EQ_COEF_CONV_SUPPORTED=1`
   b. Changes in *pcie_rc_dut_external_app_bfm.sv*.

```
        task
`SVT_PCIE_TEST_SUITE_RC_DUT_EXTERNAL_APP_BFM_TYPE::equalization_setup(svt_pcie_device_c
onfiguration cfg);
          uvm_status_e status;
          uvm_reg_data_t data;
          uvm_reg      reg_obj;

        `ifdef SERDES_TB
          if(cfg.pcie_cfg.pl_cfg.min_eq_preset_coeff_validation_delay[0] > 0) begin
            //[23:8] -> Preset Request Vector = 16'h0011
            //[3:0]  -> Feedback Mode.0000b: Direction Change = 4'h0
            write_reg("GEN3_EQ_CONTROL_OFF",32'h00001100, 32'h00FFFF0F);

            //[4:0] -> Minimum Time (in ms) To Remain EQ Master Phase.
            data = 32'h1f;
            write_reg("GEN3_EQ_FB_MODE_DIR_CHANGE_OFF",data, 32'h0000001F);
        end
```

```
    `endif

endtask: equalization_setup
```

c. Invoke preset request from RC DUT.

Perform the following steps if you are a Non-SNPS IIP user.

1. In PIPE mode:
   - ✦ Invoke preset request from RC DUT.
   - ✦ Not required to configure anything as PHY is VIP (taken care in the test itself, additional settings are not required).
2. In SERDES mode:
   - ✦ Invoke preset request from RC DUT.
   - ✦ Configure RC evaluation timer same as phase3 timeout value so that RC continues evaluating coefficients and timeout will occur.

7.20 Changing the Link Width (Target Link Width Option)

You can use the following command-line option to control the target link width.

```
SVT_PCIE_TARGET_LINK_WIDTH
```

It can be passed with the value "1/2/4/8/12/16/32" The Link width for RC and EP will be configured according to the passed value. But the passing value should be equal to or less than the value of `SVT_PCIE_MAX_LINK_WIDTH`. The following table shows the relationship between `SVT_PCIE_TARGET_LINK_WIDTH` and `SVT_PCIE_MAX_LINK_WIDTH`.

SVT_PCIE_MAX_LINK_WIDTH	SVT_PCIE_TARGET_LINK_WIDTH	Final Link-up
Any possible value from : 4/16/32 Example: 32	1/2/4/8/12/16/32	Link-up will be based on Target link width value
Not given in command line Default is 16)	Any possible value from: 1/2/4//8/12/16	Link-up will be based on Target link width value
Any possible value from: 4/16/32.	Not given in command line	Link-up will be based on Max link width value
Value less then Target link width Example: 4	16	Fatal Error

In this release, `SVT_PCIE_TARGET_LINK_WIDTH` option support is provided only to PIPE TB area.

7.21 Verdi Spec Linking Feature Usage

PCIe Test Suite delivers the test plans based on Verdi specification linking feature. The tests verifying the DUT PCIe protocol features are linked to the specification document. You can open plans in Verdi and view the mapping specification in Verdi GUI.

The plans also support back annotation of test pass/fail metric based on the regression results.

7.21.1 PCIe Test Suite Plan Deliverables

7.21.1.1 Directory

The plans are available at the following location in the installation directory:

$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/version/doc/VerificationPlans/

Note It is recommended to save a copy of these plans in your local directory before loading the plans into Verdi Coverage tool.

7.21.1.2 Naming Convention for Plans

7.21.1.2.1 Sub-plans

Sub-plans are created by copying the corresponding template and renaming as mentioned below.

`< Protocol >_dut_<dut_type>_<subplan_name>_<plan_category>_subplan`

Where,

Protocol = name of the VIP

dut_type = type of dut. Example: ep | rc

plan_category = test

subplan_name = can be one of following as per the level at which a sub plan is present in test plan hierarchy:

At Level2: <<spec name>_<chapter _name>>

Example:

`pcie_dut_rc_pci_express_base_spec_r3_1_power_management_test_subplan.hvp`

At Level3: <<chapter _name>_<functionality>>

Example:

`pcie_dut_rc_power_management_test_subplan.hvp`

Note You can organize sub-plans in a hierarchy. For example, a sub-plan can be created for each chapter/layer in the specification and each of these sub-plans may have multiple sub-plans available for a specific functionality.

- Use name as `spec_name` and layer name if the sub-plan is for a specific layer.
- Use `layer_name` and functionality name if the sub-plan represents a specific functionality described by the layer. This sub-plan may be the sub-plan of layer-wise sub-plan.

Examples:

```
pcie_dut_rc_pci_express_base_spec_r3_1_transaction_layer_test_subplan
pcie_dut_rc_transaction_layer_aer_test_subplan
pcie_dut_rc_transaction_layer_flow_control_test_subplan
```

7.21.1.2.2 Top Level Plan Naming Convention

A top level plan is the plan which includes more than one sub-plans. These sub-plans might have been linked to different specifications.

Each plan category will have a top level plan.

```
< protocol>_dut_<dut_type>_<plan_category>_toplevel_plan
```

Where,

protocol = pcie

dut_type = ep | rc

plan_category = test

Example:

```
pcie_dut_rc_test_toplevel_plan
```

7.21.2 Requirements to Setup the Verdi Spec Linking Flow

- Verdi tool Supported: verdi_2015.09 onwards
- Download the PCIE specification documents from the PCIE-SIG with the version mention below:
  - PCIe Gen3: PCI Express® Base Specification Revision 3.1, October 8, 2014
  - PCIe Gen4: PCI Express® Base Specification Revision 4.0 Version 0.5 February 6, 2015
  - SRIOV (Single Root I/O Virtualization and Sharing Specification Revision 1.1)
  - ATS (Address Translation Services Revision 1.1)
- Test pass/Fail Results text file inclusion.
  - The Verdi tool need the pass/fail status from the Regression results. Use the regression results to produce the test status information in a text file. Use the first line of the text file as it is, following that is the test file status information. In the text file, HVP metric = test is the first line, the test is a in-built Verdi metric of type enum with possible values of pass, fail, unknown.

```
HVP metric = test
test_case_name = pass|fail|unknown|...
```

Example:

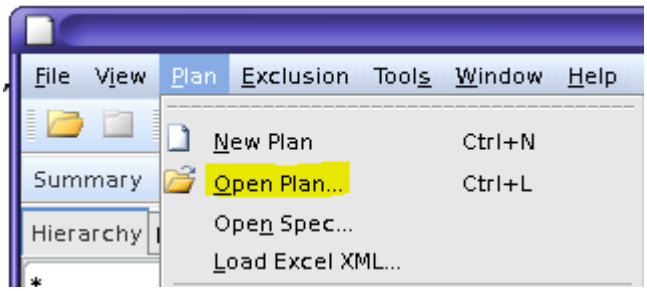
```
HVP metric = test
dl_ack_collapsed = pass
dl_ack_nack_latency_timer = fail
```

7.21.3 Commands and Guidelines to View the Test Suite with Specification

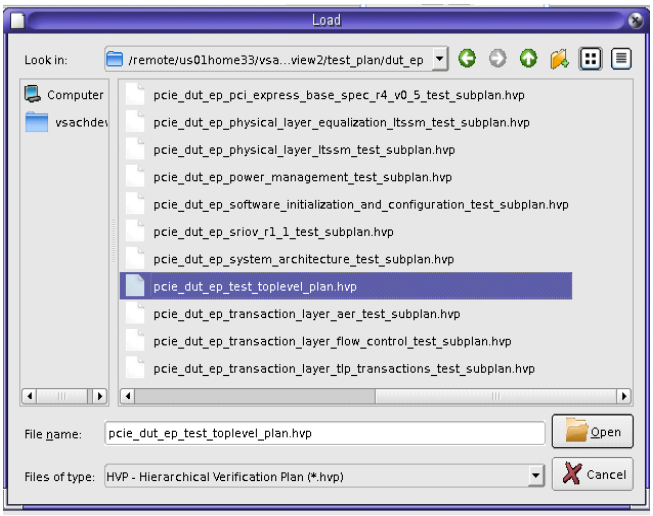
- Invoke Verdi
  - To invoke Verdi Coverage, you need to add the `-cov` option to the Verdi command line.

```
> verdi -cov &
```

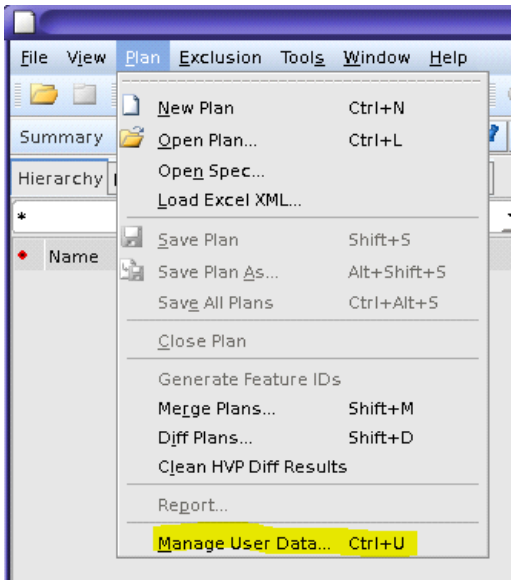
- Select the Open Plan option from the Plan menu as shown in the following figure.



- The Load dialog box appears. To import the Hierarchical Verification plan (.hvp) file, navigate to the directory and click Open after selecting the plan file. The following figure shows selection of EP top level *.hvp* file (*pcie_dut_ep_test_toplevel_plan.hvp*).

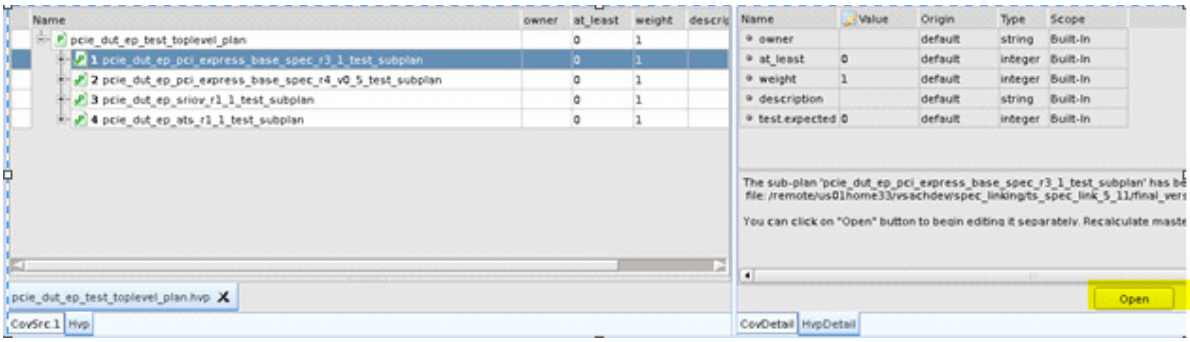


- Select the Manage User Data option from the Plan menu to load the regression result text file (Pass/Fail status information).

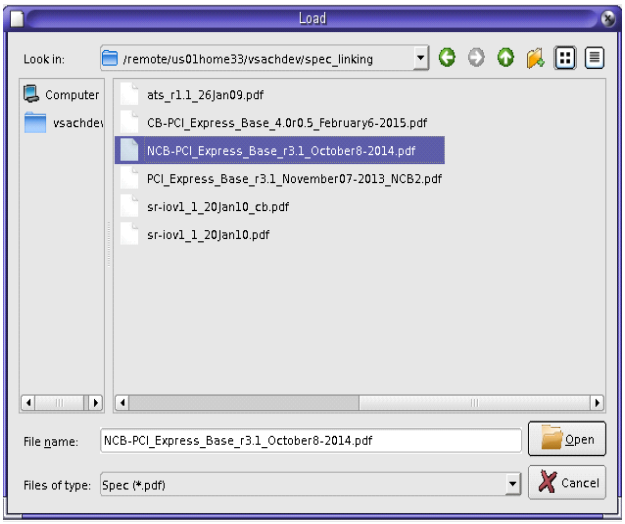


- The Load/Unload User Data dialog box appears. To import the regression results file, click the Load button. Click OK after selecting the text file.

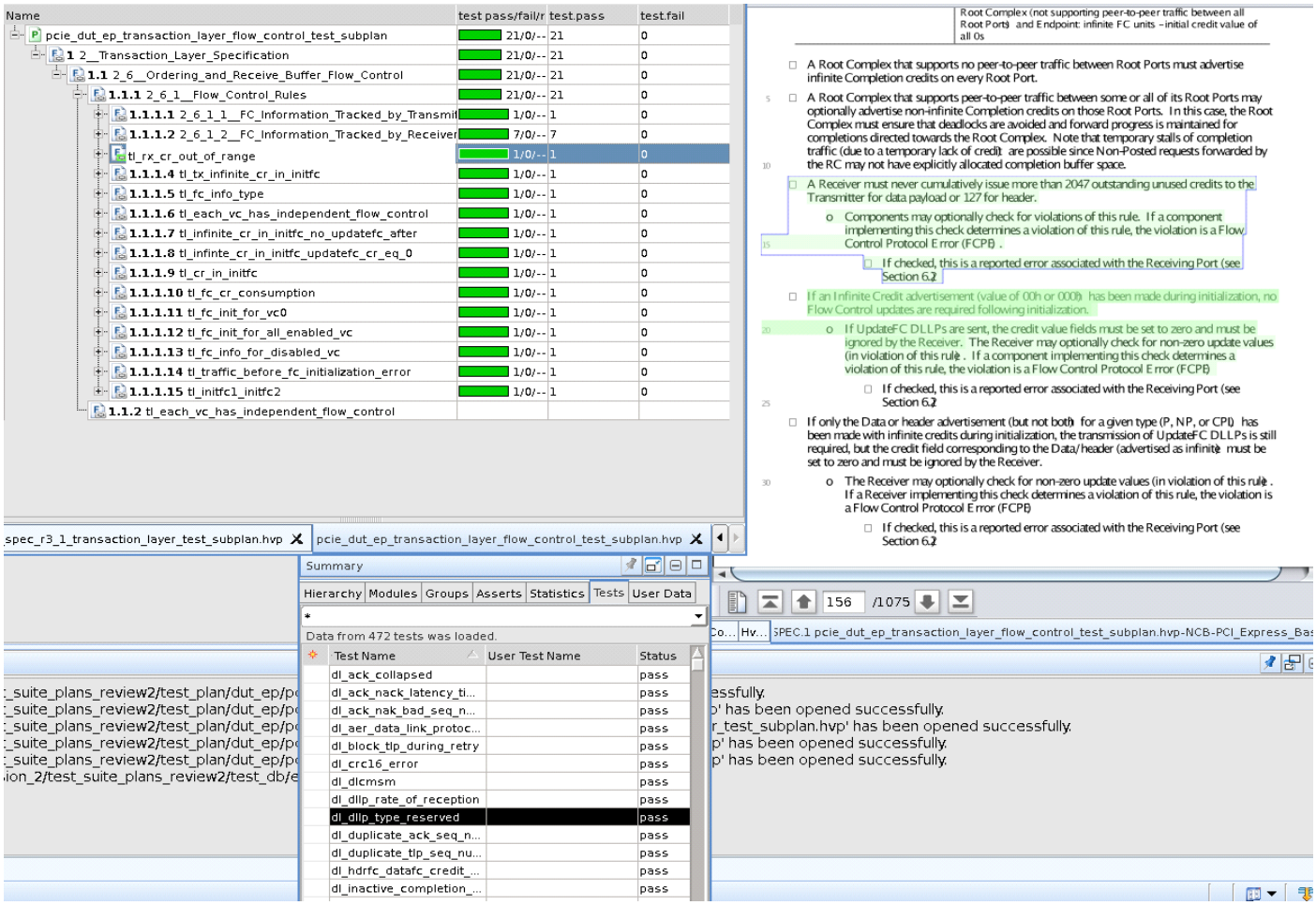
- In the Hvp dialog box, select the *.hvp* file you want to load from the spec linked tool and click Open.



- The Load dialog box appears. Select the specification version you want to load and click Open to open the Verdi spec linked tool.



- The following figure shows the complete view of Verdi spec linked tool which includes the Test back annotation results (Summary), HVP sub-plan and specification which covers the attributes for the respective test (highlighted in green).



For more details, refer to Verdi Coverage User Guide and Tutorial.

7.22 Enabling EIEOS Pattern for Gen4

Added support for new 16G EIEOS pattern:

- To enable 16G EIEOS pattern for Gen4, set the `SVT_PCIE_TEST_SUITE_ENABLE_EIEOS_16G_0000_FFFF` macro to 1. This will invoke VIP to transmit and expect new 16G EIEOS pattern. By default, it is set to 0.

```
`define  SVT_PCIE_TEST_SUITE_ENABLE_EIEOS_16G_0000_FFFF 1
```

7.23 Test Suite Specific Command-Line Options

The following table lists command-line options.

Table 7-6 Test Suite Specific Command Line Option

Command Line Option	Corresponding plusarg	Description	Interface type	Test case Applicable
SVT_PCIE_MAX_LINK_ WIDTH =4		Select (x4/x16/x32) link_width configuration (Default is x32)	PIPE/Serial	All Test
SVT_PCIE_ENABLE_ TRANSACTION_ LOGGING=1	+svt_pcie_enable_transaction_logging	To enable transaction logging and Symbol logging	PIPE/Serial	All Test
SVT_PCIE_ENABLE_PA=1	+svt_pcie_enable_pa	Enable protocol analyzer xml generation	PIPE/Serial	All Test
ENABLE_PCIE_ COVERAGE=1	+svt_pcie_enable_coverage	Enable the coverage	PIPE/Serial	All Test
SVT_PCIE_ENABLE_ BACKDOOR_CFG_ UPDATES=1	+svt_pcie_enable_backdoor_cfg_updates	This option will cause tests to initiate cfg_database_service_transactions for doing read/write operations on DUT's configuration registers instead of frontdoor CfgRd/Wr on PCIe Link.	PIPE/Serial	All Test*
SVT_PCIE_LINK_SPEED =<GEN1|GEN2|GEN3|GEN4>	+svt_pcie_link_speed=<GEN1|GEN2|GEN3>|GEN4	Force a test to run on gen1/gen2/gen3/gen4 speed	PIPE/Serial	Specific cases**
SVT_PCIE_TEST_ TIMEOUT		To set the test stop time (default 1200000ns)	PIPE/Serial	All Test
SVT_PCIE_ENABLE_POLARITY_INVERSION	+svt_pcie_enable_polarity_inversion	To enable the polarity inversion feature	Serial	All Test
SVT_PCIE_SSC_MODE=<SSC_OFF_PPM_ON| SSC_ON_PPM_ON>	+svt_pcie_ssc_mode<SSC_OFF_PPM_ON| SSC_ON_PPM_ON>	To enable the SSC mode SSC_OFF_PPM_ON: Provides Positive or Negative ppm(+/- 600) will be randomly chosen SSC_ON_PPM_ON: Provides Positive or Negative ppm(+/- 600) will be randomly chosen and default 0 to -5000ppm for SSC.	Serial	All Test
SVT_PCIE_DISBALE_ SRIOV=1		If the DUT supports SRIOV but it is desired that TL layer tests not target VFs randomly, then 'SVT_PCIE_DISBALE_SRIOV=1' command line switch should be provided. If this switch is provided then the activate_link sequence will not enable VFs in the SRIOV extended capability structure(s).	PIPE/Serial	All TL cases

Table 7-6 Test Suite Specific Command Line Option (Continued)

Command Line Option	Corresponding plusarg	Description	Interface type	Test case Applicable
SVT_PCIE_ENABLE_ EXTERNAL_APP_BFM=1		Enables the External Application BFM Use Model	PIPE/Serial	All tests
SVT_PCIE_ENABLE_ LANE_SKEW=1		User can enable lane to lane skew insertion feature in random lanes with spec. defined allowable limit.	PIPE/Serial	All tests
SVT_PCIE_TARGET_LINK_WIDTH=1/2/4/8/12/16/32	+svt_pcie_target_link_width	Whatever value is passed to this command line option, the RC and EP is configured in that link width. But, that value should be equal to or less than SVT_PCIE_MAX_LINK_WIDTH. If not, then test case will exit with FATAL Error	PIPE	All tests
SVT_PCIE_PIPE_WIDTH	+svt_pcie_pipe_ width	To enable the different PIPE width configuration. (Possible value 0, 1, and 2). to SVT_PCIE_PIPE_WIDTH to Value 0, 1 & 2 means PIPE_WIDTH of 8, 16 & 32 respectively. In demo mode this pipe width fixed for both side EP as well as RC side, while in DUT mode, this pipe width set for VIP. By default, PIPE width is constant in X4, and its dynamic for X16 mode.	PIPE	All tests
SVT_PCIE_TS_AXI_ WIDTH		To Enable AXI interface with either 128-bit or 32-Bit. Value <32|128> to this option, corresponds to width AXI Bit.	PIPE/Serial	All tests
SVT_PCIE_TEST_SUITE_PIPE_SPEC_VER_4_2_PCLK_INPUT		To enable PIPE Spec 4.2 with PCLK as PHY Input.	PIPE	All tests
SVT_PCIE_TEST_SUITE_PIPE_SPEC_VER_4_3		To enable PIPE 4.3 support in Test Suite	PIPE	All tests
SVT_PCIE_ENABLE_10_ BIT_TAGS		To enable 10-bit tag support from VIP.	PIPE/Serial	All Tests
SVT_PCIE_MAX_PPM	+svt_pcie_max_ ppm	This introduces the maximum oscillator ppm in Tx Bit clk.Testbench will choose a value between svt_pcie_max_ppm and svt_pcie_min_ppm randomly and introduce that ppm shift in Tx bit clk.	Serial	PL cases

Table 7-6 Test Suite Specific Command Line Option (Continued)

Command Line Option	Corresponding plusarg	Description	Interface type	Test case Applicable
SVT_PCIE_MIN_PPM	+svt_pcie_min_ppm	This introduces the minimum oscillator ppm in Tx Bit clk.Tes bench will choose a value between svt_pcie_max_ppm and svt_pcie_min_ppm randomly and introduce that ppm shift in Tx bit clk.	Serial	PL cases
** These tests are: -For tl_*/pl_*/dl_* cases, all Gen1/Gen2/Gen3/Gen4 speed is applicable -For gen2_pl_* cases, Gen2, Gen3, and Gen4 speed applicable -For gen3_pl_* cases, only Gen3/Gen4 speed applicable For gen4_pl_* cases, only Gen4 speed applicable				

Example:

```
gmake <test_case_name> USE_SIMULATOR=<simulator_type> SVT_PCIE_MAX_LINK_WIDTH=<4|16|32>
```

7.24 Partition Compile Support

Partition Compile is a VCS MX feature that allows you to compile portions of the design getting significantly faster turnaround times during the iterative process of compile and recompile.

