Verification Continuum™

VC Verification IP
CXL Subsystem
Device DUT/ Host DUT
Integration Guide

Version S-2021.06, June 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 http://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 CXL, and is for design or verification engineers who want to verify CXL Subsystem operation using a UVM testbench written in SystemVerilog.

Web Resources

- Documentation through SolvNet: 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:

1. Go to https://solvnetplus.synopsys.com and open a case.

   Enter the information according to your environment and your issue.
2. Send an e-mail message to support_center@synopsys.com.

   Include the Product name, Sub Product name, and Tool Version in your e-mail so it can be routed correctly.
3. Telephone your local support center.
   - North America:

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

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

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

1.1 Overview

This guide describes the steps to integrate a CXL Host/Device DUT with CXL Subsystem VIP as Device/Host (in PIPE spec v5.1, SerDes Arch, or Serial mode). This helps you to achieve the following objectives:

- You can connect the DUT and VIP using either macro-based or interface-based approach.

Synopsys recommends macro-based approach.

- You can configure the VIP and get the CXL link up by running the sanity test.

This guide discusses the following topics:

- Host/Device DUT Integration in Serial/PIPE5 Mode
- Understanding Compile and Topology Files

1.2 Documentation References

These documents can be referred for more information related to the Synopsys CXL Subsystem VIP product:

- CXL Subsystem VIP HTML Class reference:
  $DESIGNWARE_HOME/vip/svt/cxl_subsystem_svt/latest/doc/cxl_subsystem_svt_uvm_public/html/index.html
- CXL Subsystem VIP User Guide:
  $DESIGNWARE_HOME/vip/svt/cxl_subsystem_svt/latest/doc/cxl_subsystem_svt_uvm_user_guide.pdf

2
Host/Device DUT Integration in Serial/PIPE5 Mode

This chapter describes the consolidated steps to integrate a CXL Host/Device DUT in PIPE5 or Serial mode (Topology #1).

Serial topology:

- Assess the DUT Tx jitter characteristics (for example @Gen5 - 32GT/s, the low period is 31.0ps or 31.5ps instead of 31.25ps). To handle such excessive jitter situations setup the VIP receive clock recovery to use ADJUST_RX_CLK_MODE = 4

PIPE topology:

- Identify the PIPE version and operational mode. For example, PIPE 4.4.1 and PHY clock as input. Also assess the rate/width combination to ensure that DUT is not using custom signaling setup. This is important to make sure that the VIP use model is inter-operable with DUT features.

This chapter discusses the following topics:

Pre-integration Checklist

You must ensure the following conditions are met before doing the VIP - DUT integration:

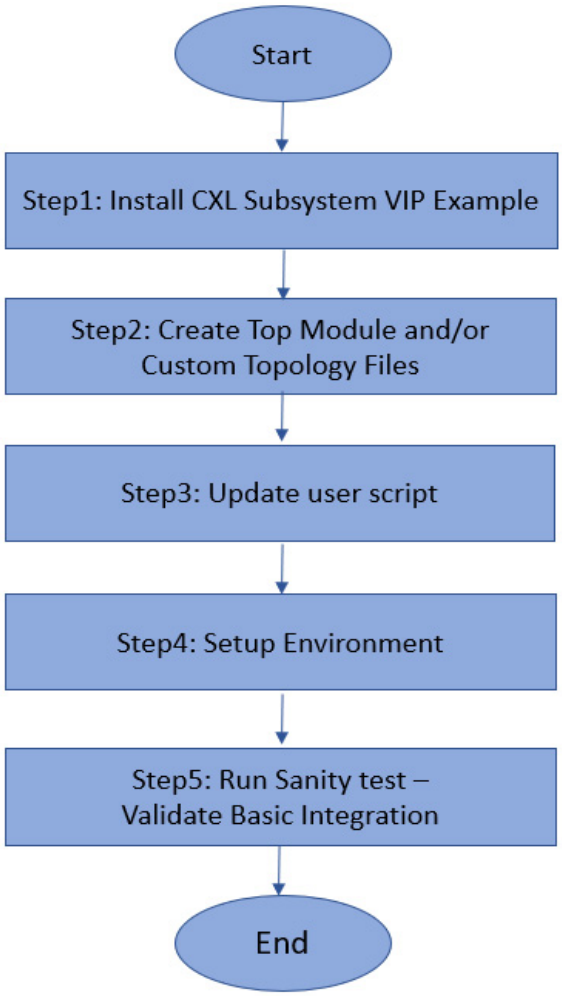
- Downloaded the .run file and extracted the CXL Subsystem VIP example.
- All the required license settings are properly done for using the CXL Subsystem VIP
  - To make sure that example is extracted and the required license settings are done, user can run any test case in VIP back to back mode.
  - If you face any issue, please contact Synopsys VIP Support
- You must also make sure that CXL Subsystem VIP is configured to align with the DUT settings
  - For example, if DUT supports 8 lanes, you must configure the VIP to use the 8 lanes. The following PL configuration function can be used for setting the link width value:

```
cust_cfg.host_cfg.cxl_io_cfg[0].pcie_cfg.pl_cfg.set_link_width_values(8);
```

2.1 Integrating the CXL Subsystem VIP Testbench Components with a DUT

You must perform these steps to integrate a DUT with the VIP:

Integration Steps Flow Chart:



Overview of the Steps:

Step 1: Install CXL Subsystem VIP Example - You can refer the CXL Subsystem User Guide for the steps for installing the CXL Subsystem VIP example. The command to run basic example is also mentioned.

Note You can skip this step if you have already installed the VIP example and ran the simulation in VIP back to back setup.

Step 2: Create Top Module and/or Custom Topology Files (Used for VIP-DUT Connection) - This step talks about the required packages to be included in the Customer top file. The step by step flow of connecting VIP

with DUT (i.e. Integration Mechanism) is provided by taking serial interface as a reference. This is followed by customer topology files (with DUT connections) for Serial and PIPE5 connections are provided for reference

Step 3: Update User Script - You can make the required changes that needs to be done in customer script.

Step 4: Setup Environment - You can then create or update the customer Environment in order to use the VIP

Step 5:Run Sanity Test - Validate Basic Integration - Here the command for running Sanity test case is provided. Also, the list of available test cases in VIP example area are mentioned.

2.1.1 Install CXL Subsystem VIP Example

You must install and run the example in the VIP back to back setup (assuming you have downloaded the CXL Subsystem VIP model and set up all licensing and access to the VCS simulator).