With Partition Compile, you specify partitions in your design that you expect to revise and recompile often. VCS MX recompiles only the modified partitions.

The PCIe Test Suite provides a partition compile configuration file (*pc.optcfg*) which has partitions defined for the Synopsys Test Suite environment. You can add adjust partitions for their DUT to fine tune the partitioning.

Following are the Synopsys provided partitions.

```
pc.optcfg for serial testbenches(tb_dut_*_serdes):
    partition package svt_pcie_test_suite_uvm_pkg;
    partition package svt_pcie_uvm_pkg;
    partition cell pciesvc_device_serdes_x4_model_8g;
    partition cell pciesvc_device_serdes_x16_model_8g;
    partition cell pciesvc_device_serdes_x32_model_8g;
    partition cell pciesvc_device_application_model;
    partition cell <Top DUT module name>;
pc.optcfg for serial testbenches(tb_dut_*_pipe):
    partition package svt_pcie_test_suite_uvm_pkg;
    partition package svt_pcie_uvm_pkg;
    partition cell pciesvc_device_mpipe_x4_model_8g;
    partition cell pciesvc_device_spipe_x4_model_8g;
    partition cell pciesvc_device_mpipe_x16_model_8g;
    partition cell pciesvc_device_spipe_x16_model_8g;
    partition cell pciesvc_device_mpipe_x32_model_8g;
    partition cell pciesvc_device_spipe_x32_model_8g;
    partition cell pciesvc_device_application_model;
    partition cell <Top DUT module name>;
```

You can run tests in the partition compile mode using the `USE_SIMULATOR=vcspcvlog` command-line switch.

7.25 Link Width Dependent Tests

Table 7-7 shows the Link Width dependent tests.

Table 7-7 Link Width Dependent Tests

S.No.	Test Name	Not applicable for Link Width
1	gen3_pl_recovery_equalization_phase0_to_phase1_case1_error	1
2	gen3_pl_recovery_equalization_phase0_to_recovery_speed_timeout_case1_error	1
3	gen3_pl_recovery_equalization_phase1_to_rcvrlock_case1_error	1
4	gen3_pl_recovery_equalization_lw_down_error	1
5	gen3_pl_recovery_equalization_phase3_to_rcvrlock_case1_error	1
6	gen4_pl_recovery_equalization_lw_down_error	1
7	gen4_pl_recovery_equalization_phase0_to_phase1_case1_error	1

Table 7-7 Link Width Dependent Tests

8	gen4_pl_recovery_equalization_phase0_to_recovery_speed_timeout_case1_error	1
9	gen4_pl_recovery_equalization_phase1_to_rcvrlock_case1_error	1
10	gen4_pl_recovery_equalization_phase3_to_rcvrlock_case1_error	1
11	pl_config_linkwidth_start_to_config_linkwidth_accept_upconfigure_link_width_down_random	1
12	pl_polling_active_to_polling_compliance_partial_lanes_timeout_error	1
13	gen2_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_corruption_error	1
14	gen2_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_link_lane_mismatch_error	1
15	gen3_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_corruption_error	1
16	gen3_pl_recovery_rcvrlock_to_recovery_rcvrcfg_timeout_link_lane_mismatch_error	1
17	pl_l0_to_recovery_linkwidth_change_error	1
18	pl_config_linkwidth_start_to_config_linkwidth_accept_upconfigure_link_width_down_error	1
19	pl_config_linkwidth_start_to_config_linkwidth_accept_upconfigure_link_width_up_error	1
20	gen4_pl_logidle_corruption_on_multilane_in_configuration_idle_error	1
21	gen4_pl_logidle_corruption_on_multilane_in_recovery_idle_error	1
22	gen4_pl_logidle_corruption_on_multilane_in_l0_error	1
23	pl_config_lanenum_accept_to_config_lanenum_wait_link_width_down_case2_error	< 4
24	gen3_pl_multi_sdp_in_symbol_time_error	<= 4
25	pl_polling_active_to_polling_config_compliance_bit_0_timeout_error	1
26	pl_polling_active_to_polling_config_loopback_bit_1_timeout_error	1
27	pl_polling_active_to_polling_config_ts2_timeout_error	1
28	gen3_pl_idle_followed_by_no_idle_next_lane_error	1

7.26 Support for PIPE 4.3 and PIPE 4.2

The testbench supports the following topologies:

- PCLK as PHY Output (Normal mode)
- PCLK as PHY Input

Follow these steps to enable the topologies.

7.26.1 PCLK as PHY Output (Normal mode)

7.26.1.1 To Enable PIPE 4.2

This is the default mode in Test Suite. No changes or addition command-line options are required. All signals under "*PCIESVC_PIPE_SPEC_VER_GTR_4_0*" macro should be connected. This was already present in earlier release. There is no change in it.

7.26.1.2 To Enable PIPE 4.3

a. Command-line option:

You must pass the following command-line option:

`SVT_PCIE_PIPE_SPEC_VER_4_3=1`

b. DUT connection: In PIPE 4.3 mode, PowerDown is converted from a 3 bit to a 4 bit signal, and an extra port "AsyncPowerChangeAck" is added. You must connect these with the EP (MAC) DUT with following way.

Following is an example which shows an X16 connection with a DUT. Similarly it can be connected with a X16 and x32 Link width as well.

Example 7-2 X16 connection

```
`ifdef PCIESVC_PIPE_SPEC_VER_4_3
wire [3:0] endpoint0_powerdown;
wire endpoint0_async_power_change_ack;
`else
wire [2:0] endpoint0_powerdown;
`endif
//VIP connection
pcie_mpipe_x16_interconnect pcie_mpipe_x16_interconnect(
.reset (test_top.reset),
......
......
.powerdown (endpoint0_powerdown),
`ifdef PCIESVC_PIPE_SPEC_VER_4_3
.async_power_change_ack (endpoint0_async_power_change_ack),
`endif
 // DUT connection:
`ifdef PCIESVC_PIPE_SPEC_VER_4_3
assign
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_powerdow
n = <MAC output4-bit powerdown dignal>;
assign
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_async_po
wer_change_ack = <MAC output AsyncPowerChangeAck >
```

```
`else
assign
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_powerdow
n = <MAC output 3-bit powerdown dignal>;
`endif
```

7.26.2 PCLK as PHY Input

7.26.2.1 To Enable PIPE 4.2 PCLK as PHY Input

a. Command-line option:

You must pass the following command-line option:

```
SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT=1
```

b. DUT connection:

- All signals which is defined under “PCIESVC_PIPE_SPEC_VER_GTR_4_0”, should be connected.
- For PIPE 4.2 PCLK as PHY input, PCLK has become the input to the PHY and also there are two more signals “PclkChangeOk” (output of PHY) and “PclkChangeAck” (Input of PHY) added. Therefore, the connection will be as follows:

The following code shows a X16 connection:

Example 7-3 X16 connection

```
`ifdef SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT
      wire              endpoint0_pclk_change_ok;
      wire              endpoint0_pclk_change_ack;
`endif // ifdef SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT

`ifdef PCIESVC_PIPE_SPEC_VER_GTR_4_0
      wire              endpoint0_rx_eq_in_progress_0;
      wire              endpoint0_rx_eq_in_progress_1;
    ......
    ......
      wire              endpoint0_rx_eq_in_progress_15;
      wire [5:0]        endpoint0_lf_0;
      wire [5:0]        endpoint0_fs_0;
      wire [5:0]        endpoint0_lf_1;
      wire [5:0]        endpoint0_fs_1;
    ......
    ......
      wire [5:0]        endpoint0_lf_15;
      wire [5:0]        endpoint0_fs_15;
`else
      wire [5:0]        endpoint0_lf;
      wire [5:0]        endpoint0_fs;
`endif
// VIP connection
   pcie_mpipe_x16_interconnect pcie_mpipe_x16_interconnect(
              .reset                  (test_top.reset),
`ifdef SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT
                              .pipe_clk_0           (endpoint0_pipe_clk),
                              .pipe_clk_1           (endpoint0_pipe_clk),
    ......
```

```
    ......
                              .pipe_clk_15          (endpoint0_pipe_clk),
                              .pclk_change_ok       (endpoint0_pclk_change_ok),
                              .pclk_change_ack      (endpoint0_pclk_change_ack),
`else
                             .pipe_clk             (endpoint0_pipe_clk),
`endif // ifdef SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT
                .......
                .......

`ifdef PCIESVC_PIPE_SPEC_VER_GTR_4_0
.rx_eq_in_progress_0  (endpoint0_rx_eq_in_progress_0),
.rx_eq_in_progress_1  (endpoint0_rx_eq_in_progress_1),
                .......
                .......
.rx_eq_in_progress_15  (endpoint0_rx_eq_in_progress_15),
.lf_0  (endpoint0_lf_0),
.fs_0  (endpoint0_fs_0),
.lf_1  (endpoint0_lf_1),
.fs_1  (endpoint0_fs_1),
                .......
                .......
.lf_15  (endpoint0_lf_15),
.fs_15  (endpoint0_fs_15),
`else
 .lf   (endpoint0_lf),
 .fs   (endpoint0_fs),
`endif

// DUT connection

`ifdef PCIESVC_PIPE_SPEC_VER_GTR_4_0
assign
{dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_rx_eq_in_progress_0,
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_rx_eq_in_progress_1,
. . . . .
              . . . . .
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_rx_eq_in_progress_15}
= <MAC output with PIPE 4.2 RxEqInProgress signal>;

assign
{dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_fs_0,
 dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_fs_1,
                . . . . .
                . . . . .
 dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_fs_15 }
  =<MAC output PIPE 4.2 FS signal>;

assign
{dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_lf_0,
 dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_lf_1,
                . . . . .
                . . . . .
 dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_lf_15}
  = <MAC output PIPE 4.2 LF signal>;

`else
    assign
```

```
      dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_fs

       = <MAC output FS signal>;
    assign
     dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_lf

      = <MAC output LF signal>;
`endif
```

7.26.2.2 To Enable PIPE 4.3 PCLK as PHY Input

a. Command-line options:

   You must pass the following command-line option:

   ```
   SVT_PCIE_PIPE_SPEC_VER_4_2_PCLK_INPUT=1
   SVT_PCIE_PIPE_SPEC_VER_4_3=1
   ```

b. DUT connection: All connection for 4.2 PCLK as PHY input, it will require all PIPE 4.3 connections to be added.

   The following codes shows a X16 connection.

Example 7-4 X16 connection

```
   `ifdef PCIESVC_PIPE_SPEC_VER_4_3
      wire [3:0] endpoint0_powerdown;
      wire endpoint0_async_power_change_ack;
   `else
      wire [2:0] endpoint0_powerdown;
   `endif
   //VIP connection
   pcie_mpipe_x16_interconnect pcie_mpipe_x16_interconnect (
          .reset (test_top.reset),
          ......
          ......
          .powerdown (endpoint0_powerdown),
   `ifdef PCIESVC_PIPE_SPEC_VER_4_3
          .async_power_change_ack (endpoint0_async_power_change_ack),
   `endif
   // DUT connection:
   `ifdef PCIESVC_PIPE_SPEC_VER_4_3
      assign dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_powerdown =
<MAC output4-bit powerdown dignal>;
      assign
dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_async_power_change_ack  =
<MAC output AsyncPowerChangeAck >
   `else
      assign dut_connect_at_x16_pipe.pcie_mpipe_x16_interconnect.endpoint0_powerdown =
<MAC output 3-bit powerdown dignal>;
   `endif
```

7.27 Configuration Space Tests

In the current release, Synopsys provides the following configuration space test cases to verify different attributes of PCIe configuration space inside DUT.

1. *ts.tl_cfg_capability_check.sv*: This test case reads out the whole configuration space from the DUT, and verifies whether all PCIe capabilities implemented by the DUT are present once inside a function.
2. *ts.tl_cfg_valid_range_test.sv*: This test reads out the whole configuration space from the DUT and verifies that no register and field advertise any reserved value as per the PCIe specification.

Note These tests must be run just after activating the PL-DL linkup.

7.28 Application Error Reporting Interface Test (Only for SNPS IIP)

- You must set the `APP_RETURN_ERR_EN` configuration parameter to enable the Application Error Reporting Interface in IIP TCL (configuration file).
- From Application BFM, you must invoke the `application_error_reporting_handler` task when the TLP is received at its application interface.

Note The `` `CX_SRIOV_ENABLE `` macro is not applicable for RC DUT. Therefore, signals related to *_vfunc_* will be ignored.

7.28.1 Configuring the *env* to use the Application Error Reporting

- Changes in DUT specific interface file (*rtl_dut_specific_if.svi*).

  You must declare the following DUT-specific interface signals:

```
// Application Error Reporting.
logic app_hdr_valid;
logic [127:0] app_hdr_log;
logic [11:0] app_err_bus;
logic [7:0] app_err_func_num;
logic app_err_advisory;
logic app_err_vfunc_active;
logic [7:0] app_err_vfunc_num;
```

- Changes in DUT Wrapper file (*pcie_dut_sv_wrapper.sv*)
  - DUT-specific signals:

```
bit            dut_app_hdr_valid  ;
bit [127:0]    dut_app_hdr_log  ;
bit [11:0]     dut_app_err_bus  ;
bit [NF-1:0]   dut_app_err_func_num  ;
bit            dut_app_err_advisory  ;
bit            dut_app_err_vfunc_active;
bit [NVF-1:0]  dut_app_err_vfunc_num  ;
```

  - Connect all Application-specific error signals to DUT specific interface:

```
assign dut_app_hdr_valid              = dut_specific_if.app_hdr_valid;
assign dut_app_hdr_log                = dut_specific_if.app_hdr_log;
assign dut_app_err_bus                = dut_specific_if.app_err_bus;
assign dut_app_err_func_num           = dut_specific_if.app_err_func_num;
assign dut_app_err_advisory           = dut_specific_if.app_err_advisory;
assign dut_app_err_vfunc_active       = dut_specific_if.app_err_vfunc_active;
assign dut_app_err_vfunc_num          = dut_specific_if.app_err_vfunc_num;
```

- Connection with DUT:

```
`ifdef APP_RETURN_ERR_EN
    .app_hdr_valid              (dut_app_hdr_valid),
    .app_hdr_log                (dut_app_hdr_log),
`ifdef CX_TLP_PREFIX_ENABLE
    .app_tlp_prfx_log           (dut_app_tlp_prfx_log),
`endif  // CX_TLP_PREFIX_ENABLE(dut_
    .app_err_bus                (dut_app_err_bus),
    .app_err_advisory           (dut_app_err_advisory),
    .app_err_func_num           (dut_app_err_func_num),
`ifdef CX_SRIOV_ENABLE // Not applicable to RC DUT.
    .app_err_vfunc_num           (dut_app_err_vfunc_num),
    .app_err_vfunc_active       (dut_app_err_vfunc_active),
`endif // CX_SRIOV_ENABLE
`endif // APP_RETURN_ERR_EN
```

- Changes in Application BFM file (*pcie_rc_dut_external_app_bfm.sv*)
  - You must invoke the `application_error_reporting_handler` method to drive the Application Error Reporting interface signals.
  - The `application_error_reporting_handler` method must be invoked after an inbound TLP is completely received at the application interface.
  - In SNPS *env*, the `process_tlp_requests` subroutine is invoked during the task method.
  - The `process_tlp_requests` subroutine is available only with AXI Interface. In case of native interface, you must call the `application_error_reporting_handler` subroutine (similar to `process_tlp_request`), because `process_tlp_request` will be bypassed due to the `SVT_PCIE_TEST_SUITE_EXCLUDE_APP_INTERFACE` subroutine.

7.29 Root Port Tests

For the following Root Port cases, you need an application BFM enhancement to validate the specific feature.

Error Interrupt (INTx or MSI)

- When the RC internally generates an error or when an error message is received by the RC, then generated interrupt information propagates up to the application.
- You can get SII Interrupt signals (SNPS IIP-specific—that is, cfg_aer_rc_err_int and cfg_aer_rc_err_msi) information as described in section 7.29.1.
- If you do not want to read or verify SII Interrupt signals, then you must disable the test suite configuration variable from test case.
- The tl_root_error_message_controls_with_system_notification test validates this feature.

Native PME Software Model

- PCI Express-aware software can enable a mode where the Root Complex signals PME via an interrupt. When configured for native PME support, a Root Port receives the PME Message and sets the PME Status bit in its Root Status register. If software has set the PME Interrupt Enable bit in the Root Control register to 1b, the Root Port then generates an interrupt.

- You can get SII Interrupt signals (SNPS IIP-specific—that is, cfg_pme_int and cfg_pme_msi) information as described in section 7.29.1.
- If you do not want to verify Interrupt generation information at application interface, then you must disable the test suite configuration variable from test case.
- The tl_root_native_pme_software_interrupt_enable test validates this feature.

Variation A or Variation B OBFF Message

- According to the PCI Express Base Specification, the controller must drop a Variation A OBFF message in this scenario. Therefore, Application must not send Variation A OBFF messages when the controller is in the L0s or L1 TX states. Application must monitor the Message type and LTSSM state if Message Type is Variation A OBFF message and Controller LTSSM state is L0s or L1 TX state then Application drop the message. For the purpose of controller optimization, the controller does not perform this check (It is specific to SNPS IIP).
- You can get the information regarding application BFM enhancement as describe in section 7.29.2.
- If the case controller supports this feature, then you can skip invoking the method application_monitor_tx_obff_msg (svt_pcie_tlp tlp_xact, output bit drop) mentioned in section 7.29.2.
- The tl_root_tx_obff_msg test validates this feature.

7.29.1 Configuring the env to Use SII Interrupt Signals

- Changes in DUT-specific interface file (*rtl_dut_specific_if.svi*).

  You must declare the following DUT-specific interface signals:

```
// SII Interrupt signals
logic [7:0] cfg_aer_rc_err_int;
logic [7:0] cfg_aer_rc_err_msi;
logic [7:0] cfg_pme_int;
logic [7:0] cfg_pme_msi;
```

- Changes in DUT Wrapper file (pcie_dut_sv_wrapper.sv)
  - DUT-specific signals:
    - `bit [NF-1:0]   dut_cfg_aer_rc_err_int  ;`
    - `bit [NF-1:0]   dut_cfg_aer_rc_err_msi  ;`
    - `bit [NF-1:0]   dut_cfg_pme_int  ;`
    - `bit [NF-1:0]   dut_cfg_pme_msi  ;`
  - Connect all SII Interrupt signals to DUT-specific interface:

```
assign dut_specific_if.cfg_aer_rc_err_int     = dut_cfg_aer_rc_err_int;
assign dut_specific_if.cfg_aer_rc_err_msi    = dut_cfg_aer_rc_err_msi;
assign dut_specific_if.cfg_pme_int               = dut_cfg_pme_int;
assign dut_specific_if.cfg_pme_msi             = dut_cfg_pme_msi;
```

  - Connection with DUT:

```
`ifndef CX_IS_SW
  .cfg_aer_rc_err_int          (dut_cfg_aer_rc_err_int),
  .cfg_aer_rc_err_msi          (dut_cfg_aer_rc_err_msi),
  .cfg_pme_int                 (dut_cfg_pme_int),
  .cfg_pme_msi                 (dut_cfg_pme_msi),
```

```
`endif // CX_IS_SW
```

- Changes in Application BFM file (pcie_rc_dut_external_app_bfm.sv)
  - You must invoke the sii_interrupt_signals_handler() method for all supported device functions to get the information if the device function generated interrupts are using INTx or MSI/MSIx mechanism.
  - The sii_interrupt_signals_handler() method must be invoked during run_phase() and only if the test suite configuration variable get_sii_intr_signals_update is set.

7.29.2 Configuring the Application BFM to drop Variation A OBFF Msg

- Changes in Application BFM file (*pcie_rc_dut_external_app_bfm.sv*)
  - You must invoke the application_monitor_tx_obff_msg (svt_pcie_tlp tlp_xact, output bit drop) method to drop the Variation A OBFF when Controller is in L0s or L1 TX states so that message will not be received by the Controller.
  - The application_monitor_tx_obff_msg (svt_pcie_tlp tlp_xact, output bit drop) method must be invoked when application BFM gets the incoming transaction from sequencer and before driving the AXI slave interface of the core.
  - In SNPS env, the application_monitor_tx_obff_msg (svt_pcie_tlp tlp_xact, output bit drop) method is invoked during the do_tlp_xact (svt_pcie_tlp xact) subroutine.
  - The do_tlp_xact (svt_pcie_tlp xact) subroutine is available only with AXI Interface. In case of native interface, you must call the application_monitor_tx_obff_msg (svt_pcie_tlp tlp_xact, output bit drop) subroutine (similar to do_tlp_xact(svt_pcie_tlp xact)), because do_tlp_xact (svt_pcie_tlp xact) will be bypassed due to SVT_PCIE_TEST_SUITE_EXCLUDE_APP_INTERFACE.

7.30 Precision Time Measurement (PTM)

The PCIe test suite has been enhanced with new test cases to support PTM protocol verification. The newly added test cases are applicable only when PTM feature is supported by controller DUT (EP/RC). The added functionality tracks the timestamps for sending and receiving PTM messages which are used for sending PTM `ResponseD` (for RC VIP) and for calculating PTM context (for EP VIP) using the DL and PL callbacks.

Note The naming convention followed for the test cases is *tl_ptm**.

- Extend a sequence class with `svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence` as the base sequence and override the base class with derived class. This is required if you want to override/implement some tasks (which are DUT implementation specific) of the base class. Refer to the following steps to determine if overriding is required. For more details, see comments provided in this base class.
- Set the following DUT capability attributes before running any test case:

1. `enable_ptm_current_t2_t3_timestamp_read`: This bit indicates if Software or test can read the current t2/t3 timestamps noted by PTM Responder (RC DUT). Default value is 1.
2. `enable_ptm_current_t2_t3_timestamp_read_through_backdoor_pcie_cfg_rd`: Software or test can read the current t2/t3 timestamps noted by PTM Responder (RC DUT) through backdoor

PCIe configuration space access. Enable this if RC DUT has some Vendor Specific Capability in its PCIe configuration space through which test can access the t2/t3 timestamps. Default value is 1.

Note If you set this bit to 0 (which implies that t2/t3 timestamp will be read from non-PCIe DUT specific interface), then you must override/implement task `ptm_current_t2_timestamp_read` and `ptm_current_t3_timestamp_read` in derived sequence class extended from `svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence`.

- Set the following TS sequence configuration attribute before running any test case:
  - `initial_setup_for_ptm_responder_required`: Specifies if some initial setup is required for PTM Responder (RC DUT) to start responding with PTM `ResponseD`. Default value is 0.

Note
- If you set this bit to 1, then you must implement task initial_setup_for_ptm_responder in derived class extended from svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence.
- SNPS-IIP users may need not to override the task implementation. They can use the implementation in base class if it works fine for their DUT.

- If DUT has some Vendor Specific Enhanced Capability (VSEC) for PTM, then For VSEC, extend the `svt_pcie_test_suite_configuration_class` and re-implement function `get_vendor_specific_reg_off_or_field_indices`. This function is used to get the values of all register offsets or field indices for PTM VSEC. Using this function, you can specify different register offsets/field indices values for
  - Different links
  - DUT modes
  - Functions within the DUT (PF/VF number)

  Set the values for following attributes in the function:

  - `vsec_ptm_base_addr`
  - `vsec_ptm_t2_timestamp_read_lower_32_reg_off`
  - `vsec_ptm_t2_timestamp_read_upper_32_reg_off`
  - `vsec_ptm_t3_timestamp_read_lower_32_reg_off`
  - `vsec_ptm_t3_timestamp_read_upper_32_reg_off`

Note It is recommended to avoid using the replaceable defines for PTM VSEC which were used in previous releases and you must use this function only. Currently, these defines are retained for backward compatibility but may be removed from the model in the upcoming releases.

Note The *svt_pcie_test_suite_configuration.sv* file includes the information about each attribute.

- Additional optional controls have been added to provide users with better test controllability:

Table 7-8 `svt_pcie_test_suite_sequence_configuration` **Attributes**

Attributes	Description
`max_time_for_sending_ptm_rsp_ns`	Specifies the maximum time in which PTM Responder should send a PTM Response/`ResponseD` after receiving a PTM Request Message. Default value is 10000 (10us) as per the specification.
`insert_ptm_retry_hold_ack_or_send_nak`	This controls the mechanism for replay from VIP or DUT for following tests (`tl_ptm_update_*`). There are two mechanisms for replay: • Holding the ACK and thus replay happens after `replay_timeout`. • Sending NAK. Default value is -1 which randomizes between any of the two mechanisms.
`num_iterations_for_background_traffic`	You can control the number of iterations for background traffic. This is used in API `generate_background_random_tlp_traffic()` which is used in tests `tl_ptm_dialogs-rc`. Default value is 200.
`time_to_wait_between_background_tlps_in_ns`	You can also control how busy link should be and how frequently background TLP packets should be coming on the link. This is used in API `generate_background_random_tlp_traffic()` which is used in `tl_ptm_dialogs-rc`. Default value is 10ns.

Note

- For running PTM test cases, time precision of user testbench should be smaller than or equal to 1ns value (for example, 100ps, 1ps, and so on).
- For running PTM test cases, you must program your DUT's TX/RX latencies (if required) correctly for each speed/linkwidth. These latencies are time gap between a packet transmission from device's core to PCIe device's physical boundaries. These latencies are used in determining exact TX/RX timestamps for the DUT. If such latencies exists for the DUT, then you must take care to program these in your environment.

7.31 Gen5 Support

The key Gen5 features supported by the PCIe test suite are as follows:

Feature	Description
`SANITY`	Basic Gen5 testing
`EQ`	32GT Equalization
`EQ_HIGHEST_RATE`	Equalization with highest rate selection feature.
`EQ_VIA_LOOPBACK`	Equalization transition from Loopback LTSSM state.
`FRAMING`	Gen5 Framing error tests
`LOOPBACK`	Gen5 Loopback tests
`MODIFIED_TS`	Gen5 modified test suite exchange tests

Feature	Description
DATA_PARITY_ERR	Gen5 Data Parity error tests
PRECODING	Gen5 Precoding tests
RECOVERY	Gen5 Recovery LTSSM transition tests
REDO_EQ	Gen5 Redo Equalization
REGISTER_CHK	Register Checking test
RX_MARGIN	Rx Margining test
RETIMER	Re timer tests

7.31.1 Mode of Qualification

- Low Pin Count (LPC)
- Serial
- PIPE 4.4.1

7.31.2 Supported Tests

All the test suite tests supported with Gen5 capable device.

- Gen5/Gen4/Gen3/Gen2/Gen1/common tests support with RC DUT
- Retimer support

Complete list of all supported tests are available at the following location:

$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/examples/sverilog/tb_dut_pcie/tests

7.31.3 Gen5 Sequence APIs

To facilitate a new sequence development, APIs have been added into the PCIe test suite product.

The APIs and their documentation is available at:

$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/examples/sverilog/tb_dut_pcie/seq_and_cb/svt_pcie_ts_device_system_virtual_gen5_sequence_collection.sv

While implemented as part of `svt_pcie_ts_device_system_virtual_gen5_sequence_collection`, all the APIs are not limited for Gen5 operations, some of these are generic and can be used for non-Gen5 specific stimulus.

Example 7-5 API Code Sample

```
/* Method to execute speed change from VIP. Should be called in L0.
   * Method will return the control back when as soon as Recovery.Lock is entered from
L0.
   * @param target_rate Specifies the desired rate to speed to.
   */
  extern virtual task execute_speed_change(svt_pcie_pl_status::link_speed_enum
target_rate);

  /* Method to initiate Loopback.Entry from Config.LW.Start
```

```
   * @param eq_via_loopback Indicates whether Equalization state will be entered via
Loopback.Entry
   * @param lut Indicates the 'Lane Under Test'. Only applicable if eq_via_loopback is 1
   * @param modified_compliance_in_loopback Indicates whether loopback master will
request slave to send modified compliance pattern. Only applicable if eq_via_loopback is
1.
   * @param device_type indicates which device will be loopback Master(VIP or VIP
instance acting as DUT). VIP instance acting as DUT can be made Loopback Master only in
back-to-back sims and for PHY DUT.
   */
  extern virtual task initiate_loopback_via_config(bit eq_via_loopback=0, int lut=0, bit
modified_compliance_in_loopback=0, device_type_enum device_type=VIP);

  /* Method to initiate Loopback.Entry from Recovery Idle state
   * @param device_type indicates which device will be loopback Master(VIP or VIP
instance acting as DUT). VIP instance acting as DUT can be made Loopback Master only in
back-to-back sims and for PHY DUT.
   */
  extern virtual task initiate_loopback_via_recovery(device_type_enum device_type=VIP);

  /* Method to initiate Loopback exit.
   * @param device_type indicates Loopback Master for which loopback exit to be
initiated. legal values are:
   *        - VIP (Initiate Loopback Exit for VIP acting as loopback Master)
   *        - DUT (Initiate Loopback Exit for VIP(DUT) acting as loopback Master).
Applicable in b2b sims and for PHY DUT.
   */
  extern virtual task initiate_loopback_exit(device_type_enum device_type=VIP);

  /** Method to enable Equalization setup with Gen5 Eq_via_Loopback cases. If this API
is called, then equalization sequences run in parallel to eq_via_loopback test sequences
which make preset/coefficient requests in Phase2(from RC VIP) and Phase3(from EP VIP).*/
  extern virtual task enable_eq_setup_for_eq_via_loopback();

  /* Method to check the DUT supports for Modified TS */
  extern virtual task check_dut_support_modified_ts();

  /* Method to request Precoding from VIP via 128b/130b EQ TS2 at 16GT/s. Should be
called in L0 or at time0.*/
  extern virtual task request_precoding_at_16gts();

  /* Method to configure VIP to advertise 'No EQ Required'.*/
  extern virtual task configure_vip_for_no_eq_required();
```

To add a new Gen5 test using APIs, create a new sequence extending from `svt_pcie_ts_gen5_base_sequence` and call the appropriate APIs from the body method of the sequence. Register this sequence as the `default_sequence` in the test.

Example 7-6 Sequence Implemented Using APIs

```
task
svt_pcie_ts_device_system_virtual_gen5_pl_clwa_no_8ts2_w_elbc_11b_in_polling_config_sequ
ence::body();
  string method_name = "body";
  registered_seq =
"svt_pcie_ts_device_system_virtual_gen5_pl_clwa_no_8ts2_w_elbc_11b_in_polling_config_seq
uence";
```

```
  begin
    super.body();
    begin
      /* Wait until DUT LTSSM is in POLLING_CONFIGURATION */

wait_for_state_at_desired_speed(1,svt_pcie_types::POLLING_CONFIGURATION,svt_pcie_pl_stat
us::SPEED_2_5G);

      //-------------------------------------------------------
      // Transmit the error injection in TS2 from VIP such that 8-TS2s ELBC != 11b on
one lane and 8-TS2 with ELBC == 11b  on rest of the lane
      //-------------------------------------------------------
      //Set mask for active lanes to place user defined TS OS on them
      lane_mask = 32'h0;

      randomize_lut();
      lane_mask[lane_under_test] = 1'b1;

       // Set user defined TS2 as '0' set
      set_lane_mask_for_user_ts(1'b0, lane_mask);

      `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: Starting User TS2
stream...")

      send_user_ts(1'b0,lane_under_test,8,0,,9'h1F7,9'h1F7,,,8'h00); //  8-user TS on
all Lane except one Lane
      `svt_pcie_ts_note(method_name,$sformatf("Starting User TS2 stream on
Lane=%0d,Transmit the error injection in TS2 from VIP such that 8-TS2s ELBC != 11b on
one lane and 8-TS2 with ELBC == 11b  on rest of the lane",lane_under_test));

      check_dut_support_modified_ts();
      if(dut_support_modified_ts_flag==0)
        `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: DUT did not support
Modified TS1/2, Exiting the test...")
      else
      begin
        `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: BEFORE_WAIT :: Waiting
for DUT to CONFIGURATION_LINKWIDTH_START")
       wait (dut_status.pcie_status.pl_status.ltssm_state ==
svt_pcie_types::CONFIGURATION_LINKWIDTH_START);
        `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: AFTER_WAIT :: DUT
entered CONFIGURATION_LINKWIDTH_START, disabling User TS2 stream now")

        // Disable user defined TS1/2 when DUT enters CONFIGURATION_LINKWIDTH_START
        stop_user_ts(1'b0,lane_under_test,0);
```

8 Using the Test Suite: PHY DUT

This chapter discusses the following topics:

8.1 L1SS With Sideband Signals

Requirements for `pl_l1_pm_substates_with_pipe_sideband_signals-phy` test:

- Test is only applicable for DUT which supports L1SS sideband signals present inside `svt_pcie_snps_pipe_if` interface and PL configuration `svt_pcie_pl_configuration::enable_l1ss_pipe_handshake` is set.
- Test uses *run_time_cfg_overrides_l1ss_phy.txt* under *tb_dut_pcie/tests* directory which will be used as command-line plusarg using `svt_pcie_test_suite_all_link_prop_file`. You can create your own .txt file with different configuration values and run this test with the plusarg option as shown below:

```
+svt_pcie_test_suite_all_link_prop_file=tests/run_time_cfg_overrides_l1ss_no_sideband_signals.txt
```

8.2 Enabling EIEOS Pattern for Gen4

Added support for new 16G EIEOS pattern:

- To enable 16G EIEOS pattern for Gen4, set the `SVT_PCIE_TEST_SUITE_ENABLE_EIEOS_16G_0000_FFFF` macro to 1. This will invoke VIP to transmit and expect new 16G EIEOS pattern. By default, it is set to 0.

```
`define  SVT_PCIE_TEST_SUITE_ENABLE_EIEOS_16G_0000_FFFF 1
```

8.3 Link Width Dependent Tests

Table 8-1 shows the Link Width dependent tests.

Table 8-1 Link Width Dependent Tests

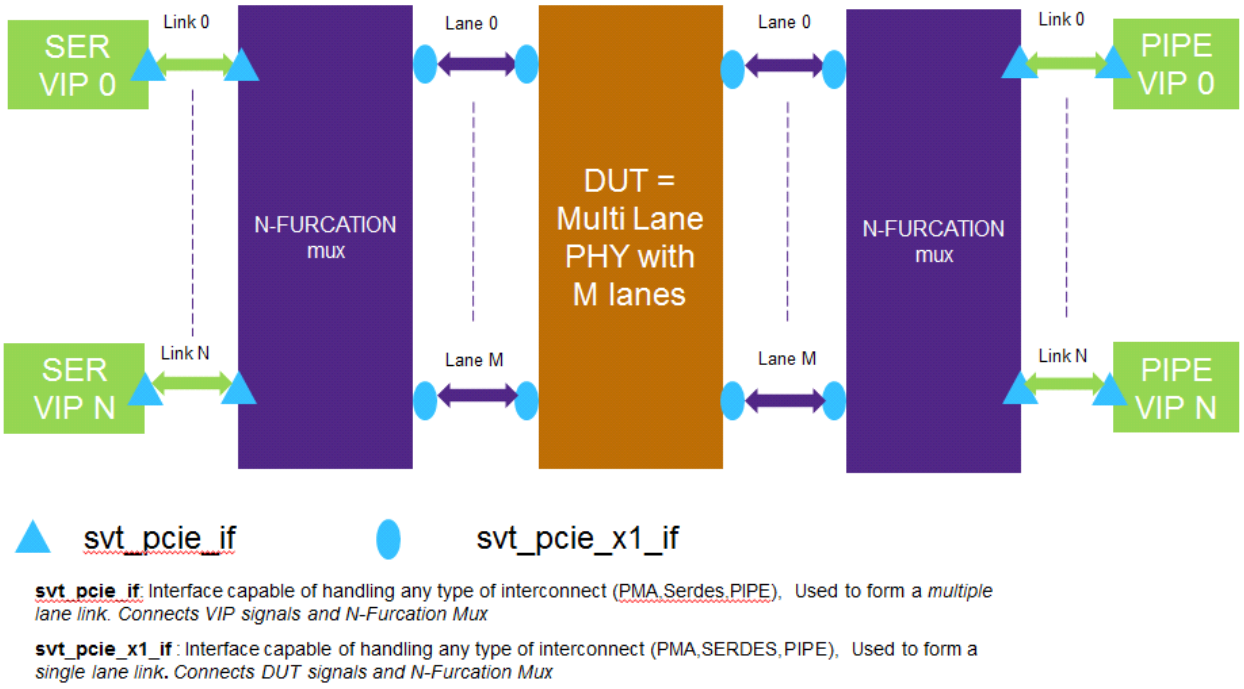
S.No.	Test Name	Not applicable for Link Width
1	gen3_pl_recovery_equalization_phase0_to_phase1_case1_error	1
1	gen3_pl_recovery_equalization_phase0_to_phase1_case1_error	1
2	gen3_pl_recovery_equalization_phase0_to_recovery_speed_timeout_case1_error	1
3	gen3_pl_recovery_equalization_phase1_to_rcvrlock_case1_error	1
4	gen3_pl_recovery_equalization_lw_down_error	1
5	gen3_pl_recovery_equalization_phase3_to_rcvrlock_case1_error	1
6	gen4_pl_recovery_equalization_lw_down_error	1
7	gen4_pl_recovery_equalization_phase0_to_recovery_speed_timeout_case1_error	1
8	pl_polling_active_to_polling_config_compliance_bit_0_timeout_error	1
9	pl_polling_active_to_polling_config_loopback_bit_1_timeout_error	1
10	pl_polling_active_to_polling_config_ts2_timeout_error	1
11	pl_polling_active_to_polling_compliance_electrical_idle_timeout_error	1
12	gen3_pl_multi_sdp_in_symbol_time_error	
13	pl_lane_reenable_with_reset_assertion	1
14	pl_lane_turn_off_and_reenable	1
15	pl_random_data_on_turned_off_lanes	1
16	gen3_pl_idle_followed_by_no_idle_next_lane_error	1

8.4 Multi-Link Topology/N-Furcation MUX Architecture

8.4.1 Multi-Lane PHY as DUT in Unified Test Environment (for PHY-DUT only)

This section describes the general architecture of the PHY DUT testbench. Figure 8-1 provides the top-level diagram of usage model for PHY DUT verification.