For installing the VIP and for extracting the example, refer to *Installing the Product* and *Installation and Running examples* sections in the *VC VIP CXL Subsystem UVM User Guide*.

2.1.1.1 Running Sanity Test in VIP Back to Back Setup

Run VIP back to back example to get familiar with the compilation steps, file/package inclusions and various simulation logs.

The following command can be used to run the test - `ts.cxl_io_random_mem_wr_rd.sv` for CXL.io traffic generation with FULL_STACK topology over serial interface.

Commands to run:

```
gmake cxl_io_random_mem_wr_rd USE_SIMULATOR=vcsvlog
SVT_CXL_SUBSYSTEM_COMPILE_FILE=compile_snps_vip_pcie_serial.f
SVT_CXL_SUBSYSTEM_TOPOLOGY_FILE=topology_snps_vip_cxl_b2b.svi
```

For more information on running the test case, refer to the README file in the example. The README contains the description of the example.

Table 2-1 Simulator Options

Flow	Use_SIMULATOR option
VCS two step flow	vcsvlog

Ensure that you maintain back to back compile and simulation log ready. These can be used to match your compile log and simulation log while integrating VIP with DUT.

2.1.2 Create Top Module and/or Custom Topology Files (Used for VIP-DUT Connection)

Before going through this step on how to create Custom top module and Topology file, to get an overview of the available topology and compile files, go through the Understanding Compile and Topology Files files chapter in this guide.

In this section, you would be using the base topology file - `topology_snps_vip_cxl_b2b.svi` as a reference to create the custom topology file.

2.1.2.1 Creating Custom Top Module

Include and import all the necessary CXL Subsystem VIP and UVM Packages in the top file of the user.

For reference, go through the top.sv file available in the installed CXL Subsystem VIP example.

The following snippet from the top.sv file lists the necessary major hooks. However, the entire `SVT_CXL_SUBSYSTEM_SNPS_TEST_TOP` (i.e. test_top) module is required.

```
`timescale 1 ns/1 fs
`include "svt_cxl_subsystem_user_defines.svi"

`ifdef SVT_UVM_TECHNOLOGY
`include "svt_cxl.uvm.pkg"
`include "svt_cxl_subsystem.uvm.pkg"
`elsif SVT_OVM_TECHNOLOGY
`include "svt_cxl.ovm.pkg"
`include "svt_cxl_subsystem.ovm.pkg"
`endif
`include "svt_cxl_subsystem_ts_env.pkg"

`include "cust_pre_tb_top.svi"

module `SVT_CXL_SUBSYSTEM_SNPS_TEST_TOP;

timeunit `SVT_CXL_SUBSYSTEM_TIMEUNIT_VALUE;
timeprecision `SVT_CXL_SUBSYSTEM_TIMEPRECISION_VALUE;

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

  `include "import_cxl_subsystem_uvm_pkgs.svi"
`elsif SVT_OVM_TECHNOLOGY
  /** Import OVM Package */
  import ovm_pkg::*;
  `include "ovm_macros.svh"

  `include "import_cxl_subsystem_ovm_pkgs.svi"
`endif

  // Import CXL Subsystem TS ENV Package.
  import svt_cxl_subsystem_ts_env_pkg::*;
```

Note Reset to VIP model is active high. Reset must be asserted at time 0 such that a posedge is seen by the VIP. For VIP, reset should be asserted at least for 100ns to configure its internal registers and buffers. As shown in the top.sv file of VIP back to back example, add reset in the user top file.

```
// This is the interface of first VIP instance acting as Host/Device for Device/Host DUT
  bit reset;
// By default the reset to interfaces is driven low using pull down
  tri0 common_pwr_on_reset = reset ? 1'b1 : 1'bz;
  initial begin
//Drive reset
    reset = 1;
    #200ns;
```

```
  reset = 0;
end
```

The hdl_interconnect_macro.sv file needs to be included in the "Top File" of the user right after including the CXL Subsystem VIP and UVM packages.

```
`include "hdl_interconnect_macros.sv"/* Contains Helper macros used in topology files*/
```

Note

Integration Step

You can copy the hdl_interconnect_macros.sv and topology_snps_vip_cxl_b2b.svi (one of the topology files from VIP example area, VIP(full stack) back to back topology with serial/PIPE5 connection) files from the installed example location to your location which makes these files to get compiled.

Attention

First, just check if you can compile the VIP in your environment (Before connecting DUT and VIP signals)

For description about various topology and compile files available in CXL Subsystem VIP example area, refer to the Overview of Topology and Compile Files Available in the CXL Subsystem VIP Example Area.

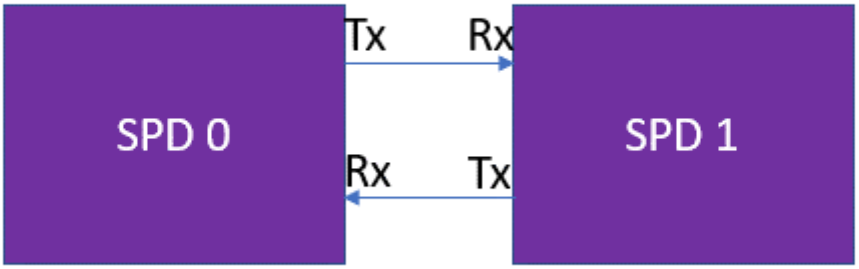
2.1.2.2 Creating Customer Topology File (Used for VIP-DUT Connection)

You must first understand the following VIP-DUT integration plan.

In VIP back to back setup, two VIP instances (spd_0 and spd_1) are created and connected as shown below.

You must note that spd0 and spd1 (Single Port Device1 and 2) are the module names of the VIP instances created. The VIP instance is the Verilog instance of PCIe/CXL VIP not the agent level instance.