Figure 8-1 Multi-Lane PHY as DUT



The RC, EP, and N-Furcation Mux VIP module instances are part of the top module scope in the environment.

The maximum number of physical lanes used by each VIP instance is configurable through compile time parameter. The actual link width of PCIe link is configurable using run time settings.

Based on PHY design requirements it is possible to use the N-Furcation Mux and share PHY DUT lanes across multiple VIP instances.

For example, with 16 lanes at PIPE/SERDES interface, the following topologies can be supported.

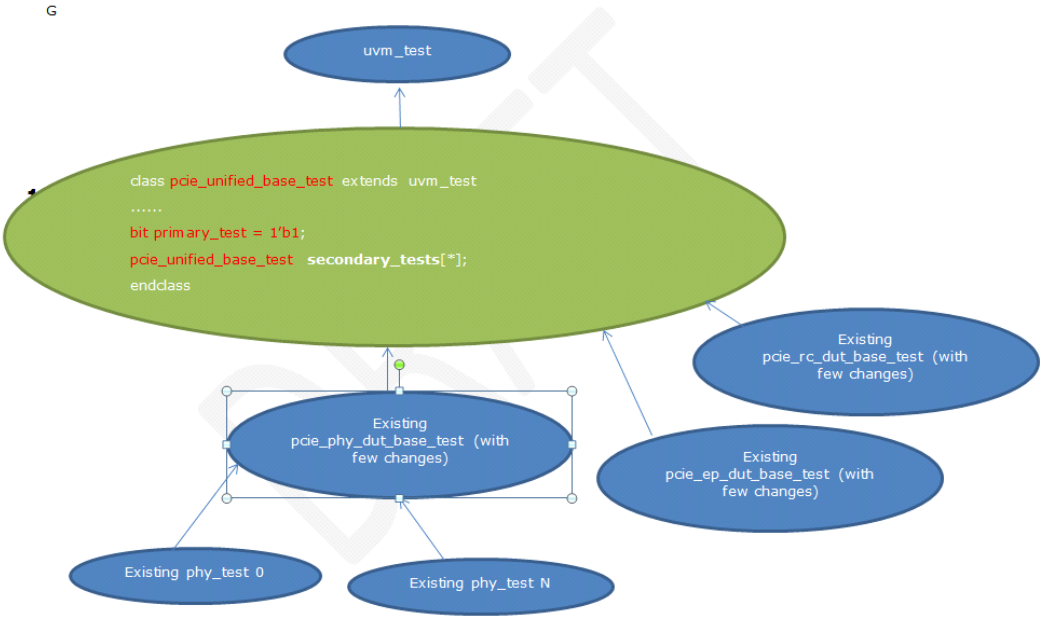
- 1 link with 16 lanes (x16)
- 2 links with 8 lanes (x8, x8)
- 4 links with 4 lanes (x4, x4, x4, x4)
- 8 links with 2 lanes (x2, x2…)

8.4.2 Reuse of Existing Single Link Tests in Multi-Link Topologies

The UVM Methodology permits use of a single top-level test class instance whose type is determined using the `UVM_TESTNAME` runtime switch. As each existing single link test is self-sufficient to operate independently on a single link (collection of environment instance, config object instance and some error demotion logic), the requirement to reuse the existing test in multi-link topology could be best addressed by creating separate or multiple instances of these test classes.

To meet the technical requirement stated above without violating the constraints laid down by `UVM_METHODOLOGY`, a new class `pcie_unified_base_test` has been introduced and test component arrangement is shown in Figure 8-2.

Figure 8-2 tb_dut_pcie Class Hierarchy



When `run_test` is invoked by initial block in *top.sv*, the first (primary) instance of `uvm_test` component is created (type is determined by `UVM_TESTNAME`), during the build phase execution this test instance does the following:

- Executes factory override (`set_type_override_by_name`) for `pcie_unified_base_test` with type requested by `UVM_TESTNAME`.
- Retrieves hex plusarg `svt_pcie_disable_link_creation`.See below for more details
- Retrieves compile-time topology from `uvm_config_db` (that is, `highest_link_num`, handles to `svt_pcie_unified` interfaces).
- If required (based on information from 2 and 3), it invokes create calls for one or more instances of `pcie_unified_test` and populates the array `secondary_tests[*]` with handles of these newly created child (secondary) test instances.
- Flips the value of `primary_test` flag in any of the newly created secondary test instances (this prevents recursive creation of test instances).
- Executes the legacy flow of setting up the configuration object and the environment instance.

Note By default, any code that is part of test class is going to be independently executed for each of the test instances, if the code must be executed only once then it should be guarded with if (primary_test) begin .... end.

8.4.3 Plusarg svt_pcie_disable_link_creation

This global plusarg is retained for backward compatibility reasons and equivalent functionality is provided by `svt_pcie_test_suite_configuration::disable_link`.

Need for the `svt_pcie_disable_link_creation` plusarg:

- The compile-time topology may consist of N links but this can be changed at runtime. This plusarg informs the `build_phase` of `uvm_test_top` which links from compile-time configuration are active.

The default setting of `uvm_test.disable_link_creation` variable is 32'hFFFF_FFFE which means create only one test instance and assign it to link 0 from compile-time topology.

- For operating with 2 links, this plusarg value must me 32'hFFFF_FFFC.
- For operating with 3 links, this plusarg value must me 32'hFFFF_FFF8. Not supported by example topology but supported by the base test.
- For operating with 32 links, this plusarg value must me 32'h0. No supported by example topology but supported by the base test.

8.4.4 Using svt_pcie_nfurcation_mux

8.4.4.1 Assumptions

- Connections to Physical Lane 0 of each VIP instance (active at runtime) will be used to connect shared signals of the VIP.
- If a certain DUT lane is not used due to runtime topology settings, then the signals corresponding to it will be left in unknown state (that is, x).

Table 8-2 Parameters Supported by N-furcation MUX

Parameter	Default Value	Description
SVT_PCIE_NMUX_MAX_NUM_VIP_IFS	2	Indicates the number of VIP instances that share the DUT interface. Valid values: 1–31
SVT_PCIE_NMUX_MAX_NUM_X1_LANES	4	Indicates the maximum number of lanes that the shared DUT supports.
SVT_PCIE_NMUX_UVM_CFG_DB_RELATIVE_SCOPE	uvm_test_top	String indicating which `uvm_component (s)` can access the API class instance through `uvm_config_db` calls.
SVT_PCIE_NMUX_UVM_CFG_DB_VAR_NAME	numx_api	String indicating what value should be used by `uvm_component (s)` to access the API through `uvm_config_db` calls.

Table 8-3 Signals Used by N-furcation MUX

Signals	Description
vip_ifs[SVT_PCIE_NMUX_MAX_NUM_VIP_IFS]	An array of interface of type `svt_pcie_if` to connect to VIP instances.
vip_x1_ifs[SVT_PCIE_NMUX_MAX_NUM_X1_LANES]	An array of interface of type `svt_pcie_x1_if` to connect to DUT lanes.
reset_to_unused_vip	Output signal with width = `SVT_PCIE_NMUX_MAX_NUM_VIP_IFS`. Each bit corresponds to a VIP instance. The logic to generate reset to the VIP instance should use this. It ensures if any of the VIP instances are unused due to runtime configuration, then you must hold it in the reset state.

8.4.5 Interfacing the N-Furcation MUX Module to Class World

Each `svt_pcie_nfurcation_mux` instance is connected to `uvm_test` instance through the following class pair:

- Virtual class `svt_pcie_nfurcation_mux_api`
- Concrete class `svt_pcie_nfurcation_mux_api_impl` extends `svt_pcie_nfurcation_mux_api` implemented within module `svt_pcie_nfurcation_mux`.

An initial begin-end block within `svt_pcie_nfurcation_mux` creates an instance of `svt_pcie_nfurcation_mux_api_impl` and deposits its handle in `uvm_config_db`.

`pcie_unified_base_test` instance (only with primary_test == 1) retrieves this handle during `connect_phase` and using the methods provided by the above class pair configures the MUX with required runtime configuration.

The method used from the above class pair:

- `function int set_curr_link_width (int vip_number, int link_width, int max_link_width);`
- `function int apply_link_widths();`
- `function bit get_on_off_status_for_vip_if (int vip_number);`

Expected use model using the above listed methods:

- Feed runtime topology to MUX instance using `set_curr_link_width` calls:

```
set_curr_link_width(rc_0_if_index, rc_0_run_time_width, rc_0_compile_time_width);
set_curr_link_width(ep_0_if_index, ep_0_run_time_width, ep_0_compile_time_width);
```

- Take action of information fed by preceding `set_curr_link_width` calls `apply_link_widths();`
- Retrieve status of individual VIP interfaces to determine what should be done with `uvm_components` corresponding to these VIP during remaining phases (on each VIP instance basis):

```
rc_0_off = get_on_off_status_for_vip_if(rc_0_if_index);
ep_0_off = get_on_off_status_for_vip_if(ep_0_if_index);
```

8.5 Test Cases for RX Margining

RX Margining test cases for local PHY are only applicable with Gen4 speed (16GT/s) and PIPE Version 4.4.1.

The portion of sequences involving read/read_completion have been minimally verified.

Table 8-4 shows the attributes added in class `svt_pcie_dut_capabilities` for all optional features.

Table 8-4 Attributes in svt_pcie_dut_capabilities

Attribute	Applicable DUT Type	Description
rx_margining_voltage_supported	EP/RC/PHY	This attribute needs to be set for verification of voltage margining related test cases. Without this attribute, the voltage margining tests will fail with the following warning: `OptionalFeatureNotSupportedByDUT` For this scenario, all such type of cases should be excluded.
rx_margining_independent_error_sampler_supported	PHY	This attribute indicates whether error sampler is supported or not. A value of 1 indicates that this feature is supported. Based on this, the error count updating is checked. With value 0, test will continue without checking error count. Without this attribute, error count update tests will fail with the following warning: `OptionalFeatureNotSupportedByDUT` For this scenario, all such type of cases should be excluded.
rx_margining_sample_reporting_method_supported	PHY	This attribute indicates whether a sample frequency or a sample count is reported. It is encoded as follows: • 0 – Sample Count Reported When set to 0, tests will monitor the 'Sample Count' field and expect PHY to update this field whenever applicable. • 1 – Sample Frequency Reported Without this attribute, error count update tests will fail with the following warning: `OptionalFeatureNotSupportedByDUT` For this scenario, all such type of cases should be excluded.
mbi_read_completion_supported	PHY	This Indicates whether PHY will respond back with a `read_completion` for a read command it received from MAC. Without this attribute, tests involving read and read completion will fail with the following warning: `OptionalFeatureNotSupportedByDUT` For this scenario, all such type of cases should be excluded.

Table 8-4 Attributes in svt_pcie_dut_capabilities

Attribute	Applicable DUT Type	Description
phy_clears_margin_status_post_margining	PHY	Specifies whether PHY will send a write command to clear the Margin Status bit after MAC has stopped margining process. • A value of 1 indicates that PHY DUT will send a write to clear Margin Status bit. • A value of 0 indicates that PHY DUT will not send a write to clear Margin Status bit. Without this attribute, stop margining will print the following string: `OptionalFeatureNotSupportedByDUT`
sample_count_update_after_reaching_max_value_supported	PHY	Specifies the behavior of PHY DUT after sample count has been saturated during margining. • Write transaction from PHY DUT updates MAC status 1 register with value 7F for `num_samples_margined`. • No more write transaction from PHY DUT to update MAC Margin Status 1 register until `phy_dut_max_time_to_update_margin_status_registers_timer_ns` time has elapsed.

Table 8-5 shows the attributes added in class `svt_pcie_test_suite_configuration`:

Table 8-5 Attributes in svt_pcie_test_suite_configuration

Attribute	Applicable DUT Type	Description
rx_margining_max_timing_offset_supported	PHY	This attribute sets the maximum offset that can be set by the command to set the offset value in the Margin Control 1 register. The timing offset value is randomized between a value of 20 to maximum supported.
rx_margining_max_voltage_offset_supported	PHY	If voltage margining is supported, this attribute sets the maximum offset that can be set by the command to set the offset value in the Margin Control 1 register. The voltage offset value is randomized between a value of 5 to maximum supported.

num_samples_margined	EP/RC/PHY	This attribute sets the number of writes onto the Margin Status 1 register while margining is in process before moving to the next step. Example:
		`[svt_pcie_ts_device_system_virtual_local_phy_step_margin_sequence] monitoring for num_samples_margined samples for errors during margining`
		`[svt_pcie_ts_device_system_virtual_local_phy_step_margin_sequence] Sample count is 23`

Table 8-5 Attributes in svt_pcie_test_suite_configuration

Attribute	Applicable DUT Type	Description
phy_dut_max_time_to_update_margin_status_registers_timer_ns	PHY	It is used to set the maximum amount of time with in which a PHY DUT is expected to send a MBI command to update VIP MAC Margin Status registers. Example: `[svt_pcie_ts_device_system_virtual_initiate_step_margin_in_mbi_sequence] Starting timer timer_phy_dut_max_time_to_update_margin_status to track Margin Status update.`
max_mbi_response_timer_ns	PHY	This attribute sets the maximum amount of time with in which a PHY DUT is expected to respond to an MBI command issued by VIP MAC. Example: • If response obtained: `[svt_pcie_ts_device_system_virtual_initiate_step_margin_in_mbi_sequence] write_ack for margin start received.` • If response is not obtained: `[svt_pcie_ts_device_system_virtual_initiate_step_margin_in_mbi_sequence] Write Ack not received for start margin`
rx_margining_max_lanes_supported	PHY	It is used to set maximum lanes that support RX Margining.
mbi_read_completion_timer	PHY	It is used to set the time within which PHY is expected to send Read Completion after receiving a Read command.

Table 8-6 shows the attributes added in class `svt_pcie_test_suite_sequence_configuration`:

Table 8-6 Attributes in svt_pcie_test_suite_sequence_configuration

Attribute	Applicable DUT Type	Description
rx_message_bus_write_buffer_depth	EP/RC/PHY	Used to set the maximum RX Write Buffer Depth per committed command.
mbi_write_uncommitted_supported	PHY	Used to control selecting from committed writes only or uncommitted write followed by committed write for RX Margining commands. • 0 – Indicates committed writes only. • 1 – Indicates uncommitted writes followed by committed writes.

Table 8-6 Attributes in svt_pcie_test_suite_sequence_configuration

Attribute	Applicable DUT Type	Description
rx_margining_interval_step_command_ns	PHY	Used to control the configurable time before issuing each command to update the step for test gen4_pl_rx_margin_local_phy_step_margin_to_timing_incrementing_offset-phy.sv.
wait_sample_count_before_clearing	PHY	Used to control the minimum sample count which should be reached during RX Margining before clearing Margin Status register. It is dependent on frequency update of sample count by DUT.
rx_margining_timing_steps	PHY	Used to control total number of timing steps. The allowed range is 8–63. The steps value will be randomized between 1 and rx_margining_timing_steps for each applicable timing margin request.
rx_margining_voltage_steps	PHY	Used to control total number of voltage steps. The allowed range is 32–127. The steps value will be randomized between 1 and rx_margining_voltage_steps for each applicable voltage margin request.
rx_margining_direction_or_type_change_test_flow_decide	PHY	Used to control the flow of RX Margining tests that require a change in direction or type of margining. • 2'b10 or 2'b01 indicates randomizing the flow. • 2'b00 indicates that RX Margining will be stopped and then restarted as per intention of the test. • 2'b00 indicates that RX Margining will be stopped and then restarted as per intention of the test.
rx_margining_direction_control	PHY	Used to control whether a particular direction for Margining will be selected or the direction will be randomized. • 1 – specifies direction based scenario • 0 – specifies direction will be randomized
rx_margining_margin_direction	PHY	Only valid if `rx_margining_direction_control` is set to 1. • 1 – Direction for margining will be Right for timing margin and Down for voltage margin tests. • 0 – Direction for margining will be Left for timing margin and Up for voltage margin tests.
rx_margining_num_timing_step	PHY	Indicates step change in RX Margining while Margining. Allowable range: 0 – RxMarginingTiming Step (8-63 as per spec).

Table 8-6 Attributes in svt_pcie_test_suite_sequence_configuration

Attribute	Applicable DUT Type	Description
rx_margining_max_timing_offset_change	PHY	Controls value of RxMarginingMaxTimingOffsetChange. Allowable range: 1-127 (decimal)
rx_margining_max_voltage_offset_change	PHY	Controls value of RxMarginingMaxVoltageOffsetChange. Allowable range: 1-127 (decimal)
choose_mbi_lanes	PHY	Used to select lanes for MBI. • 2'b00 indicates lanes will be chosen randomly. • 2'b11 indicates user defined logical lanes between `min_mbi_lane_no` and `max_mbi_lane_no` will be chosen. Set seq_cfg `min_mbi_lane_no` and `max_mbi_lane_no` accordingly. • 2'b01 or 2'b10 indicates all logical lanes between 0 and ts_cfg `rx_margining_max_lanes_supported` will be chosen.
min_mbi_lane_no	PHY	Only applicable when `choose_mbi_lanes = 2'b11`. Indicates logical min lane chosen for margining.
max_mbi_lane_no	PHY	Only applicable when `choose_mbi_lanes = 2'b11`. Indicates logical max lane chosen for margining.

8.6 Long Running Test

The `pl_passive_compliance_test_elec_idle_timeout_error-phy` test is expected to run for long time because of its number of iterations from `Polling.Active` to `Polling.Compliance` after timeout and other DUT dependent timing parameters to achieve bit/symbol lock and transmit data.

The test is expected to fail for the first time because of default UVM timeout value. The following debug tips will be printed in the `final_phase` if the test fails because of the testbench timeout. This helps user to configure the specific UVM timeout value and also prints the message to reduce the simulation time depending on the configuration values.