Figure 2-1 VIP Instances Back to Back

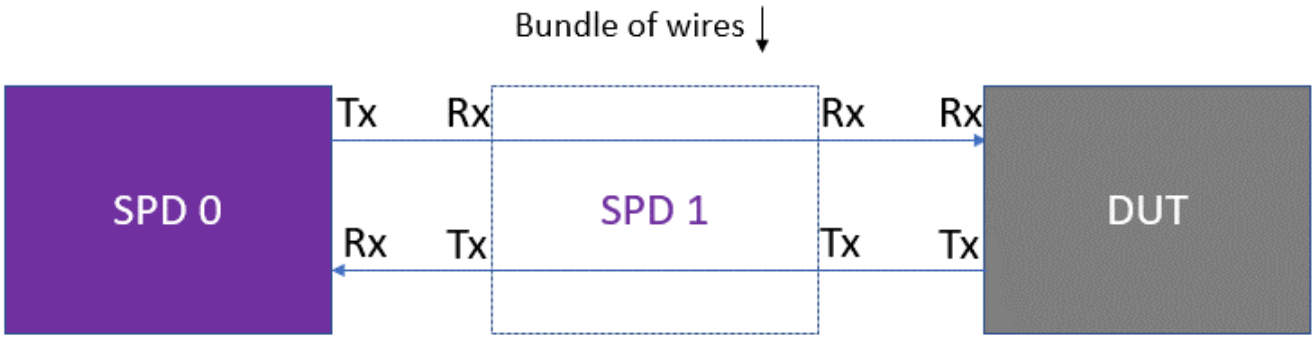


When the VIP is integrated with the DUT, the functionality of any one of the VIP instance is turned off (by setting PHY Interface type as 'APP') and make it a bundle of wires.

Then, you need to connect the Tx of DUT with SPD1's (the VIP instance which you are making as a bundle of wires) Tx and Rx of DUT with SPD1's Rx. Hence, Tx of SPD0 would be connected to DUT Rx and Rx of SPD0 will be connected to Tx of DUT. So, the data will simply pass through the SPD1 instance of VIP.

This is a mirror based approach. i.e. Tx of SPD1 (VIP #2) is same as Tx of DUT and Rx of SPD1 (VIP #2) is same as Rx of DUT.

Figure 2-2 VIP DUT Connection with One of the VIP instances (spd_1) Acting as 'Bundle of Wires'



2.1.2.2.1 VIP Instantiation in a Topology File:

You can follow one of the two methods for connecting VIP with DUT.

1. Interface based approach
   a. Creating instances of PCIe VIP HDL agent and its associated interface and forming the link using two instances of the interface is done directly in the file without the use of helper macros.
2. Macro based approach (Recommended):
   a. Macros for connecting the VIP instance are defined in the hdl_interconnect_macros.sv file
   b. You can either use the macros as it is or can change the definition in the above file. Macros ease the usage by wrapping the entire process in one command.

   Example - SVT_PCIE_ICM_SER_SER_LINK is used for creating the serial link connections and SVT_PCIE_ICM_PIPE_PIPE_LINK is used for the PIPE connections. For more information, see the description in hdl_interconnect_macros.sv file.

You can see the creation of 2 VIP instances and the connection between them.

Note You can move to the next 'integration step' (DUT Connections in a Topology File) and skip these steps and to use the topology file provided in the example area for VIP back to back connection which is created using Macro based approach.

The following code is to create 2 VIP instances and to connect them back to back. You can skip these points and use the `topology_snps_vip_cxl_b2b.svi` file present in VIP example area as mentioned in above note.

- Create two VIP instances i.e. spd_0 and spd_1.

```
`SVT_PCIE_ICM_CREATE_PORT_INST(0,0)
```

```
`SVT_PCIE_ICM_CREATE_PORT_INST(0,1)
```

- Create the link between them and do the dependent signal inter-connections using these macros:

```
`SVT_PCIE_ICM_CREATE_LINK(0,spd_0,spd_1) // Create link 0 using ports
spd_0,spd_1.
`SVT_PCIE_ICM_DO_CONDITIONAL_INTERCONNECT(0,spd_0,1,spd_1) // Certain signal
inter-connections are dependent on VIP parameter settings. This macro handles
those connections and simplified the topology file
```

- Compile-time settings of first VIP instance:

```
defparam spd_0.SVT_PCIE_UI_DISPLAY_NAME = " spd_0.";
defparam spd_0.SVT_PCIE_UI_ENABLE_CFG_BLOCK= 0;
defparam spd_0.SVT_PCIE_UI_PCIE_SPEC_VER =`SVT_PCIE_UI_PCIE_SPEC_VER_5_0;
defparam spd_0.SVT_PCIE_UI_PIPE_SPEC_VER =`SVT_PCIE_UI_PIPE_SPEC_VER_5_1;
defparam spd_0.SVT_PCIE_UI_NUM_PHYSICAL_LANES= 1;
defparam spd_0.SVT_PCIE_UI_HIERARCHY_NUMBER= 0;
defparam spd_0.SVT_PCIE_UI_MPIPE= 0;
`ifdef SERDES_TB
defparam spd_0.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES;
`else
defparam spd_0.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE;
`endif
defparam spd_0.SVT_PCIE_UI_DEVICE_IS_ROOT= 1; // for Device DUT, VIP is acts
as Host (RC).

defparam spd_0.SVT_PCIE_UI_ENABLE_SHADOW_MEMORY_CHECKING= 0;
defparam spd_0.SVT_PCIE_UI_CONNECT_ACTIVE_VIP= 1;
defparam spd_0.SVT_PCIE_UI_PIPE_CLK_FROM_MAC= 0;
defparam spd_0.SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR = 0;
defparam spd_0.SVT_PCIE_UI_CONNECT_DOWNSTREAM_PORT_MONITOR = 0;
```

- Compile-time settings of second VIP Instance:

```
defparam spd_1.SVT_PCIE_UI_DISPLAY_NAME = "spd_1.";
defparam spd_1.SVT_PCIE_UI_ENABLE_CFG_BLOCK= 0;
defparam spd_1.SVT_PCIE_UI_PCIE_SPEC_VER =`SVT_PCIE_UI_PCIE_SPEC_VER_5_0;
defparam spd_1.SVT_PCIE_UI_PIPE_SPEC_VER =`SVT_PCIE_UI_PIPE_SPEC_VER_5_1;
defparam spd_1.SVT_PCIE_UI_NUM_PHYSICAL_LANES= 1;
defparam spd_1.SVT_PCIE_UI_HIERARCHY_NUMBER= 0;
defparam spd_1.SVT_PCIE_UI_MPIPE= 1;
`ifdef SERDES_TB
defparam spd_1.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES;
`else
defparam spd_1.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_PIPE;
`endif

defparam spd_1.SVT_PCIE_UI_DEVICE_IS_ROOT = 0;
defparam spd_1.SVT_PCIE_UI_ENABLE_SHADOW_MEMORY_CHECKING= 0;
defparam spd_1.SVT_PCIE_UI_CONNECT_ACTIVE_VIP= 1;
defparam spd_1.SVT_PCIE_UI_PIPE_CLK_FROM_MAC= 0;
defparam spd_1.SVT_PCIE_UI_CONNECT_UPSTREAM_PORT_MONITOR = 0;
defparam spd_1.SVT_PCIE_UI_CONNECT_DOWNSTREAM_PORT_MONITOR = 0;
```