Example 8-1 Log File Snippet

```
UVM_FATAL /global/apps/vcsmx_2018.09/etc/uvm/base/uvm_phase.svh(1272) @ 4800000000.00
ps: reporter [PH_TIMEOUT] Explicit timeout of 4800000000.00 ps hit, indicating a
probable testbench issue
---
UVM_WARNING ./tests/ts.pl_passive_compliance_test_elec_idle_timeout_error-phy.sv(212) @
4800000000.00 ps: uvm_test_top [final_phase] Test Failing Summary: Test could complete
only 18 settings of passive compliance load board testing within the stipulated UVM
timeout(4800000000.00 ps). Remaining settings to be covered = 16. See below for tips
UVM_INFO ./tests/ts.pl_passive_compliance_test_elec_idle_timeout_error-phy.sv(213) @
4800000000.00 ps: uvm_test_top [SNPS_PCIE_TS_DEBUG_TIP] Grep for 'Delta time to switch
from last setting' from the log to know the actual time consumed to move from one
setting to another. This will help to estimate the additional simulation time required
for this test to complete remaining settings. This might be speed dependent also.
```

```
UVM_INFO ./tests/ts.pl_passive_compliance_test_elec_idle_timeout_error-phy.sv(214) @
4800000000.00 ps: uvm_test_top [SNPS_PCIE_TS_DEBUG_TIP] Current Polling.Active
timeout(svt_pcie_pl_configuration::polling_active_timeout_ns) is set to 240000ns. Test
performs N number of timeouts in Polling.Active equal to number of compliance settings.
If this is consuming huge simulation time and faster simulation time is expected, then
you can scale down the Polling.Active timeout which is also sufficient to achieve
bit/symbol lock by PHY DUT. For ex. If scaled down by 10, simulation will be faster by
~7x
```

8.7 Gen5 Support

The key Gen5 features supported by the PCIe test suite are as follows:

Feature	Description
SANITY	Basic Gen5 testing
EQ	32GT Equalization
EQ_HIGHEST_RATE	Equalization with highest rate selection feature.
EQ_VIA_LOOPBACK	Equalization transition from Loopback LTSSM state.
FRAMING	Gen5 Framing error tests
LOOPBACK	Gen5 Loopback tests
MODIFIED_TS	Gen5 modified test suite exchange tests
DATA_PARITY_ERR	Gen5 Data Parity error tests
PRECODING	Gen5 Precoding tests
RECOVERY	Gen5 Recovery LTSSM transition tests
REDO_EQ	Gen5 Redo Equalization
REGISTER_CHK	Register Checking test
RX_MARGIN	Rx Margining test
RETIMER	Re timer tests

8.7.1 Mode of Qualification

- Low Pin Count (LPC)
- Serial
- PIPE 4.4.1

8.7.2 Supported Tests

All the test suite tests supported with Gen5 capable device.

- Gen5/Gen4/Gen3/Gen2/Gen1/common tests support with PHY DUT
- Retimer support

Complete list of all supported tests are available at the following location:

$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/examples/sverilog/tb_dut_pcie/tests

8.7.3 Gen5 Sequence APIs

To facilitate a new sequence development, APIs have been added into the PCIe test suite product.

The APIs and their documentation is available at:

$DESIGNWARE_HOME/vip/svt/pcie_test_suite_svt/latest/examples/sverilog/tb_dut_pcie/seq_and_cb/svt_pcie_ts_device_system_virtual_gen5_sequence_collection.sv

While implemented as part of `svt_pcie_ts_device_system_virtual_gen5_sequence_collection`, all the APIs are not limited for Gen5 operations, some of these are generic and can be used for non-Gen5 specific stimulus.

Example 8-2 API Code Sample

```
/* Method to execute speed change from VIP. Should be called in L0.
   * Method will return the control back when as soon as Recovery.Lock is entered from
L0.
   * @param target_rate Specifies the desired rate to speed to.
   */
  extern virtual task execute_speed_change(svt_pcie_pl_status::link_speed_enum
target_rate);

  /* Method to initiate Loopback.Entry from Config.LW.Start
   * @param eq_via_loopback Indicates whether Equalization state will be entered via
Loopback.Entry
   * @param lut Indicates the 'Lane Under Test'. Only applicable if eq_via_loopback is 1
   * @param modified_compliance_in_loopback Indicates whether loopback master will
request slave to send modified compliance pattern. Only applicable if eq_via_loopback is
1.
   * @param device_type indicates which device will be loopback Master(VIP or VIP
instance acting as DUT). VIP instance acting as DUT can be made Loopback Master only in
back-to-back sims and for PHY DUT.
   */
  extern virtual task initiate_loopback_via_config(bit eq_via_loopback=0, int lut=0, bit
modified_compliance_in_loopback=0, device_type_enum device_type=VIP);

  /* Method to initiate Loopback.Entry from Recovery Idle state
   * @param device_type indicates which device will be loopback Master(VIP or VIP
instance acting as DUT). VIP instance acting as DUT can be made Loopback Master only in
back-to-back sims and for PHY DUT.
   */
  extern virtual task initiate_loopback_via_recovery(device_type_enum device_type=VIP);

  /* Method to initiate Loopback exit.
   * @param device_type indicates Loopback Master for which loopback exit to be
initiated. legal values are:
   *        - VIP (Initiate Loopback Exit for VIP acting as loopback Master)
   *        - DUT (Initiate Loopback Exit for VIP(DUT) acting as loopback Master).
Applicable in b2b sims and for PHY DUT.
   */
  extern virtual task initiate_loopback_exit(device_type_enum device_type=VIP);

  /** Method to enable Equalization setup with Gen5 Eq_via_Loopback cases. If this API
is called, then equalization sequences run in parallel to eq_via_loopback test sequences
which make preset/coefficient requests in Phase2(from RC VIP) and Phase3(from EP VIP).*/
  extern virtual task enable_eq_setup_for_eq_via_loopback();
```

```
  /* Method to check the DUT supports for Modified TS */
  extern virtual task check_dut_support_modified_ts();

  /* Method to request Precoding from VIP via 128b/130b EQ TS2 at 16GT/s. Should be
called in L0 or at time0.*/
  extern virtual task request_precoding_at_16gts();

  /* Method to configure VIP to advertise 'No EQ Required'.*/
  extern virtual task configure_vip_for_no_eq_required();
```

To add a new Gen5 test using APIs, create a new sequence extending from `svt_pcie_ts_gen5_base_sequence` and call the appropriate APIs from the body method of the sequence. Register this sequence as the `default_sequence` in the test.

Example 8-3 Sequence Implemented Using APIs

```
task
svt_pcie_ts_device_system_virtual_gen5_pl_clwa_no_8ts2_w_elbc_11b_in_polling_config_sequ
ence::body();
  string method_name = "body";
  registered_seq =
"svt_pcie_ts_device_system_virtual_gen5_pl_clwa_no_8ts2_w_elbc_11b_in_polling_config_seq
uence";
  begin
    super.body();
    begin
      /* Wait until DUT LTSSM is in POLLING_CONFIGURATION */

wait_for_state_at_desired_speed(1,svt_pcie_types::POLLING_CONFIGURATION,svt_pcie_pl_stat
us::SPEED_2_5G);

      //-------------------------------------------------------
      // Transmit the error injection in TS2 from VIP such that 8-TS2s ELBC != 11b on
one lane and 8-TS2 with ELBC == 11b  on rest of the lane
      //-------------------------------------------------------
      //Set mask for active lanes to place user defined TS OS on them
      lane_mask = 32'h0;

      randomize_lut();
      lane_mask[lane_under_test] = 1'b1;

       // Set user defined TS2 as '0' set
      set_lane_mask_for_user_ts(1'b0, lane_mask);

      `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: Starting User TS2
stream...")

      send_user_ts(1'b0,lane_under_test,8,0,,9'h1F7,9'h1F7,,,8'h00); //  8-user TS on
all Lane except one Lane
      `svt_pcie_ts_note(method_name,$sformatf("Starting User TS2 stream on
Lane=%0d,Transmit the error injection in TS2 from VIP such that 8-TS2s ELBC != 11b on
one lane and 8-TS2 with ELBC == 11b  on rest of the lane",lane_under_test));

      check_dut_support_modified_ts();
      if(dut_support_modified_ts_flag==0)
```

```
        `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: DUT did not support
Modified TS1/2, Exiting the test...")
      else
      begin
        `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: BEFORE_WAIT :: Waiting
for DUT to CONFIGURATION_LINKWIDTH_START")
        wait (dut_status.pcie_status.pl_status.ltssm_state ==
svt_pcie_types::CONFIGURATION_LINKWIDTH_START);
        `svt_pcie_ts_note(method_name,"SNPS_PCIE_TS_TEST_STEP :: AFTER_WAIT :: DUT
entered CONFIGURATION_LINKWIDTH_START, disabling User TS2 stream now")

        // Disable user defined TS1/2 when DUT enters CONFIGURATION_LINKWIDTH_START
        stop_user_ts(1'b0,lane_under_test,0);
```

9 Multi-Link Controller DUT

This chapter discusses the following topics:

9.1 Assumptions

Following are the assumptions considered in this chapter:

- User has already setup single link environment.
- Helper macros are used to create multiple links.
- Configuration attributes to be set via *.txt*/*.scr* file instead of plusargs (for more details, see File Formats).
- Only plusarg supported for multi-link configuration is `+svt_pcie_disable_link_creation`.

  A reference *txt* file is added for deprecated plusargs.

  This is available at *tb_dut_pcie/tests/ reference_txt_file_for_deprecated_plusargs.txt.*
- Selection of EP and RC mode operation can only be done at compile time.

9.2 Overview

Multi-link support for EP/RC DUT controller is the extended use model of single link EP/RC usage. This section lists the steps for setting up multi-link environment for EP/RC DUT.

Multi-link environment can be created using the following two approaches:

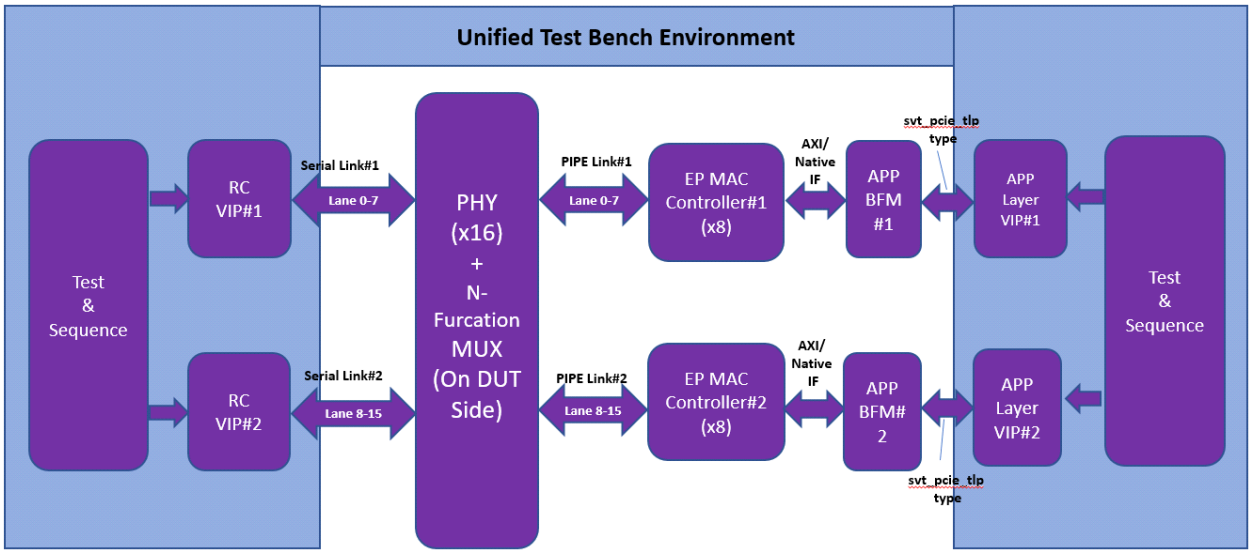
- Fixed Topology (Compile Time Selection of Link)

In this topology, number of links are programmed at compile time, such that it will be fixed and cannot be changed at runtime.

This document focuses on setting up multi-link environment for a fixed topology.

For example, Figure 9-1 shows compile-time topology of two links, which are formed with two `x8 EP controller` connected to serial side of two VIP instances (or PIPE interface, with VIP as SPIPE) such that link 1 uses lanes 0–7 of PHY whereas link 2 uses lanes 8–15 of PHY.

Figure 9-1 Compile-Time Approach of Two Links



- Configurable Topology (Runtime Approach)

  Synopsys PCIe Test Suite framework provides features to build runtime configurable multi-link topologies, where links can be programmed at runtime. With this feature, it is possible to compile the environment once and run tests with different configurations using runtime definition of configuration supplied by the files. If you want to build a configurable multi-link setup, contact Synopsys Support.

Note For an overview of testbench architecture and configuration files, see Test Suite Testbench Architecture.

9.3 Conversion of Single Link Environment to Muti-Link Environment

Integration steps are similar to the steps discussed in section Integrating Controller (EP/RC) DUT. Additionally, perform the following steps to setup a fixed configuration multi-link environment.

Following changes are required in the topology file.

- Instantiate the multiple instance of VIP based on the requirement.

Example below covers two link setup.

String "dm" in following code snippet stands for "Dual Mode" (Root or Endpoint).

- Instantiation of VIPs

// First link: This is the interface and VIP first instance on link 1, acting as RC/EP for EP/RC DUT

```
svt_pcie_if vip_if_link1(clkreq_n[0],wake_n[0]);
svt_pcie_single_port_device_agent_hdl dm_vip_as_vip_link1(vip_if_link1);
```

// First link: This is VIP instance, acting as Application VIP on first link sitting above the DUT TL layer

```
svt_pcie_if app_vip_if_link1(clkreq_n[0]);
svt_pcie_single_port_device_agent_hdl dm_vip_as_dut_link1(app_vip_if_link1);
```

// Second Link: This is the interface of VIP second instance on link2, acting as RC/EP for EP/RC DUT

```
svt_pcie_if vip_if_link2(clkreq_n[0],wake_n[0]);
svt_pcie_single_port_device_agent_hdl dm_vip_as_vip_link2(vip_if_link2);
```

// Second link: This is VIP instance, acting as Application VIP on second link sitting above the DUT TL layer

```
svt_pcie_if app_vip_if_link2(clkreq_n[0]);
svt_pcie_single_port_device_agent_hdl dm_vip_as_dut_link2(app_vip_if_link2);
```

- Compile Time Configuration of Each Instance of VIP

// Compile time settings of PCIE LINK SIDE VIP instance of Link 1

```
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_DISPLAY_NAME = "dm_vip_as_vip_link1.";
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_PCIE_SPEC_VER =
`SVT_PCIE_UI_PCIE_SPEC_VER_4_0;
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_PIPE_SPEC_VER =
`SVT_PCIE_UI_PIPE_SPEC_VER_4_2;
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_NUM_PHYSICAL_LANES= 16;
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_HIERARCHY_NUMBER= 0;
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_MPIPE= 0;
`ifdef SVT_PCIE_TEST_SUITE_SERDES_TB
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES;
`else
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE;
`endif

`ifdef SVT_PCIE_TEST_SUITE_EP_IS_DUT
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_DEVICE_IS_ROOT= 1; // Configured as Root.
`else
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_DEVICE_IS_ROOT= 0; // Configured as Endpoint
`endif
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_CONNECT_ACTIVE_VIP= 1;
```

```
  defparam dm_vip_as_vip_link1.SVT_PCIE_UI_PIPE_CLK_FROM_MAC= 0;
```

//Compile time settings of PCIE Application side VIP instance of Link 1

//Highlighted code shows the parameters which are configured differently for PCIE Application side VIP

```
   defparam dm_vip_as_dut_link1.SVT_PCIE_UI_DISPLAY_NAME = "dm_vip_as_dut_link1.";
  defparam dm_vip_as_dut_link1.SVT_PCIE_UI_PCIE_SPEC_VER =
`SVT_PCIE_UI_PCIE_SPEC_VER_4_0;
  defparam dm_vip_as_dut_link1.SVT_PCIE_UI_PIPE_SPEC_VER =
`SVT_PCIE_UI_PIPE_SPEC_VER_4_2;
  defparam dm_vip_as_dut_link1.SVT_PCIE_UI_HIERARCHY_NUMBER= 0;
  defparam dm_vip_as_dut_link1.SVT_PCIE_UI_MPIPE= 1;
`ifdef SVT_PCIE_TEST_SUITE_VIP_DUT
  `ifdef SVT_PCIE_TEST_SUITE_SERDES_TB
    defparam dm_vip_as_dut_link1.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES;
  `else
    defparam dm_vip_as_dut_link1.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE;
  `endif
`endif

  `ifdef SVT_PCIE_TEST_SUITE_EP_IS_DUT
     defparam dm_vip_as_dut_link1.SVT_PCIE_UI_DEVICE_IS_ROOT = 0;
  `else
     defparam dm_vip_as_dut_link1.SVT_PCIE_UI_DEVICE_IS_ROOT = 1;
  `endif

  defparam dm_vip_as_dut_link1.SVT_PCIE_UI_CONNECT_ACTIVE_VIP= 1;
  defparam dm_vip_as_dut_link1.SVT_PCIE_UI_PIPE_CLK_FROM_MAC= 0;


`ifndef SVT_PCIE_TEST_SUITE_VIP_DUT
  // Deactivate the TL,DL,PL stack of VIP instance and substitute it with Controller DUT
TL,DL,PL stack. Only use the App layer logic of this VIP instance
  defparam dm_vip_as_dut_link1.SVT_PCIE_UI_PHY_INTERFACE_TYPE =
  `SVT_PCIE_UI_PHY_INTERFACE_TYPE_APP;
`endif
```

// Compile time settings of PCIE LINK SIDE VIP instance of Link 2

```
 defparam dm_vip_as_vip_link2.SVT_PCIE_UI_DISPLAY_NAME = "dm_vip_as_vip_link2.";
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_PCIE_SPEC_VER =
`SVT_PCIE_UI_PCIE_SPEC_VER_4_0;
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_PIPE_SPEC_VER =
`SVT_PCIE_UI_PIPE_SPEC_VER_4_2;
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_NUM_PHYSICAL_LANES= 16;
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_HIERARCHY_NUMBER= 0;
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_MPIPE= 0;
`ifdef SVT_PCIE_TEST_SUITE_SERDES_TB
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES;
`else
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE;
`endif
```

```
`ifdef SVT_PCIE_TEST_SUITE_EP_IS_DUT
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_DEVICE_IS_ROOT= 1;
`else
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_DEVICE_IS_ROOT= 0;
`endif

  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_CONNECT_ACTIVE_VIP= 1;
  defparam dm_vip_as_vip_link2.SVT_PCIE_UI_PIPE_CLK_FROM_MAC= 0;
```

//Compile time settings of PCIE APPLICATION SIDE VIP instance of Link 2

//Highlighted code shows the parameters which are configured differently for PCIE APP Side VIP

```
  defparam dm_vip_as_dut_link2.SVT_PCIE_UI_DISPLAY_NAME = "dm_vip_as_dut_link2.";
  defparam dm_vip_as_dut_link2.SVT_PCIE_UI_PCIE_SPEC_VER =
`SVT_PCIE_UI_PCIE_SPEC_VER_4_0;
  defparam dm_vip_as_dut_link2.SVT_PCIE_UI_PIPE_SPEC_VER =
`SVT_PCIE_UI_PIPE_SPEC_VER_4_2;
  defparam dm_vip_as_dut_link2.SVT_PCIE_UI_HIERARCHY_NUMBER= 0;
  defparam dm_vip_as_dut_link2.SVT_PCIE_UI_MPIPE= 1;
`ifdef SVT_PCIE_TEST_SUITE_VIP_DUT
  `ifdef SVT_PCIE_TEST_SUITE_SERDES_TB
    defparam dm_vip_as_dut_link2.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES;
  `else
    defparam dm_vip_as_dut_link2.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE;
  `endif
`endif

  `ifdef SVT_PCIE_TEST_SUITE_EP_IS_DUT
     defparam dm_vip_as_dut_link2.SVT_PCIE_UI_DEVICE_IS_ROOT = 0;
  `else
     defparam dm_vip_as_dut_link2.SVT_PCIE_UI_DEVICE_IS_ROOT = 1;
  `endif

  defparam dm_vip_as_dut_link2.SVT_PCIE_UI_CONNECT_ACTIVE_VIP= 1;
  defparam dm_vip_as_dut_link2.SVT_PCIE_UI_PIPE_CLK_FROM_MAC= 0;


`ifndef SVT_PCIE_TEST_SUITE_VIP_DUT
```

// Deactivate the TL,DL,PL stack of VIP instance and substitute it with Controller DUT TL,DL,PL stack. Only use the App layer logic of this VIP instance

```
  defparam dm_vip_as_dut_link2.SVT_PCIE_UI_PHY_INTERFACE_TYPE =
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_APP;
`endif
```

- Invoke update_if_variables(...) for Each VIP Instance

```
initial begin
  dm_vip_as_vip1.update_if_variables(4'h0, //port_id
                          4'h0, //link_id
                          "uvm_test_top" // UVM class instance to which the Active VIP Interface will be targetted
                          ,"" // UVM class instance to which the Passive VIP Interface will be targetted . Not used for now
                          ,1  // Bit to enable whether link_id value should be used in construction of string used to store the ACTIVE VIP  virtual interface handle in uvm_config_db
                          ,0  // Bit to enable whether link_id value should be used in construction of string used to store the Passive VIP virtual interface handle in uvm_config_db
                          );
  dm_vip_as_dut1.update_if_variables(4'h1, //port_id
                          4'h0, //link_id
                          "uvm_test_top" // UVM class instance to which the Active VIP Interface will be targetted
                          ,"" // UVM class instance to which the Passive VIP Interface will be targetted . Not used for now
                          ,1  // Bit to enable whether link_id value should be used in construction of string used to store the ACTIVE VIP  virtual interface handle in uvm_config_db
                          ,0  // Bit to enable whether link_id value should be used in construction of string used to store the Passive VIP virtual interface handle in uvm_config_db
                          );


 dm_vip_as_vip2.update_if_variables(4'h0, //port_id
                          4'h1, //link_id
                          "uvm_test_top" // UVM class instance to which the Active VIP Interface will be targetted
                          ,"" // UVM class instance to which the Passive VIP Interface will be targetted . Not used for now
                          ,1  // Bit to enable whether link_id value should be used in construction of string used to store the ACTIVE VIP  virtual interface handle in uvm_config_db
                          ,0  // Bit to enable whether link_id value should be used in construction of string used to store the Passive VIP virtual interface handle in uvm_config_db
                          );
 dm_vip_as_dut2.update_if_variables(4'h1, //port_id
                          4'h1, //link_id
                          "uvm_test_top" // UVM class instance to which the Active VIP Interface will be targetted
                          ,"" // UVM class instance to which the Passive VIP Interface will be targetted . Not used for now
                          ,1  // Bit to enable whether link_id value should be used in construction of string used to store the ACTIVE VIP  virtual interface handle in uvm_config_db
                          ,0  // Bit to enable whether link_id value should be used in construction of string used to store the Passive VIP virtual interface handle in uvm_config_db
                          );
```

Pass the `update_if_variables` interface instance to `uvm_test_top`. For more details, see “Using a Virtual Interface Handle", in the *VC Verification IP PCIe UVM User Guide*.