- Map the parameter values to interface variables by invoking update_if_variables(...) for each VIP instance.

```
initial begin
spd_0.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
                                          targeted.
                           "", // UVM class instance to which the Passive VIP
                               interface will be targeted. Not used for now.
                           1, // Bit to enabled 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.
                          );


spd_1.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
                                         targeted.
                          "", // UVM class instance to which the Passive VIP
                                interface will be targeted. Not used for now.
                          1, // Bit to enabled 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.
                          );
 end
```

- Cross-connect signals of one interface instance to another interface instance using the macro.

```
//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, spd_0,spd_1)
//OR
//cross connect PCIe PIPE5 interfaces of VIP instances to form PCIe PIPE5 link

// Macro assumes the arguments as link_number, vip instance representing phy
side i.e. MPIPE=0 , vip instance representing Controller (MPIPE=1)
`SVT_PCIE_ICM_PIPE5_PIPE5_LINK(0, spd_0, spd_1)
```

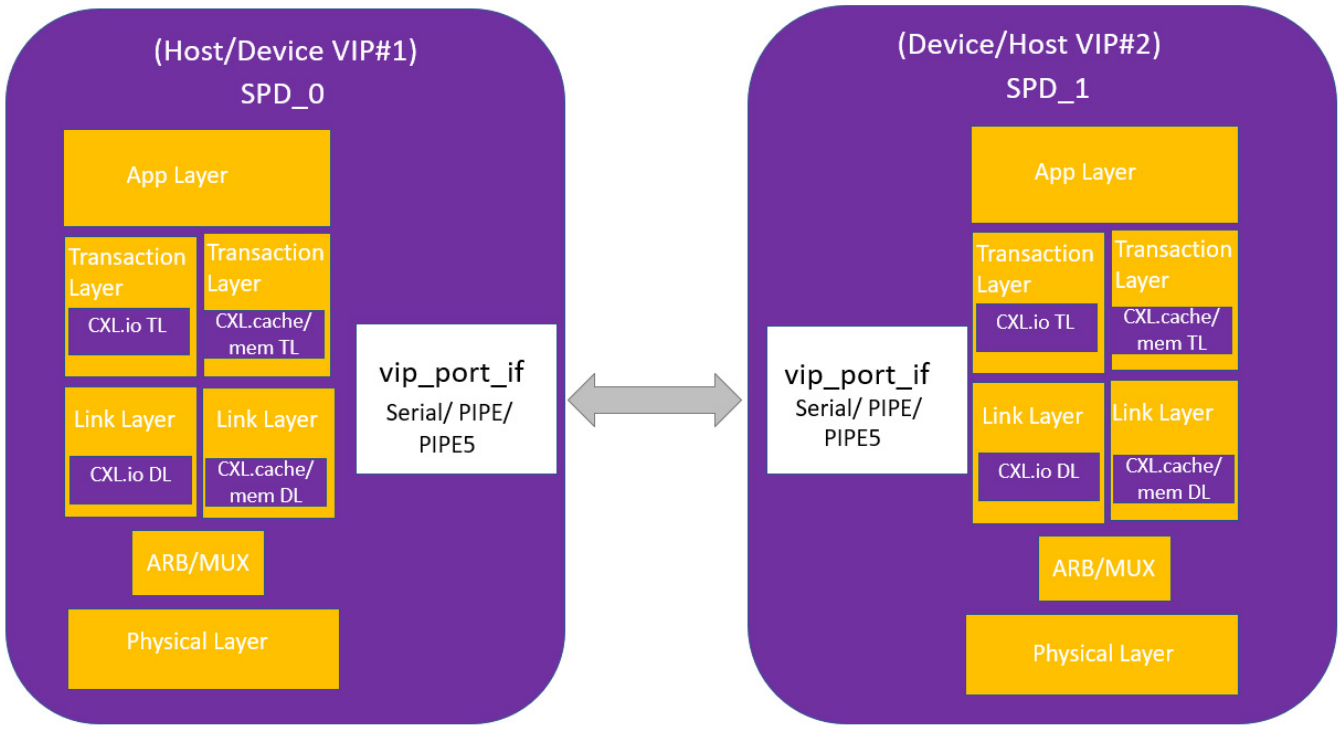
2.1.2.2.2 DUT Connections in a Topology File

Copy the files `hdl_interconnect_macros.sv` and `topology_snps_vip_cxl_b2b.svi` from the VIP example area to user test bench area to avoid creating the VIP back to back connection.

Note Rename the copied topology with another suitable name, for example topology_snps_vip_cxl_dut.svi to avoid overriding of existing files when extracting the example from another/latest version of VIP.

These files copied to User test bench area would ensure to instantiate the two instances of dual-mode VIP supporting various CXL signal interconnects. This VIP back to back integration is done with Macro based approach.

Figure 2-3 Host and Device VIP Interfaces Cross-connected



Note Integration step: Deactivate the TL, DL, and PL stack of Device/Host VIP#2 (VIP acting as a DUT) instance by setting the interface type to `SVT_PCIE_UI_PHY_INTERFACE_TYPE_APP.

```
// Application VIP connection and configuration and DUT connection
defparam SVT_PCIE_TEST_SUITE_PHY_INTERFACE_TYPE_P1 =
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_APP;
```

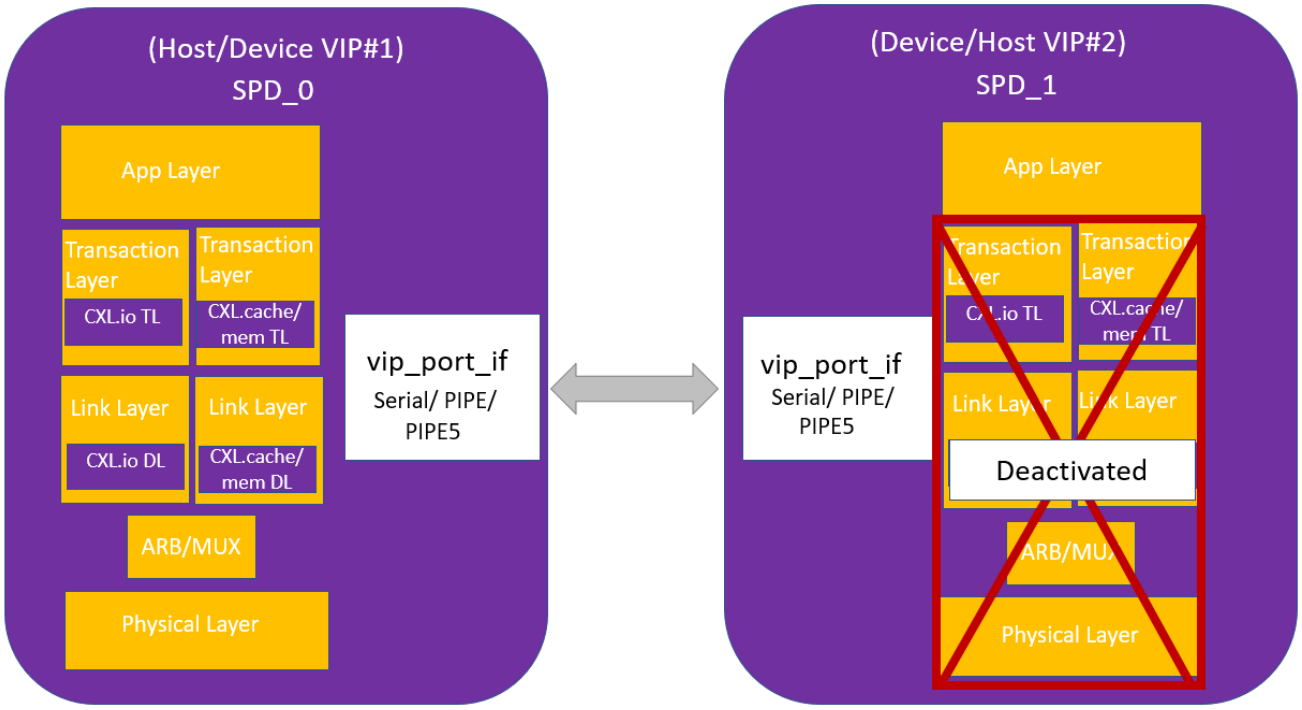
Note When using Serial interface, add the following configuration inside topology file to handle excessive jitter situations at VIP receive clock. This needs to be set for all the lanes.

```
<vip_instance>.m_ser.port0.SER_GEN_<0/1/2…>.serdes.ADJUST_RX_CLK_MODE = 4;
```

For example, set the configuration for Lane1 of spd_0 like the following:

```
spd_0.m_ser.port0.SER_GEN_1.serdes.ADJUST_RX_CLK_MODE = 4;
```

Figure 2-4 VIP Instance #2 Converted as Application VIP



Setting `SVT_PCIE_UI_PHY_INTERFACE_TYPE_P1` to `` `SVT_PCIE_UI_PHY_INTERFACE_TYPE_APP `` will disable the functionality of SPD 1 VIP instance.

The definition of the macros present in the topology file can be found in hdl_interconnect_macros.sv file. With this setting, the TL, DL, and PL stack of the VIP#2 that is acting as a DUT (spd_1) would be disabled.

Make sure to drive the reset and appl_clk to default values. i.e. 0

With these changes, the VIP #2 will just act as bundle of wires. i.e. the data will pass through the VIP #2

Set clock and reset of Application VIP to default values.

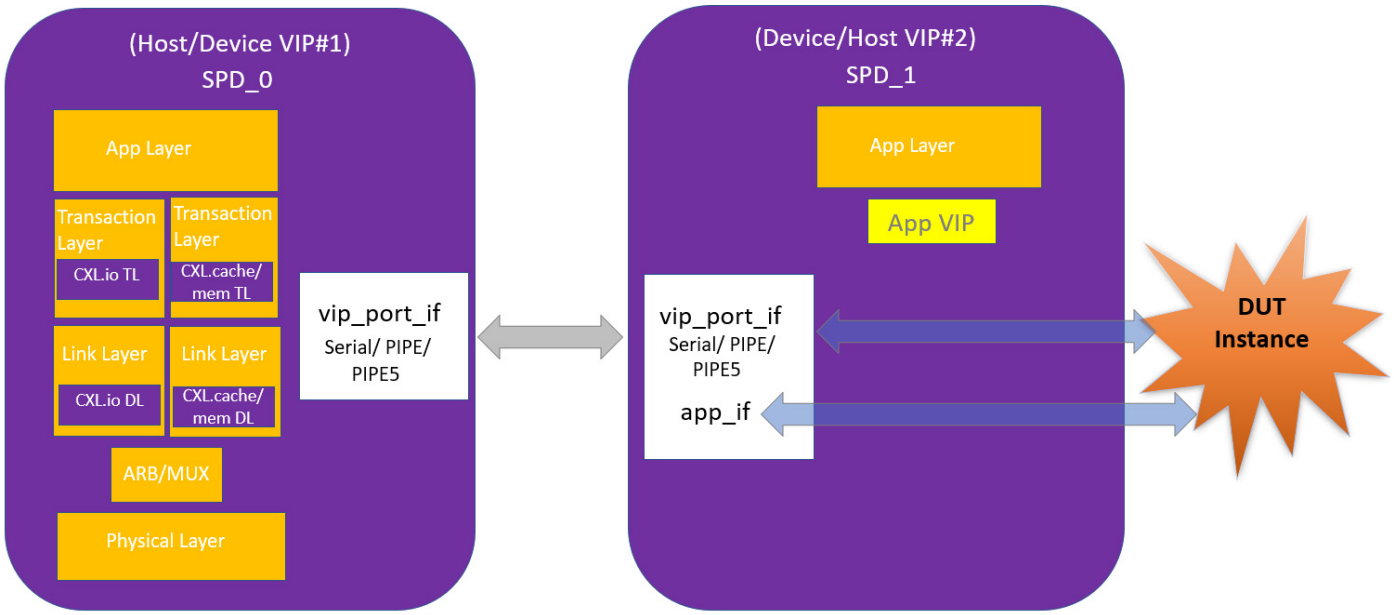
```
// Set clock and reset of the VIP whose application layer is active but TL,DL
and PL stack is INACTIVE

always @(*)spd_1.vip_port_if.app_if.reset = 0;

always @(*) spd_1.app_if.appl_clk = 0;
```

The Host/Device VIP#1 interface (ser/pipe_if) is connected to VIP#2 Interface (ser/pipe_if) which in turn gets connected to DUT as shown in the code below. This makes the VIP#2 interface connection just a pass-through (see Figure5).