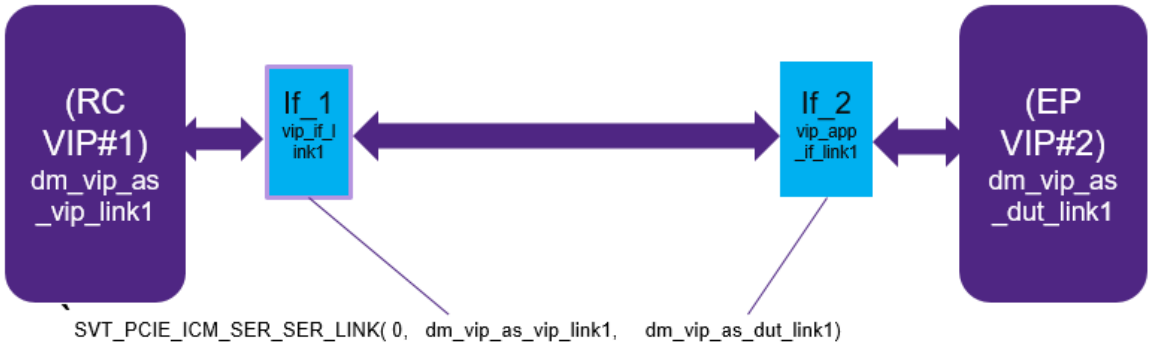
- Clock and Reset Supply to PCIe Application VIP

// Supply clock and reset to the VIP whose application layer is active but TL,DL and PL stack is INACTIVE

```
  always @(*)  dm_vip_as_dut_link1.vip_port_if.app_if.reset   =
u_dut1.cts.u_application_model.m_app.reset; // input to VIP instance to reset the app
layer which is driving the DUT TL,DL,PL stack
   always @(*)  dm_vip_as_dut_link1.vip_port_if.app_if.appl_clk            =
u_dut1.cts.u_application_model.m_app.appl_clk; // input to VIP instance for app layer
which driving the DUT TL,DL,PL stack
   always @(*)  dm_vip_as_dut_link2.vip_port_if.app_if.reset   =
u_dut2.cts.u_application_model.m_app.reset; // input to VIP instance to reset the app
layer which is driving the DUT TL,DL,PL stack   always @(*)
dm_vip_as_dut_link2.vip_port_if.app_if.appl_clk            =
u_dut2.cts.u_application_model.m_app.appl_clk; // input to VIP instance for app layer
which driving the DUT TL,DL,PL stack
```

- VIP-VIP Connection

  In VIP-VIP mode, both sides of VIP are connected as specified in the below diagram with Helper macro.



(VIP-VIP connected by using Helper Macro)
This macro provide connection between interfaces of instances mentioned in argument

```
 // Serial Connection
`ifdef SVT_PCIE_TEST_SUITE_SERDES_TB
//Connect PCIe Serial interfaces of VIP instances to form PCIe Serial link.
//Macro assumes the arguments as link_number, a serial vip instance , another serial vip
instance
  `SVT_PCIE_ICM_SER_SER_LINK(0,dm_vip_as_vip_link1,dm_vip_as_dut_link1)
  `SVT_PCIE_ICM_SER_SER_LINK(1,dm_vip_as_vip_link2,dm_vip_as_dut_link2)
// PIPE connection
`else
  `SVT_PCIE_ICM_PIPE_PIPE_LINK(0,dm_vip_as_vip_link1,dm_vip_as_dut_link1)
  `SVT_PCIE_ICM_PIPE_PIPE_LINK(1,dm_vip_as_vip_link2,dm_vip_as_dut_link2)
`endif
```

- DUT Instantiation

  Based on the framework, you must instantiate an appropriate DUT instance. For illustration, two DUT instances are shown below.

  //DUT instance 1

```
dut inst_dut1 (…);
```

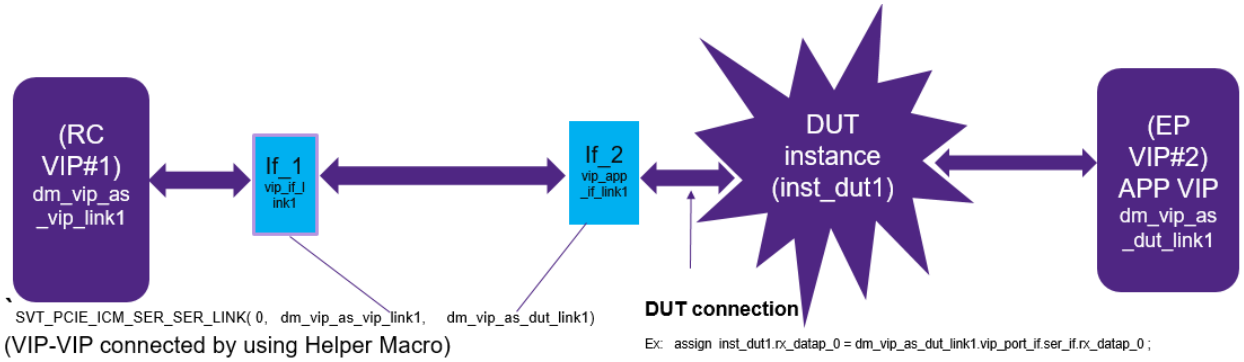
  //DUT instance 2

```
dut inst_dut2 (…);
```

- DUT Integration

  DUT can be integrated with the following two approaches:

- Approach 1: You can directly connect DUT signals to interface(if_2) of the link. The following diagram illustrates only link#1 flow of DUT connections. Helper macro provides connection between interfaces. (as discussed in section VIP-VIP Connection).



// VIP1↔DUT 1 for Link 1

// Per lane Serial interconnect code to be replicated as per DUT requirement in terms of number of lanes. Code

//placed below is only applicable for lane 0

```
`ifdef SERDES_TB
assign  inst_dut1.rx_datap_0 =
dm_vip_as_dut_link1.vip_port_if.ser_if.rx_datap_0 ;
assign  inst_dut1.rx_datan_0 =
dm_vip_as_dut_link1.vip_port_if.ser_if.rx_datan_0 ;
assign  dm_vip_as_dut_link1.vip_port_if.ser_if.tx_datap_0 =
inst_dut1.tx_datap_0 ;
assign dm_vip_as_dut_link1.vip_port_if.ser_if.tx_datan_0 =
inst_dut1.tx_datan_0 ;
`else
```

// PIPE Interconnect code irrespective of number of lanes

```
always @(*) inst_dut1.pclk = dm_vip_as_dut_link1.vip_port_if.pipe_if.pclk;
assign   inst_dut1.max_pclk =
dm_vip_as_dut_link1.vip_port_if.pipe_if.max_pclk;
assign   inst_dut1.reset = common_pwr_on_reset;
assign   dm_vip_as_dut_link1.vip_port_if.pipe_if.rate = inst_dut1.rate;
assign  dm_vip_as_dut_link1.vip_port_if.pipe_if.pclk_rate =
inst_dut1.pclk_rate;
. . . . . .
. . . . . .
```

// per lane PIPE interconnect code to be replicated as per DUT requirement in terms of number

// of lanes

```
assign inst_dut1.rx_data_0 = dm_vip_as_dut_link1.vip_port_if.pipe_if.rx_data_0
;
assign inst_dut1.rx_data_k_0 =
dm_vip_as_dut_link1.vip_port_if.pipe_if.rx_data_k_0 ;
assign inst_dut1.rx_status_0 =
dm_vip_as_dut_link1.vip_port_if.pipe_if.rx_status_0 ;
assign inst_dut1.rx_valid_0 =
dm_vip_as_dut_link1.vip_port_if.pipe_if.rx_valid_0 ;
. . . . . .
```

```
. . . . . .
assign dm_vip_as_dut_link1.vip_port_if.pipe_if.rx_polarity_0 =
inst_dut1.rx_polarity_0 ;
assign dm_vip_as_dut_link1.vip_port_if.pipe_if.tx_data_0 = inst_dut1.tx_data_0
;
assign dm_vip_as_dut_link1.vip_port_if.pipe_if.tx_data_k_0 =
inst_dut1.tx_data_k_0 ;
assign dm_vip_as_dut_link1.vip_port_if.pipe_if.tx_ei_code_0 =
inst_dut1.tx_ei_code_0 ;
assign dm_vip_as_dut_link1.vip_port_if.pipe_if.tx_compliance_0 =
inst_dut1.tx_compliance_0 ;
assign dm_vip_as_dut_link1.vip_port_if.pipe_if.tx_elec_idle_0
=inst_dut1.tx_elec_idle_0 ;
assign dm_vip_as_dut_link1.vip_port_if.pipe_if.tx_data_valid_0 =
inst_dut1.tx_data_valid_0 ;
. . . . . .
. . . . . .
`endif
```

// VIP2↔DUT 2 for Link 2

// Per lane Serial interconnect code to be replicated as per DUT requirement in terms of number of lanes. Code

// placed below is only applicable for lane 0

```
`ifdef SERDES_TB
assign inst_dut2.rx_datap_0 =
dm_vip_as_dut_link2.vip_port_if.ser_if.rx_datap_0 ;
assign inst_dut2.rx_datan_0 =
dm_vip_as_dut_link2.vip_port_if.ser_if.rx_datan_0 ;
assign dm_vip_as_dut_link2.vip_port_if.ser_if.tx_datap_0 =
inst_dut2.tx_datap_0 ;
assign dm_vip_as_dut_link2.vip_port_if.ser_if.tx_datan_0 =
inst_dut2.tx_datan_0 ;
`else
```

// PIPE Interconnect code irrespective of number of lanes

```
always @(*) inst_dut2.pclk = dm_vip_as_dut_link2.vip_port_if.pipe_if.pclk;
assign inst_dut2.max_pclk = dm_vip_as_dut_link2.vip_port_if.pipe_if.max_pclk;
assign inst_dut2.reset = common_pwr_on_reset;
assign dm_vip_as_dut_link2.vip_port_if.pipe_if.rate = inst_dut2.rate;
assign dm_vip_as_dut_link2.vip_port_if.pipe_if.pclk_rate =
inst_dut2.pclk_rate;
. . . . . .
. . . . . .
```

// per lane PIPE interconnect code to be replicated as per DUT requirement in terms of number

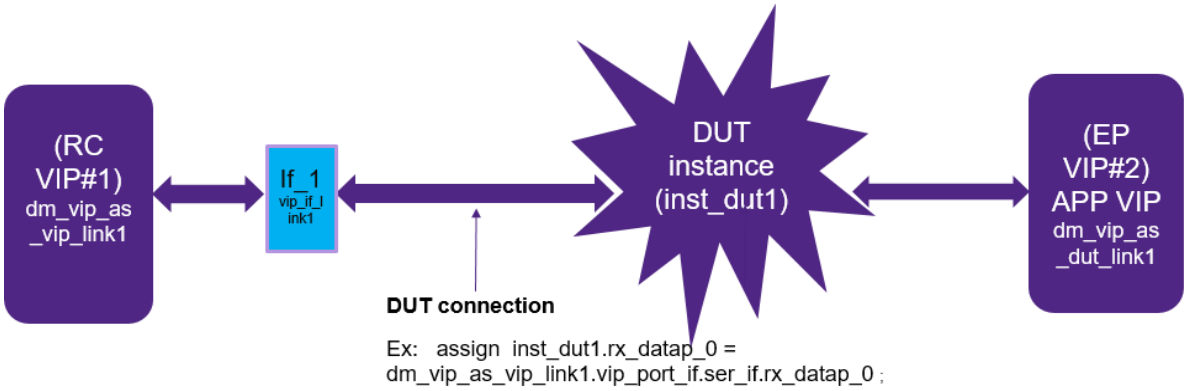
// of lanes

```
assign inst_dut2.rx_data_0 = dm_vip_as_dut_link2.vip_port_if.pipe_if.rx_data_0
;
assign inst_dut2.rx_data_k_0 =
dm_vip_as_dut_link2.vip_port_if.pipe_if.rx_data_k_0 ;
assign inst_dut2.rx_status_0 =
dm_vip_as_dut_link2.vip_port_if.pipe_if.rx_status_0 ;
assign inst_dut2.rx_valid_0 =
dm_vip_as_dut_link2.vip_port_if.pipe_if.rx_valid_0 ;
. . . . . .
. . . . . .
```

```
assign dm_vip_as_dut_link2.vip_port_if.pipe_if.rx_polarity_0 =
inst_dut2.rx_polarity_0 ;
assign dm_vip_as_dut_link2.vip_port_if.pipe_if.tx_data_0 = inst_dut2.tx_data_0
;
assign dm_vip_as_dut_link2.vip_port_if.pipe_if.tx_data_k_0 =
inst_dut2.tx_data_k_0 ;
assign dm_vip_as_dut_link2.vip_port_if.pipe_if.tx_ei_code_0 =
inst_dut2.tx_ei_code_0 ;
assign dm_vip_as_dut_link2.vip_port_if.pipe_if.tx_compliance_0 =
inst_dut2.tx_compliance_0 ;
assign dm_vip_as_dut_link2.vip_port_if.pipe_if.tx_elec_idle_0
=inst_dut2.tx_elec_idle_0 ;
assign dm_vip_as_dut_link2.vip_port_if.pipe_if.tx_data_valid_0 =
inst_dut2.tx_data_valid_0 ;
. . . . . .
. . . . . .
`endif
```

For details about macros, see file *tb_dut_pcie/env/hdl_interconnect_macros.sv*. These macros were created using the approach listed in section "Helper Macros" in the *VC Verification IP PCIe UVM User Guide*.

- Approach 2: For second approach, you can connect DUT signals directly to first instance of VIP interface(if_1).



9.4 Multi-Link Use Model

- Configuration to enable multi-link env (mandatory configuration).
  - Use the macro `SNPS_PCIE_MULTI_LINK_ENV` for running simulation in multi-link environment.
  - Modify the `highest_link_num` setting in the testbench.
    - Single link: Default value is 0.
    - Multi-link: Default value is 1 as two link example is discussed in this chapter.
    - You must change this value depending on number of links you want to create. For three links, the value should be 2 and so on.

- Based on fixed link size, pass the equivalent information through plusarg `+svt_pcie_disable_link_creation`.

Compile Time Selected Links	Value of plusarg
1	32'hFFFF_FFFE
2	32'hFFFF_FFFC
3	32'hFFFF_FFF8

Note By default, 1 link is active. For multi-link, you must pass this in .scr file or any .f file which can be passed to simulator executable.

- Setting Configuration Attributes

This section describes the use model for setting configuration attributes of VIP/Test Suite (for example, link_width and link_speed).

The configurations can be controlled by the following two approaches

- Runtime Control of VIP

  In this approach, VIP attributes can be configured at runtime without recompiling by using SCR file format *.scr, and/or .txt file format txt files (recommended approach).

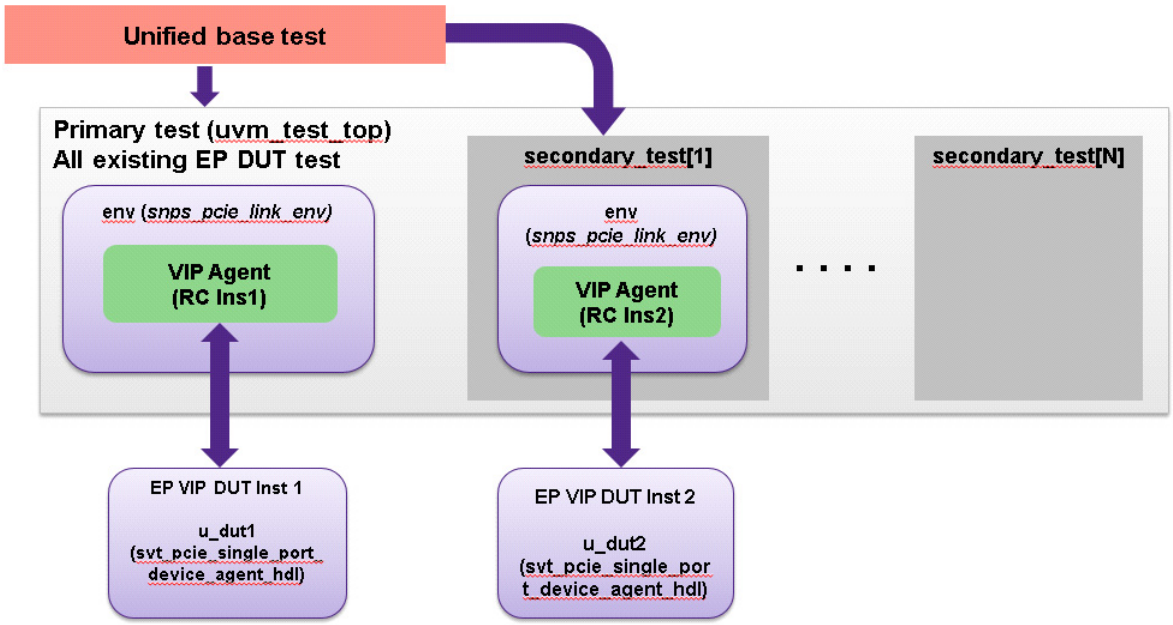
  This approach provides runtime controls similar to plusargs but with instance level granularity avoiding recompilation. For more details, see Test Suite Testbench Architecture.

- Compile Time Control of VIP

  In this approach, each VIP related configuration is directly programmed in user base test.

  Example is shown in pcie_ep_dut_cust_base_test and pcie_rc_dut_cust_base_test.

- Hierarchy and Selection of Test on Each Link



  - Hierarchy for each link
    - `Link 0:uvm_test_top.env`
    - `Link 1:uvm_test_top.secondary_tests[1].env`
  - Test selection on each link
    - Primary test selection on link 1

      Direct test selection of test will run on primary link.

      Example: If you want to run test case `tl_tx_mem_mapped-ep` on primary link, then directly pass test name with `+UVM_TESTNAME= tl_tx_mem_mapped-ep`.
    - Secondary test selection on link 2

      To select the secondary test, use the following plusarg.

      Example: If you want to run test case `tl_tx_msg_transactions` on second link whereas test passed directly with `+UVM_TESTNAME` is `tl_tx_mem_mapped-ep`, then use below plusarg `+uvm_set_inst_override=pcie_unified_base_test,tl_tx_msg_transactions,uvm_test_top.secondary_tests[1].`

      Therefore, now the test `tl_tx_mem_mapped` will run on link_0 whereas test `tl_tx_msg_transactions` will run on link_2.

Note This plusarg can be passed in .scr file or any .f file. Avoid passing plusargs via command line, and use files instead.

- Example Command for Runtime Control of Attributes
  - To run different test on each link and to specify test for first link through `UVM_TESTNAME` and other through .scr file.

```
gmake tl_tx_mem_mapped-ep
```

```
USE_SIMULATOR=vcsvlog
SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_multi_link_ep_rc_pue_
iip.svi
SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_multilink_ep_pue_ii
p_serial.f SVT_PCIE_TEST_SUITE_PLUSARG_FILE=tests/<file_name>.scr OR
+svt_pcie_test_suite_all_link_prop_file=<file_name>.txt(check usage for SVT
property)
```

- ✧ Primary test will run on link 1 => tl_tx_mem_mapped-ep
- ✧ Secondary test will run on link 2 (specified in .scr file through +uvm_set_inst_override=pcie_unified_base_test,tl_tx_msg_transactions,uvm_test_top.secondary_tests[1]) => tl_tx_msg_transactions-ep
- ✧ All the plusarg and configuration attributes are passed in .scr/.txt file.

✦ To run different test on each link and wants to specify test for each link through .scr file

```
gmake
USE_SIMULATOR=vcsvlog
SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_multi_link_ep_rc_pue_
iip.svi
SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_multilink_ep_pue_ii
p_serial.f SVT_PCIE_TEST_SUITE_PLUSARG_FILE=tests/<file_name>.scr OR
+svt_pcie_test_suite_all_link_prop_file=<file_name>.txt(check usage for SVT
property)
```

- ✧ Primary test will run on link 1 (specified in .scr file through +uvm_set_inst_override=pcie_unified_base_test,tl_tx_mem_mapped,uvm_test_top) => tl_tx_msg_transactions-ep
- ✧ Secondary test will run on link 2 (specified in .scr file through +uvm_set_inst_override=pcie_unified_base_test,tl_tx_msg_transactions,uvm_test_top.secondary_tests[1]) => tl_tx_msg_transactions-ep
- ✧ All the plusarg and configuration attributes are passed in .scr/.txt file.

✦ To run same test on each link, it can either be passed on command line through +UVM_TESTNAME or in .scr file.

❖ Example Command for Compile-Time Control of Attributes

```
gmake tl_tx_mem_mapped-ep
USE_SIMULATOR=vcsvlog
SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_multi_link_ep_rc_pue_iip.svi
SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_multilink_ep_pue_iip_serial.f
```

You must configure each instance configuration separately in your locally saved base test.

✦ For link 1

```
all_links_cfg.Cfg[0].rc_cfg.pcie_cfg.pl_cfg.link_width = 2;
all_links_cfg.Cfg[0].ep_cfg.pcie_cfg.pl_cfg.link_width = 2;
```

✦ For link 2

```
all_links_cfg.Cfg[1].rc_cfg.pcie_cfg.pl_cfg.link_width = 4;
all_links_cfg.Cfg[1].ep_cfg.pcie_cfg.pl_cfg.link_width = 4;
```

❖ Log File/Dump Naming Convention

✦ Transcript

A single transcript is generated for all links.

```
<testname>-<vip_mode>.transcript
```

- Transaction log

  Separate transaction log is generated for all links

  ```
  <testname>_<link_num>_<vip_mode>.xact_log
  ```

- Symbol Log

  Separate symbol log is generated for all links

  ```
  <testname>_<link_num>_<vip_mode>.sym_log
  ```

- Dump name

  ```
  <testname>_<vip_mode>.vpd
  <testname>_<vip_mode>.fsdb
  ```

Note <testname> is the primary test name.

Example 9-1

Considering 2 links such that test `tl_tx_mem_mapped` is operating on one link and test `tl_tx_msg_transaction` is running on second link such that VIP is operating as RC.

- Transcript

  `Tl_tx_mem_mapped-rc.transcript`

- Transaction and symbol log
  - Link 0

    `tl_tx_mem_mapped_0_rc.xact_log`

    `tl_tx_mem_mapped_0_rc.sym_log`
  - Link 1

    `tl_tx_msg_transaction_1_rc.xact_log`

    `tl_tx_msg_transaction_1_rc.sym_log`
- Dump Name

  `tl_tx_mem_mapped-rc.vpd`

- Limitations
  - Current structure does not support EP DUT and RC DUT operation in a single simulation.

Note For large number of topology combinations, Synopsys recommends runtime approach discussed in section Configurable Topology (Runtime Approach).

9.4.1 File Formats

- scr format

- First Format

```
+svt_pcie_disable_link_creation=32'hFFFF_FFFC
+uvm_set_inst_override=pcie_unified_base_test,tl_tx_msg_transactions,uvm_test_top.secondary_tests[1]
+svt_pcie_test_suite_all_link_prop_file=<file_name>.txt
```

  - In this format, plusargs are passed directly in .scr file whereas configuration settings are done in a .txt file which are again invoked using another plusarg.
  - This is Synopsys recommended format as this is more human readable and less prone to errors.

- Second Format

```
+svt_pcie_disable_link_creation=32'hFFFF_FFFC
+uvm_set_inst_override=pcie_unified_base_test,tl_tx_msg_transactions,uvm_test_top.secondary_tests[1]
// Following plusarg does width over ride for link 0
+cfg[0]=ep_cfg.pcie_cfg.pl_cfg.target_speed:32'h1e,rc_cfg.pcie_cfg.pl_cfg.target_speed:32'h10,ep_cfg.pcie_cfg.pl_cfg.supported_speeds:32'h1e
// Following plusarg does width over ride for link 1
+cfg[1]=ep_cfg.pcie_cfg.pl_cfg.target_speed:32'h4,rc_cfg.pcie_cfg.pl_cfg.target_speed:32'h4,ep_cfg.pcie_cfg.pl_cfg.supported_speeds:32'h6

```

  - In this format, plusargs and configurations settings are passed directly in .scr file.
  - Here all the configuration setting per link should be in single line.
  - This format is more prone to errors and not human readable. Therefore, this is not a recommended approach.

Note

- For both the approaches values of configuration attributes are assigned based on data type of attribute. For more details, see *svt_pcie_test_suite_name_value_pair_sample_file.txt* in tb_dut_pcie.
- Configurations of one link can also be cloned to other link.
- In .scr file, back slash(/) does not work well in the value of string.

- txt format

```
//Setting sfor link 0
cfg[0].ep_is_dut=1
cfg[0].ep_cfg.device_is_root=0
//Settings for link 1
cfg[1].ep_is_dut=1
cfg[1].ep_cfg.device_is_root=0

```

This file can either be invoked by .scr file or can be passed directly on the command line using plusarg `+svt_pcie_test_suite_all_link_prop_file=<file_name>.txt`

Note

- For both the approaches values of configuration attributes are assigned based on data type of attribute. For more details, see *svt_pcie_test_suite_name_value_pair_sample_file.txt* in tb_dut_pcie.
- Configurations of one link can also be Replicating Configuration Attributes of One Link to Another cloned to other link.
- In .scr file, back slash(/) does not work well in the value of string.
- In this approach, along with other attributes you must set version specific attributes—that is,
  - `cfg[0].rc_cfg.pcie_spec_ver`
  - `cfg[0].ep_cfg.pcie_spec_ver`
  - `cfg[0].rc_cfg.pipe_spec_ver`
  - `cfg[0].ep_cfg.pipe_spec_ver`

❖ Replicating Configuration Attributes of One Link to Another

✦ Example to clone configuration attributes of link0 to link1

```
Cfg[1].link_cfg_to_be_cloned=0
```

✦ Example to clone configuration attributes of link9 to link16

```
Cfg[16].link_cfg_to_be_cloned=9
```

10 Frequently Asked Questions

This chapter discusses the following topics:

10.1 Errors

- `SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE`: If this switch is used incorrectly, then the default compile file *compile_dut_snps_pcie_bifurcated_phy_vip.f* will be selected. This could create an incorrect combination of compile and topology files leading to errors.
- `SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE`: If this switch is used incorrectly, then the default topology file *topology_dut_snps_pcie_bifurcated_phy_vip.svi* will be selected. This could create an incorrect combination of compile and topology files leading to errors.
- `UVM_TESTNAME`: Usage of this switch with `gmake` option will lead to an error.

A
Reporting Problems

This chapter discusses the following topics:

A.1 Introduction

This chapter outlines the process for working through and reporting VIP transactor issues encountered in the field. It describes the data you must submit when a problem is initially reported to Synopsys. After a review of the initial information, Synopsys may decide to request adjustments to the information being requested, which is the focus of the next section. This section outlines the process for working through and reporting problems. It shows how to use Debug Automation to enable all the debug capabilities of any VIP. In addition, the VIP provides a case submittal tool to help you pack and send all pertinent debug information to Synopsys Support.

A.2 Debug Automation

Every Synopsys model contains a feature called “debug automation”. It is enabled through *svt_debug_opts* plusarg. The Debug Automation feature allows you to enable all relevant debug information. The following are critical features of debug automation:

- Enabled by the use of a command line run-time plusarg.
- Can be enabled on individual VIP instances or multiple instances using regular expressions.
- Enables debug or verbose message verbosity:
  - The timing window for message verbosity modification can be controlled by supplying `start_time` and `end_time`.

- Enables at one time any, or all, standard debug features of the VIP:
  - Transaction Trace File generation
  - Transaction Reporting enabled in the transcript
  - PA database generation enabled
  - Debug Port enabled
  - Optionally, generates a file name *svt_model_out.fsdb* when Verdi libraries are available

When the Debug feature is enabled, then all VIP instances that are enabled for debug will have their messages routed to a file named *svt_debug.transcript* steps. You can use the generated debug data for your own debugging, or as debug data you would send to Synopsys support.

A.3 Enabling and Specifying Debug Automation Features

Debug Automation is enabled through the use of a run-time plusarg named +*svt_debug_opts. T*his plusarg accepts an optional string-based specification to control various aspects Debug Automation. If this command control specification is not supplied, then the feature will default to being enabled on all VIP instances with the default options listed as follows:

Note the following about the plusarg:

- The command control string is a comma separated string that is split into the multiple fields.
- All fields are optional and can be supplied in any order.

The command control string uses the following format (white space is disallowed):

```
inst:<inst>,type:<string>,feature:<string>,start_time:<longint>,end_time:<longint>,verb
osity:<string>
```

The following table explains each control string:

Table A-1 Control Strings for Debug Automation plusarg

Field	Description
inst	Identifies the VIP instance to apply the debug automation features. Regular expressions can be used to identify multiple VIP instances. If this value is not supplied, and if a type value is not supplied, then the debug automation feature will be enabled on all VIP instances.
type	Identifies a class type to apply the debug automation features. When this value is supplied then debug automation will be enabled for all instances of this class type.
feature	Identifies a sub-feature that can be defined by VIP designers to identify smaller grouping of functionality that is specific to that title. The definition and implementation of this field is left to VIP designers, and by default it has no effect on the debug automation feature. (Specific to VIP titles)
start_time	Identifies when the debug verbosity settings will be applied. The time must be supplied in terms of the timescale that the VIP is compiled. If this value is not supplied, then the verbosity settings will be applied at time zero.
end_time	Identifies when the debug verbosity settings will be removed. The time must be supplied in terms of the timescale that the VIP is compiled. If this value is not supplied, then the debug verbosity remains in effect until the end of the simulation.

Table A-1 Control Strings for Debug Automation plusarg (Continued)

Field	Description
verbosity	Message verbosity setting that is applied at the `start_time`. Two values are accepted in all methodologies: DEBUG and VERBOSE. UVM and OVM users can also supply the verbosity that is native to their respective methodologies (UVM_HIGH/UVM_FULL and OVM_HIGH/OVM_FULL). If this value is not supplied then the verbosity defaults to DEBUG/UVM_HIGH/OVM_HIGH. When this feature is enabled, then all VIP instances that are enabled for debug will have their messages routed to a file named `svt_debug.transcript`.

Examples:

Enable on all VIP instances with default options:

```
+svt_debug_opts
```

Enable on all instances:

- containing the string "endpoint" with a verbosity of UVM_HIGH
- starting at time zero (default) until the end of the simulation (default):

```
+svt_debug_opts=inst:/.*endpoint.*/,verbosity:UVM_HIGH
```

Enable on all instances:

- starting at time 1000 until time 1500:

```
+svt_debug_opts=start_time:1000,end_time:1500,verbosity:VERBOSE
```

Enable debug feature on all instances using default options:

- By setting the macro DEBUG_OPTS to 1 in the command line, the debug feature is enabled on all instances using default options. The macro will enable the XMLs and Trace files.

```
gmake <testname> DEBUG_OPTS=1 PA=FSDB
```

Note

The DEBUG_OPTS option is available through the installed VIP examples, but if required, in customer environments, then a similar feature should be added to their environment.
The PA=FSDB option is available in public examples and is required to enable Verdi libraries, and that when this option is used, then the Debug Opts file will record VIP activity to a file named `svt_model_log.fsdb`.
In addition, the SVT Automated Debug feature will enable waveform generation to an FSDB file, if the Verdi libraries are available. When enabled this feature, it should cause the simulator to dump waveform information only for the VIP interfaces.

When this feature is enabled then all VIP instances that have been enabled for debug will have their messages routed to a file named `svt_debug.transcript`.

A.4 Debug Automation Outputs

The Automated Debug feature generates a *svt_debug.out* file. It records important information about the debug feature itself, and data about the environment that the VIPs are operating in. This file records the following information:

- The compiled timeunit for the SVT package
- The compiled timeunit for each SVT VIP package

- Version information for the SVT library
- Version information for each SVT VIP
- Every SVT VIP instance, and whether the VIP instance has been enabled for debug
- For every SVT VIP enabled for debug, a list of configuration properties that have been modified to enable debug will be listed
- A list of all methodology phases will be recorded, along with the start time for each phase

The following are the output files generated:

- *svt_debug.out*: It records important information about the debug feature itself, and data about the environment that the VIPs are operating. One file is optionally created when this feature is enabled, depending on if the Verdi libraries are available.
- *svt_debug.transcript*: Log files generated by the simulation run.
- *transaction_trace*: Log files that records all the different transaction activities generated by VIPs.
- *svt_model_log.fsdb*: Contains PA FSDB information (if the VIP supports this), and which contains other recorded activity. The additional information records signal activity associated with the VIP interface, TLM input (through SIPP ports), other TLM output activity, configurations applied to the VIP, and all callback activity (recorded by before and after callback execution).

A.5 FSDB File Generation

To enable FSDB writing capabilities, the simulator compile-time options and environment must be updated to enable this. The steps to enable this are specific to the simulator being used (the {LINUX/LINUX64} label needs to be replaced based on the platform being used). The ability to write to an FSDB file requires that the user supplies the Verdi dumper libraries when they compile their testbench. If these are not supplied then the VIP will not be enabled to generate the *svt_model_log.fsdb* file.

A.5.1 VCS

The following must be added to the compile-time command:

-debug_access

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

A.5.2 Questa

The following must be added to the compile-time command:

```
+define+SVT_FSDB_ENABLE -pli novas_fli.so
```

A.5.3 Incisive

The following must be added to the compile-time command:

```
+define+SVT_FSDB_ENABLE -access +r
```

A.6 Initial Customer Information

Follow these steps when you call the Synopsys Support Center:

1. Before you contact technical support, be prepared to provide the following:

- A description of the issue under investigation.
- A description of your verification environment.

Enable the Debug Opts feature. For more information, see the Debug Automation.

A.7 Sending Debug Information to Synopsys

To help you debug testing issues, follow the given instructions to pack all pertinent debug information into one file which you can send to Synopsys (or to other users in your company):

1. Create a description of the issue under investigation. Include the simulation time and bus cycle of the failure, as well as any error or warning messages that are part of the failure.
2. Create a description of your verification environment. Assemble information about your simulation environment, making sure to include:
   - OS type and version
   - Testbench language (SystemVerilog or Verilog)
   - Simulator and version
   - DUT languages (Verilog)
3. Use the VIP case submittal tool to pack a file with the appropriate debug information. It has the following usage syntax:

   ```
   $DESIGNWARE_HOME/bin/snps_vip_debug [-directory <path>]
   ```

   The tool will generate a “<username>.<uniqid>.svd” file in the current directory. The following files are packed into a single file:
   - FSDB
   - HISTL
   - MISC
   - SLID
   - SVTO
   - SVTX
   - TRACE
   - VCD
   - VPD
   - XML

   If any one of the above files are present, then the files will be saved in the "<username>.<uniqid>.svd" in the current directory. The simulation transcript file will not be part of this and it will be saved separately.

   The -directory switch can be specified to select an alternate source directory.
4. You will be prompted by the case submittal tool with the option to include additional files within the SVD file. The simulation transcript files cannot be automatically identified and it must be provided during this step.

For SVT, you must set the verbosity to UVM_HIGH/OVM_HIGH.

5. The case submittal tool will display options on how to send the file to Synopsys.

A.8 Limitations

Enabling DEBUG or VERBOSE verbosity is an expensive operation, both in terms of runtime and disk space utilization. The following steps can be used to minimize this cost:

- Only enable the VIP instance necessary for debug. By default, the `+svt_debug_opts` command enables Debug Opts on all instances, but the `'inst'` argument can be used to select a specific instance.
- Use the `start_time` and `end_time` arguments to limit the verbosity changes to the specific time window that needs to be debugged.

A.9 Multi-Link Changes

This section discusses the changes (from N-2018.03-3 to O-2018.06-1 release) in the sequence/test cases/callback to make them multi-link compatible.

Note

For users migrating from N-2018.03-3 or previous releases, the changes listed in Table A-2 are applicable. All these changes are updated in the STAR 9001333361.

- For backward incompatible change, you must make changes given in section A.9.1 to make it compatible.
- For backward compatible change, there is no action required from the user.

Table A-2 Summary of Changes

Category	Type of Changes	Is this Backward Incompatible Change	Comments
Backward Incompatible Change	Avoid/Minimize the use of static variables	Yes	
	Sequence to Application BFM communication	Yes	
	Test to lower level component communication	Yes (Partial)	If user has copied sequence/test and created their own test.
	Sequence to test communication	Yes (Partial)	If user has copied sequence /test and created their own test.
	Sequence and callback communication	Yes (Partial)	If user has copied callback, sequence/test and created their own test.
	Error demotion logic and its callback registrations	Yes (Partial)	If user has copied sequence/test and created their own test.

Table A-2 Summary of Changes

Category	Type of Changes	Is this Backward Incompatible Change	Comments
Backward Compatible Change	Do not use disable <label_name>	No	

A.9.1 Backward Incompatible Change

- Avoid/Minimize the use of static variables

  This is a backward incompatible change, it will affect both single link and multi-link users.