Note

This connection is equivalent to Device/Host DUT directly connected to Host/Device VIP.
The above Macro based approach ensures the proper connection between RC and EP interface which are defined in macros provided in `hdl_interconnect.macros.sv` file.
Additionally, you can run any test in demo (VIP-VIP) mode without modifying the connection.

Figure 2-5 VIP DUT Connection



Note

The two VIP instances are connected back to back (i.e. Tx signals of VIP #1 with Rx signals of VIP #2 and vice versa). This Tx-Rx cross-connection is done using the macro (`SVT_PCIE_ICM_SER_SER_LINK`/`SVT_PCIE_ICM_PIPE_PIPE_LINK`/ `SVT_PCIE_ICM_PIPE5_PIPE5_LINK`)
Now, the VIP instance #1 will be connected to DUT through "VIP instance #2". For this the Tx signals of VIP #2 interface (spd_1.vip_port_if) must be connected to Tx signals of the DUT. Similarly, Rx signals of the VIP #2 interface (spd_1.vip_port_if) must be connected to Rx signals of the DUT.

Here, the VIP instance #2 interface will act as mirror of DUT interface signals. This means that Tx of VIP #2 is same as Tx of DUT and Rx of VIP #2 is same as Rx of DUT.

Connect the Device/Host DUT to Host/Device VIP (#1) through spd_1 instance (VIP instance #2) of VIP

The following code is presented for reference to specify how the VIP instance #2 (which is acting as bundle of wires) is connected to DUT instance.

```
`ifdef SERDES_TB
// 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
always @(*) `CUSTOMER_DUT_INSTANCE.rx_datap_0 =
spd_1.vip_port_if.ser_if.rx_datap_0 ;
```

```
always @(*) `CUSTOMER_DUT_INSTANCE.rx_datan_0 =
spd_1.vip_port_if.ser_if.rx_datan_0 ;

always @(*)spd_1.vip_port_if.ser_if.tx_datap_0 =
`CUSTOMER_DUT_INSTANCE.tx_datap_0 ;
               always @(*)spd_1.vip_port_if.ser_if.tx_datan_0 =
`CUSTOMER_DUT_INSTANCE.tx_datan_0 ;
`else
// PIPE Interconnect code irrespective of number of lanes
always @(*)`CUSTOMER_DUT_INSTANCE.pclk =spd_1.vip_port_if.pipe_if.pclk;
always @(*)`CUSTOMER_DUT_INSTANCE.max_pclk =
spd_1.vip_port_if.pipe_if.max_pclk;
always @(*)`CUSTOMER_DUT_INSTANCE.reset = common_pwr_on_reset;
always @(*)spd_1.vip_port_if.pipe_if.rate= `CUSTOMER_DUT_INSTANCE.rate;
always @(*)spd_1.vip_port_if.pipe_if.pclk_rate=
`CUSTOMER_DUT_INSTANCE.pclk_rate;
. . . . . .
. . . . . .
// per lane PIPE interconnect code to be replicated as per DUT requirement in
terms of number of lanes
always @(*)`CUSTOMER_DUT_INSTANCE.rx_data_0 =
spd_1.vip_port_if.pipe_if.rx_data_0 ;
always @(*)`CUSTOMER_DUT_INSTANCE.rx_data_k_0 =
spd_1.vip_port_if.pipe_if.rx_data_k_0 ;
always @(*)`CUSTOMER_DUT_INSTANCE.rx_status_0 =
spd_1.vip_port_if.pipe_if.rx_status_0 ;
always @(*)`CUSTOMER_DUT_INSTANCE.rx_valid_0 =
spd_1.vip_port_if.pipe_if.rx_valid_0 ;
. . . . . .
. . . . . .

always @(*) spd_1.vip_port_if.pipe_if.rx_polarity_0=
`CUSTOMER_DUT_INSTANCE.rx_polarity_0 ;
     always @(*) spd_1.vip_port_if.pipe_if.tx_data_0=
   `CUSTOMER_DUT_INSTANCE.tx_data_0 ;
   always @(*) spd_1.vip_port_if.pipe_if.tx_data_k_0=
   `CUSTOMER_DUT_INSTANCE.tx_data_k_0 ;
   always @(*) spd_1.vip_port_if.pipe_if.tx_ei_code_0=
   `CUSTOMER_DUT_INSTANCE.tx_ei_code_0 ;
   always @(*) spd_1.vip_port_if.pipe_if.tx_compliance_0=
   `CUSTOMER_DUT_INSTANCE.tx_compliance_0 ;
   always @(*) spd_1.vip_port_if.pipe_if.tx_elec_idle_0=
   `CUSTOMER_DUT_INSTANCE.tx_elec_idle_0 ;
   always @(*) spd_1.vip_port_if.pipe_if.tx_data_valid_0=
   `CUSTOMER_DUT_INSTANCE.tx_data_valid_0 ;
. . . . . .
. . . . . .
```

2.1.2.3 Reference Customer Topology File for Serial Connection

```
/****************** DUT Connections for reference ******************/
// Supply clock and reset to the VIP whose application layer is active but
TL,DL and PL stack is INACTIVE
              always @(*) spd_1.vip_port_if.app_if.reset =
common_pwr_on_reset; // input to VIP instance to reset the
         app layer which is driving the DUT TL,DL,PL stack
     always @(*) spd_1.vip_port_if.app_if.appl_clk = u_pcie_device.core_clk;
//input to VIP instance for app layer which is driving the DUT TL,DL,PL stack.
reg [7:0] rxp;
reg [7:0] rxn;
always @(*)  rxp[7]   = spd_1.vip_port_if.ser_if.rx_datap_7;
always @(*)  rxn[7]   = spd_1.vip_port_if.ser_if.rx_datan_7;
always @(*)  rxp[6]   = spd_1.vip_port_if.ser_if.rx_datap_6;
always @(*)  rxn[6]   = spd_1.vip_port_if.ser_if.rx_datan_6;
always @(*)  rxp[5]   = spd_1.vip_port_if.ser_if.rx_datap_5;
always @(*)  rxn[5]   = spd_1.vip_port_if.ser_if.rx_datan_5;
always @(*)  rxp[4]   = spd_1.vip_port_if.ser_if.rx_datap_4;
always @(*)  rxn[4]   = spd_1.vip_port_if.ser_if.rx_datan_4;
always @(*)  rxp[3]   = spd_1.vip_port_if.ser_if.rx_datap_3;
always @(*)  rxn[3]   = spd_1.vip_port_if.ser_if.rx_datan_3;
always @(*)  rxp[2]   = spd_1.vip_port_if.ser_if.rx_datap_2;
always @(*)  rxn[2]   = spd_1.vip_port_if.ser_if.rx_datan_2;
always @(*)  rxp[1]   = spd_1.vip_port_if.ser_if.rx_datap_1;
always @(*)  rxn[1]   = spd_1.vip_port_if.ser_if.rx_datan_1;
always @(*)  rxp[0]   = spd_1.vip_port_if.ser_if.rx_datap_0;
always @(*)  rxn[0]   = spd_1.vip_port_if.ser_if.rx_datan_0;

 always @(*)  spd_1.vip_port_if.ser_if.tx_datap_3        = DUT.txp[3] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datap_2        = DUT.txp[2] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datap_1        = DUT.txp[1] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datap_0        = DUT.txp[0] ;

 always @(*)  spd_1.vip_port_if.ser_if.tx_datan_3        = DUT.txn[3] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datan_2        = DUT.txn[2] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datan_1        = DUT.txn[1] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datan_0        = DUT.txn[0] ;

 always @(*)  spd_1.vip_port_if.ser_if.tx_datap_7        = DUT.txp[7] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datap_6        = DUT.txp[6] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datap_5        = DUT.txp[5] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datap_4        = DUT.txp[4] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datan_7        = DUT.txn[7] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datan_6        = DUT.txn[6] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datan_5        = DUT.txn[5] ;
 always @(*)  spd_1.vip_port_if.ser_if.tx_datan_4        = DUT.txn[4] ;

USER_DUT DUT(
  .refclk_p                    (refclk_p),
  .refclk_n                    (refclk_n),
  .txp                         (txp),
  .txn                         (txn),
  .rxp                         (rxp[7:0]),
  .rxn                         (rxn[7:0]),
)

      /****************** DUT Connections for reference ******************/
```

2.1.2.4 Reference Customer Topology File for PIPE5 (SerDes Arch. only) Connection

```
/****************** DUT Connections for reference ******************/
// Supply clock and reset to the VIP whose application layer is active but
TL,DL and PL stack is INACTIVE

always @(*) spd_1.vip_port_if.app_if.reset = common_pwr_on_reset; // input to
VIP instance to reset the app layer which is driving the DUT TL,DL,PL stack
always @(*) spd_1.vip_port_if.app_if.appl_clk = u_pcie_device.core_clk;
//input to VIP instance for app layer which is driving the DUT TL,DL,PL stack.


reg max_pclk;
reg reset;
...
reg [31:0] rx_data;
reg [31:0] rx_data_k;
....

// PIPE Interconnect code irrespective of number of lanes
always @(*)max_pclk = spd_1.vip_port_if.pipe5_if.max_pclk;
always @(*)reset = common_pwr_on_reset;
always @(*)spd_1.vip_port_if.pipe5_if.serdes_arch= serdes_arch;
always @(*)spd_1.vip_port_if.pipe5_if.reset = common_pwr_on_reset;


// per lane PIPE interconnect code to be replicated as per DUT requirement in
terms of number of lanes
always @(*)rx_data[0] = spd_1.vip_port_if.pipe5_if.rx_data_0 ;
always @(*)rx_data_k[0] = spd_1.vip_port_if.pipe5_if.rx_data_k_0 ;
always @(*)rx_status[0] = spd_1.vip_port_if.pipe5_if.rx_status_0 ;
....
....
//Note: These are some of the signals of per lane PIPE interconnect code.
Please refer to the Marco-SVT_PCIE_ICM_PIPE5_PIPE5_PER_LANE_CODE description
in the - hdl_interconnect_macros.sv file for the information about remaining
signals needs to be connected.

USER_DUT DUT(
  .refclk_p                  (refclk_p),
  .refclk_n                  (refclk_n),
  .rx_data                   (rx_data[31:0]),
  .rx_data_k                 (rx_data_k[31:0]),
  .....
      )/****************** DUT Connections for reference
******************/
```

Note The example provided here for Serial topology and reference topology files are given for Serial/PIPE5 interfaces. You are requested to take these topology files as a reference and do the changes based on your requirement.

2.1.3 Update User Script

In this section, the script changes are mentioned for VCS flow for 2-step flow.

Note Contact Synopsys if you want to use VCS 3-step flow or partition compile flow

You can see the script file changes for VCS 2-step flow for reference.

- Include the necessary directories in the script to make them visible

  You can refer to the back to back compile log (generated under logs directory when an example test case is run in VIP back to back mode) to check the incdir needs to be handpicked and add them in the your script.

Mandatory files to be included:

design_dir has two major folders - src and include. These files are mandatory to be included in the user's script (include in same sequence):

```
+incdir+<design_dir>/src/sverilog/vcs \
+incdir+<design_dir>/include/sverilog \
+incdir+<design_dir>/src/verilog/vcs \
+incdir+<design_dir>/include/verilog \
```

Add -y option (include this also in same sequence):

```
+libext+.v+.sv -y  <design_dir>/src/verilog/vcs \
-y  <design_dir>/src/sverilog/vcs \
```

Here is the list of defines need to add based on requirement.