  - Changed the following static type attributes to non-static type in `test_siute_tl_status`:
    - `max_num_functions_supported_in_ep`
    - `max_num_functions_supported_in_ep`
  - This makes these variables inaccessible by scope resolution operator.
  - Due to this change, you must move the existing code in your base test `build_phase` to `connect_phase`:

build_phase	connect_phase
`test_suite_tl_status::max_num_functions_supported_in_ep`	`test_suite_tl_status.max_num_functions_supported_in_ep`
`test_suite_tl_status::max_num_functions_supported_in_ep`	`test_suite_tl_status.max_num_functions_supported_in_ep`

- Sequence to Application BFM communication

  This is a backward incompatible change that will affect both single link and multi-link testbenches. Communication from sequence to Application BFM is enhanced as shown below:

  - Inside the sequence `set` call of `uvm_config_db` is enhanced by adding `link_cfg.get_path_to_app_bfm()` for providing accurate hierarchy of Application BFM.

    For example, the sequence changes are done for one of the following examples:

Existing Code in Sequence	Modified Code
`uvm_config_db#(bit)::set(this, "*","assert_app_err_signals",1'b1);`	`uvm_config_db#(bit)::set(null, link_cfg.get_path_to_app_bfm(),"assert_app_err_signals",1'b1);`

  - Change 1: Following `uvm_config_db::set` call is present in `pcie_unified_base_test`.

    ```
    uvm_config_db#(string)::set(null,cfg.get_path_to_app_bfm(),"path_to_ts_sequencer",{cfg.get_path_to_test_env_seqr(),"*"});
    ```

    External Application BFM must retrieve `path_to_ts_sequencer` from `uvm_config_db` to communicate information back to sequences. You must add corresponding `uvm_config_db ::get` call in your Application BFM.

```
uvm_config_db#(string)::get(m_parent, "","path_to_ts_sequencer",
path_to_ts_sequencer));
```

- Change 2: Sequence to Application BFM

  Inside Application BFM, `uvm_config_db#::get` call has been updated as shown below, you must make sure that `get` call is working correctly.

Change from	to
`void'(uvm_config_db#(bit)::get(this, "", "assert_app_err_signals", assert_app_err_signals));`	`void'(uvm_config_db#(bit)::get(m_parent, "", "assert_app_err_signals", assert_app_err_signals));`

- Change 3: Application BFM to sequence

  Example of recommended approach to communicate information from `app_bfm` to sequence.

Change from	to
`uvm_config_db#(bit)::set(null, "*", "hp_int_asserted",1);`	`uvm_config_db#(bit)::set(null, path_to_ts_sequencer, "hp_int_asserted", 1);`

- Test to lower level component communication

  This is the only backward incompatible change if you have duplicated the code while creating your own classes.

  ```
  uvm_config_db#(`SVT_PCIE_TEST_SUITE_CONFIGURATION_TYPE)::set(this,cfg.path_to_pcie_test_
  suite_env, "cfg", this.cfg);
  ```

  The second argument is critical, it should not be blank or plain wild character.

- Sequence to test communication

  This is the only backward incompatible change if you have duplicated the code while creating your own classes.

  ```
  uvm_config_db#(int unsigned)::set(this,{ cfg.path_to_pcie_test_suite_env,
  ".test_env_seqr.*" }, "sequence_length", 4);
  ```

  The second argument is critical, it should not be blank or plain wild character.

- Sequence and callback communication

  This change is required if you have used those callbacks and created your own callback. There are few cases where communication also happens between sequence and the callback. In such cases, configuration handle must pass, so that test suite configuration attribute will be also be visible in the callback. This is the only backward incompatible change if you have duplicated the code while creating your own classes.

  Sequence to callback and callback to sequence communication can be enhanced by adding a new string property to `svt_pcie_test_suite_configuration::unique_string_for_link_instance`. No changes are required here as this value is assigned by the class `pcie_unified_base_test` during its build. The string variable is accessed via macro `` `SVT_PCIE_TEST_SUITE_LINK_STRING `` although it could have been accessed via `link_cfg. unique_string_for_link_instance`. This was done to simplify future improvisation.

All the sequences have access to `svt_pcie_test_suite_configuration` instance. Now, callback instances are also provided with access to corresponding `svt_pcie_test_suite_configuration` instance while doing the callback registration.

- Change to test case: Added the following line to file *ts.gen3_pl_invalid_sync_header_on_partial_lane_recovery_idle_error-ep.sv*:

```
rc_sym_cb.link_cfg = cfg;
```

- Changes to sequence and callback

```
svt_pcie_dl_bit_corruption_in_dllp_callback::
void'(uvm_config_db#(bit)::get(null, `SVT_PCIE_TEST_SUITE_LINK_STRING,
"corrupt_updatefc",corrupt_updatefc));
svt_pcie_ts_device_system_virtual_exception_dl_bit_corruption_in_dllp_sequence
::      uvm_config_db#(bit)::set(null, `SVT_PCIE_TEST_SUITE_LINK_STRING,
"corrupt_updatefc",1'b1);
svt_pcie_ts_device_system_virtual_exception_dl_bit_corruption_in_dllp_sequence
::      uvm_config_db#(bit)::set(null, `SVT_PCIE_TEST_SUITE_LINK_STRING,
"corrupt_updatefc",1'b0);
```

- Error demotion logic and its callback registrations

This is the only backward incompatible change if you have duplicated the code while creating your own test cases.

The test cases consisted of callback registrations which were registered using null handle, this caused them to affect all link instances. Most of the callbacks registered using null handle were demotions. Also, many of these demotions are part of the `build_phase`. The code placement remains same in `build_phase` but the actual callback registration via macro `SVT_PCIE_TEST_SUITE_ADD_RECURSIVE_DEMOTION` is done during `connect_phase`.

Example:

Change from	to
`uvm_report_cb::add(null,err_catcher);`	`` `SVT_PCIE_TEST_SUITE_ADD_RECURSIVE_DEMOTION(`SVT_PCIE_TEST_SUITE_SNPS_ENV,err_catcher); ``
`uvm_report_cb::delete(null,err_catcher);`	`` `SVT_PCIE_TEST_SUITE_DEL_RECURSIVE_DEMOTION(`SVT_PCIE_TEST_SUITE_SNPS_ENV,err_catcher); ``

A.9.2 Backward Compatible change

- Do not use disable `<label_name>`

This is a backward compatible change (only if you have not duplicated the code in the class extensions that you have created for sequence override) as the scope is restricted to sequences which are left largely untouched by the users.

Existing sequence collection has been purged of any code using disable `label_name`.

- Following macro pair is used to replace any existing block of code with named `fork/join_any` followed by disable `label_name`:

```
`SVT_PCIE_TEST_SUITE_FORK_JOIN_ANY_START(label_name)
`SVT_PCIE_TEST_SUITE_FORK_JOIN_ANY_FINISH(label_name)
```

- Following macro pair is used to replace any existing block of code with named `fork/join` followed by disable `label_name`:

```
`SVT_PCIE_TEST_SUITE_FORK_JOIN_ALL_START(label_name)
`SVT_PCIE_TEST_SUITE_FORK_JOIN_ALL_FINISH(label_name)
```

- Any disable `named_block` statement has been replaced with

```
`SVT_PCIE_TEST_SUITE_FORK_TERMINATE(label_name).
```

B

Testbench (tb_dut_pcie) Environment Changes and New Use Model

This chapter discusses the following topics:

B.1 Top Level Changes

- All the three base tests (`pcie_ep_dut_base_test`, `pcie_rc_dut_base_test`, and `pcie_phy_dut_base_test`) have been consolidated in a single base test (`pcie_unified_base_test`).
  - The following files have only a new function:
    - *pcie_rc_dut_base_test.sv*
    - *pcie_ep_dut_base_test.sv*
    - *pcie_phy_dut_base_test.sv*
  - Added a new class for PHY `pcie_phy_dut_cust_base_test` for user tb area.
- Changed the order of precedence in configuration settings.
- Corrected some `uvm_config_db` calls with correct accessor.
- Callback for error demotion which was earlier getting registered with null is now getting registered per instance.
- Restructured the code in base test.
  - Error demotions have been moved to environment layer.
  - Plusargs related code moved to TS configuration class.
- Added new configuration attributes.
- For PHY testbench, you must extend your test from `pcie_phy_dut_cust_base_test` instead of `pcie_phy_dut_base_test`.

B.2 Backward Incompatible Changes

- Due to changed order of precedence in configuration settings.

  If you are configuring link width through SCR file, then correct use mode must set the following three properties for link width.

  - `pl_cfg.supported_link_width_vector`
  - `pl_cfg.supported_link_width`
  - `pl_cfg. expected_link_width`

  If you are not setting all these three attributes for configuring link width, then the following warning is issued and the configuration will be auto-corrected accordingly.

  ```
  Invalid expected_link_width of " " provided, must be "". We are autocorrecting the configuration in that case.
  ```

- Due to `UVM_CONFIG_DB` Calls Change

  For uvm_config_db calls in the test, for configuration attributes, the following issues are expected:

  You must use the configuration attributes directly.

  - Legacy Code

    ```
    if ((uvm_config_db# (bit)::get (null, "", "enable_enumeration", enable_enumeration)) && enable_enumeration)
    ```

  - Corrected Code

    ```
    if (cfg.enable_enumeration )
    ```

If you have copied similar code in your test, then those tests will not work as expected.

- Error demotion is done inside environment class and is not done globally.

  The `svt_err_catcher` was instantiated inside base test, so it was accessible to extended test. Because they are now moved to environment, they will not be accessible to tests if you have used that in your internal test.

B.3 Code Changes

- Error demotions have been moved to ENV Layer.

Table B-1 describes the place holder file for demotion based on the agent to demote the warning/error.

Table B-1 Files for Error Demotions

Demotion	File Name
`Error/Warning reported by active VIP agent`	*Snps_pcie_dm_vip_env.sv*
`Error/Warning reported by DUT agent`	*Snps_pcie_dm_dut_env.sv*
`Error/Warning reported by DUT agent (SNPS IIP specific)`	*dwc_pcie_dm_dut_env.sv*

- Moved plusargs code to test suite configuration class.
  - `process_global_plusarg_for_backward_compatibility` (`bit [1:0] dut_type`): Top level container to call all plusargs related code based on dut type. Input to this task is `dut_type`:

`svt_pcie_types::DUT_PCIE_PIPE_PHY` for PHY TB

`svt_pcie_types::DUT_PCIE_RC` for RC DUT

`svt_pcie_types::DUT_PCIE_EP` for EP DUT

- `process_ts_specific_plusargs_for_backward_compatibility ()`: Function to process TS configuration attribute specific global plusargs for backward compatibility. This method processes all plusargs that involves setting test suite configuration attributes. You should not invoke this method directly, it must be invoked via method `process_global_plusarg_for_backward_compatibility`, but if needed the implementation can be changed by class extension.

  This method is associated only with test suite configuration attributes which are dependent on plusargs.

Table B-2 Test Suite Configuration Attributes and Plusargs

Test Suite Configuration Attributes	Plusargs	Comments
enable_enumeration	svt_pcie_enable_enumeration	
enable_sriov	svt_pcie_disable_sriov	
phy_layer_error_inj_on_random_lane	svt_pcie_error_random_lane_selection	
enable_backdoor_cfg_updates	svt_pcie_enable_backdoor_cfg_updates	
app_sys_cfg.master_cfg[0].data_width -------- cfg.app_sys_cfg.slave_cfg[0].data_width	svt_pcie_axi_width	Only applicable with AXI interface when SVT_PCIE_TEST_SUITE_EXCLUDE_APP_INTERFACE is not used.

- `process_per_device_specific_plusargs_for_backward_compatibility` (`svt_pcie_device_configuration device_cfg`):

  Function to process VIP configuration attribute (both for DUT as well as link partner VIP) global plusargs for backward compatibility. Input to this is VIP configuration.

  This method is invoked independently once for DUT and once for VIP. You should not invoke this method directly, it must be invoked via method `process_global_plusarg_for_backward_compatibility`, but if needed the implementation can be changed by class extension.

  This method is associated with only VIP configuration attributes.

  - Plusargs

    `svt_pcie_enable_pa`

    `svt_pcie_enable_transaction_logging`

    `svt_pcie_enable_coverage`

    `svt_pcie_link_speed`

    `svt_pcie_target_link_width`

    `svt_pcie_pipe_width`

- `process_both_device_specific_plusargs_for_backward_compatibility` (`svt_pcie_device_configuration` `vip_cfg`, `svt_pcie_device_configuration` `dut_cfg`):

  Function to process global plusargs affecting configuration of both link partners. This method has two inputs, `vip_cfg` and `dut_cfg`.

  You must not invoke this method directly, it must be invoked via method `process_global_plusarg_for_backward_compatibility`, but if needed the implementation can be changed by class extension.

  This method associated with only VIP configuration attributes but based on DUT capabilities

  - Plusargs

    `svt_pcie_enable_polarity_inversion`

    `svt_pcie_ssc_mode, svt_pcie_max_ppm & svt_pcie_min_ppm`

    `svt_pcie_enable_lane_skew`

- Test configuration attributes have been moved to the `set_dut_capabilities` method.
- Added a new TS configuration attribute `rc_on_pipe`.
  - This can be used as a replacement of macro `SVT_PCIE_TEST_SUITE_RC_ON_PIPE`.

B.4 Use Model

- Extended base test use model
  - You can extend your test from any of the following tests based on the type of DUT to be verified:
    - `pcie_phy_dut_base_test`
    - `pcie_ep_dut_base_test`
    - `pcie_rc_dut_base_test`
  - Additionally, you can also extend your base test directly from `pcie_unified_base_test`.
    - This is the recommended approach which allows you to use the same setup for verification of three DUT types (EP/RC/PHY).
- Overriding the configuration attribute.
  - You can either override the configurations in `*cust_base_test` or SCR/txt file, overriding the method.

C

Integrating Synopsys IIP (default configuration) in Test Suite Testbench

Attention Integration steps described in this chapter are recommended for test suite evaluation with IIP (only with default configuration).

C.1 Synopsys IIP (with default configuration) Integration Steps

The following steps are applicable when the DUT is Synopsys-IIP with default configuration and application interface being XALI and Target. These steps are recommended in case of an evaluation of test suite with IIP (default configuration). These steps will be invalid when an IIP configuration deviates from the default configuration.

1. Copy the *vip_pcie_test_suite_svt_<latest_version>.run* file.
2. Set the `DESIGNWARE_HOME`.
3. Install the VIP/TS using *vip_pcie_test_suite_svt_<latest_version>.run* file.
4. Install the *tb_dut_pcie* example:

```
% cd <path_to_extract_example>
% $DESIGNWARE_HOME/bin/dw_vip_setup -path ./design_dir -e
     pcie_test_suite_svt/tb_dut_pcie -svtb
% setenv DESIGN_DIR <path_to_extract_example>/design_dir
```

5. Install the IIP using coreConsultant using `coreConsultant` based on the recommended steps for IIP installation.
6. Create a symbolic link for IIP *src* directory inside the testbench area:

```
% cd $DESIGN_DIR/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie
% setenv IIP_PATH < path where IIP is installed using Cc>
% ln -s $IIP_PATH/src src
```

7. Create a symbolic link for IIP *sim* directory inside the testbench area:

```
% cd $DESIGN_DIR/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie
% ln -s $IIP_PATH/sim sim
```

8. Create a symbolic link for IIP *examples* directory inside the testbench area:

```
% cd $DESIGN_DIR/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie
% ln -s $IIP_PATH/examples examples
```

9. Create a symbolic link for the below file inside the testbench area:

```
% cd $DESIGN_DIR/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie
% ln -s sim/testbench/common/modules/pipe_test_mux.v pipe_test_mux.v
```

10. Get the IIP `file_list` inside the `tb_dut_pcie` area:

```
% grep -v "\-undef" $IIP_PATH/src/*DWC_pcie_*.lst > iip.f
% grep -v "\-undef" $IIP_PATH/src/Phy/generic/compile.f >> iip.f
% echo "-f sim/testbench/common/customer_sva/compile.f" >> iip.f
```

11. Set the following environment variables:

```
% cd $DESIGN_DIR/examples/sverilog/pcie_test_suite_svt/tb_dut_pcie
% setenv PUE_HOME env/dwc_pcie_dut_bfm/pue
% setenv UVM_EXT_HOME $PUE_HOME/uvm_ext
```

12. Valid combination of Topology/compile files for EP controller as IIP with serdes interface.

    Topology file is:

    SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE= topology_dut_snps_pcie_ep_rc_pue_iip.svi

    Compile file is:

    SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE= compile_dut_snps_pcie_ep_pue_iip_serial_reg.f

13. Run one of the testcases from example using the following command-line options:

```
% gmake USE_SIMULATOR=vcsvlog demo_tl_gen1_linkup_followed_by_tlps-ep
SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_ep_pue_iip_serial_reg.f
SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_pue_iip.svi VERBOSE=1
WAVES=fsdb SVT_PCIE_ENABLE_TRANSACTION_LOGGING=1 UVM_VERBOSITY=UVM_HIGH
```

D

Implementing Partition Compile in Testbench

This chapter discusses the following topics:

D.1 Introduction

To effectively use partition compile, PCIe Test Suite creates multiple sub-packages of the sequence collection file, `snps_pcie_test_suite_seq_collection.svi`. The created multiple sub packages are compiled in parallel thereby improving the overall compilation time. The improvement in the compilation time depends on the number of cores selected for parallel execution. PCIe Test Suite sub-packages are mainly based on PCIe protocol layers like Transport, Data Link and Physical Layers containing sequence collection files related to the layers.

The partition compile feature is disabled by default. To enable the partition compile mode, you must perform the following functions:

- Include the file, `import_pcie_test_suite_pkgs.svi` to import all individual sub-packages.
- Set the compile macro, `SVT_PCIE_TEST_SUITE_OPTIMIZED_COMPILE` in the compilation command.

D.2 High Level Architecture

The PCIe Test Suite creates multiple sub-packages for parallel compilation under existing PCIe Test Suite Package, `snps_pcie_test_suite_pkg`. Following is the list of multiple sub-packages required to enable parallel compilation:

- `snps_pcie_test_suite_common_pkg`
- `snps_pcie_test_suite_tl_pkg`
- `snps_pcie_test_suite_dl_pkg`
- `snps_pcie_test_suite_pl_pkg`

D.3 Use Model

You must perform these steps to enable the parallel partition compilation flow with SVT PCIe Test Suite:

1. Provide the VCS compile argument:

   ```
   +define+SVT_PCIE_TEST_SUITE_OPTIMIZED_COMPILE
   ```

2. Enable the partition compile flow using VCS partition compile options:

   ```
   -partcomp -fastpartcomp=j<N> +optconfigfile+pc.optcfg
   ```

3. Import all the packages or sub-packages at user testbench Top file. It is recommended to include methodology specific import file, say for example,

   ```
   `include "import_pcie_test_suite_pkgs.svi"
   ```

Use auto partition if you have less number of cores. When the number of available cores is less than 4, use the auto partitioning—that is, do not provide the `pc.optcfg` file with new packages because the partition compile on less cores with many partition degrades the compilation performance.

D.4 Running Examples

You must perform these steps to run the tests present in the `tests` directory in the example in the VCS two-step flow with Optimized compile:

1. Update `vcs_build_options` by including the optimized compile macros in the file

   ```
   +define+SVT_PCIE_TEST_SUITE_OPTIMIZED_COMPILE
   ```

2. Run the following command:

   ```
   gmake tl_directed_traffic-ep
   USE_SIMULATOR=vcspcvlog
   SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=compile_dut_snps_pcie_ep_vip_serial.f
   SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=topology_dut_snps_pcie_ep_rc_vip.f
   ```

D.5 Guidelines for OPTIMIZED Compile Users

The PCIe Test Suite support for partition compile feature is mostly backward compatible with the default Test Suite flow. Following are the scenarios where changes may be required to support partition compile flow:

1. **Possible conflict can occur from typedefs.**

   There are chances that the class which is declared as `typedef` gets included before `typedef` declaration. This can occur because of change in compilation order in parallel partition compile.

To avoid such errors, following changes can be made:

For example:

- Without optimized compile arch changes

```
typedef class
svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence;
```

- With optimized compile arch changes

```
`ifndef SVT_PCIE_TEST_SUITE_OPTIMIZED_COMPILE
typedef class
svt_pcie_ts_device_system_virtual_tasks_to_be_overriden_by_user_sequence;
`enidf
```