Define	Value/Requirement
UVM_DISABLE_AUTO_ITEM_RECORDING	Optional
UVM_PACKER_MAX_BYTES	1500000
SVT_MEM_SA_STATUS_RD_B4_WR	Use if required
SVT_MEM_SA_STATUS_RD_RD_NO_WR	Use if required

Compile defines to be added in script

```
+define+SVT_MEM_SA_STATUS_RD_B4_WR=0
+define+SVT_MEM_SA_STATUS_RD_RD_NO_WR=0
+define+UVM_PACKER_MAX_BYTES=1500000
+define+UVM_DISABLE_AUTO_ITEM_RECORDING
```

Note You can copy these compile options and can add them in the script file of the user.

```
vcs

-l ./logs/compile.log
```

```
-ntb_opts uvm
-full64
-sverilog

+define+SVT_MEM_SA_STATUS_RD_B4_WR=0
+define+SVT_MEM_SA_STATUS_RD_RD_NO_WR=0
+define+UVM_PACKER_MAX_BYTES=1500000
+define+UVM_VERDI_NO_COMPWAVE
+define+UVM_DISABLE_AUTO_ITEM_RECORDING
+define+SVT_UVM_TECHNOLOGY
+define+SYNOPSYS_SV

-timescale=1ns/1fs
+incdir+$DESIGNWARE_HOME/design_dir/src/sverilog/vcs \
+incdir+$DESIGNWARE_HOME/design_dir/include/sverilog \
+incdir+$DESIGNWARE_HOME/design_dir/src/verilog/vcs \
+incdir+$DESIGNWARE_HOME/design_dir/include/verilog \

+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/env/seq_and_cb \
+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/env/seq_and_cb/pcie_ts_seq_and_cb \
+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/env/pcie_env \
+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/env/pcie_include \

+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/../../env \
+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/../env \
+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/env \
+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/dut \
+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/hdl_interconnect \
+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/lib \
+incdir+$DESIGNWARE_HOME/design_dir/examples/sverilog/cxl_subsystem_svt/tb_cxl
_subsystem_uvm_basic_sys/tests \
+libext+.v+.sv -y $DESIGNWARE_HOME/design_dir/src/verilog/vcs \
-y $DESIGNWARE_HOME/design_dir/src/sverilog/vcs \
-f svt_cxl_subsystem_compile_file.f \
-f top_files
-f hdl_files
-P pli.tab msglog.o
-debug_acc+pp+dmptf+thread
-debug_region=cell+encrypt
-o ./output/simvcssvlog
Simulation?
./output/simvcssvlog  <options>
```

Note Each file must be compiled in the correct order (in the same order as in back to back example). Reference can always be taken from the compile log generated after running the VIP back to back example. Also, you must check if the same pkgs/files are included/imported correctly or not.

Attention

If you get any "macros not defined" errors or any other errors - Cross check whether the 'DESIGNWARE_HOME' is properly set or not.

If errors are not resolved after setting the DESIGNWARE_HOME, check the generated compile log to make sure that everything is included in the same order. You can compare it with the compile log (generated when running the VIP back to back example)

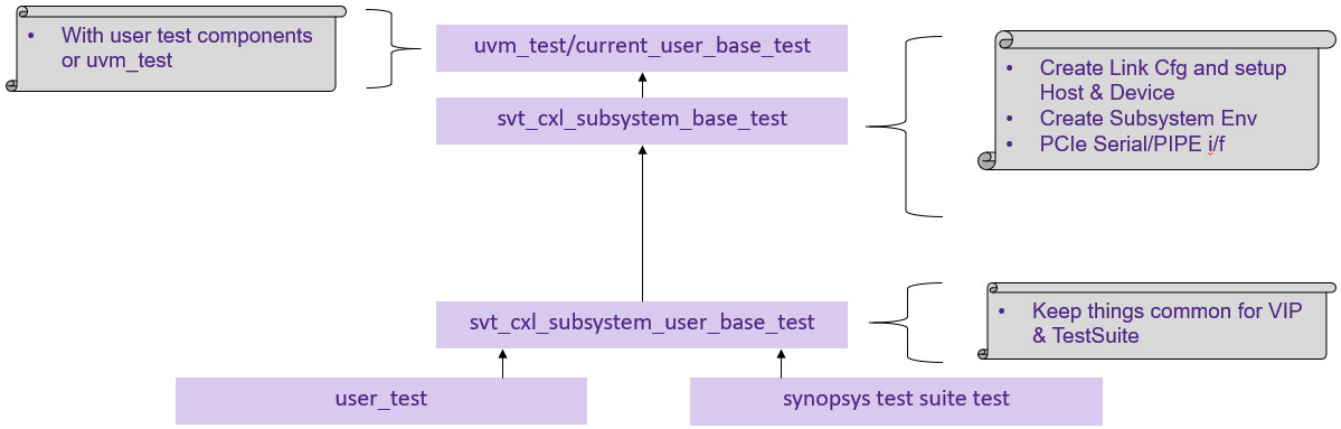
If the errors are still not resolved - compare pkgs and files that are included in user top file with top.sv file in back to back example.

2.1.4 Setup Environment

If everything is set and the compilation is passing, the next step is to set up the 'Environment'. Refer to the base test (env/svt_cxl_subsystem_base_test.sv) and the Environment in back to back example to create/update your own environment.

2.1.4.1 Creating/Updating User-Specific Base Test

To simplify the creation of test case by extending from the base test (svt_cxl_subsystem_user_base_test), which includes all the configurations/settings required for both VIP and DUT, the Synopsys CXL Subsystem VIP has implemented the following hierarchical flow for the base test.



Understanding the above implementation in the VIP Example

You can specify the user-specific base test/ uvm_test as the parent class for the `svt_cxl_subsystem_base_test` by defining `SVT_CXL_SUBSYSTEM_USER_BASE_TEST` macro in `svt_cxl_subsystem_user_defines.svi`.

```
`ifndef SVT_CXL_SUBSYSTEM_USER_BASE_TEST
`ifdef SVT_UVM_TECHNOLOGY
  `define SVT_CXL_SUBSYSTEM_USER_BASE_TEST uvm_test
```

You can find that svt_cxl_subsystem_base_test is extended from the macro - SVT_CXL_SUBSYSTEM_USER_BASE_TEST

```
class svt_cxl_subsystem_base_test extends `SVT_CXL_SUBSYSTEM_USER_BASE_TEST;
```

Objective of this test class- svt_cxl_subsystem_base_test is to do the basic settings related to VIP.

The svt_cxl_subsystem_user_base_test is extended from the svt_cxl_subsystem_base_test

```
class svt_cxl_subsystem_user_base_test extends svt_cxl_subsystem_base_test;
```

Objective of this test class-svt_cxl_subsystem_user_base_test is to keep things common for VIP and TestSuite.

Now, you need to create their own class for any extra addition which is extending with this class - svt_cxl_subsystem_user_base_test.

Also, you need to set SVT_CXL_SUBSYSTEM_TEST_BASE with user test. By default, it is set to svt_cxl_subsystem_user_base_test.

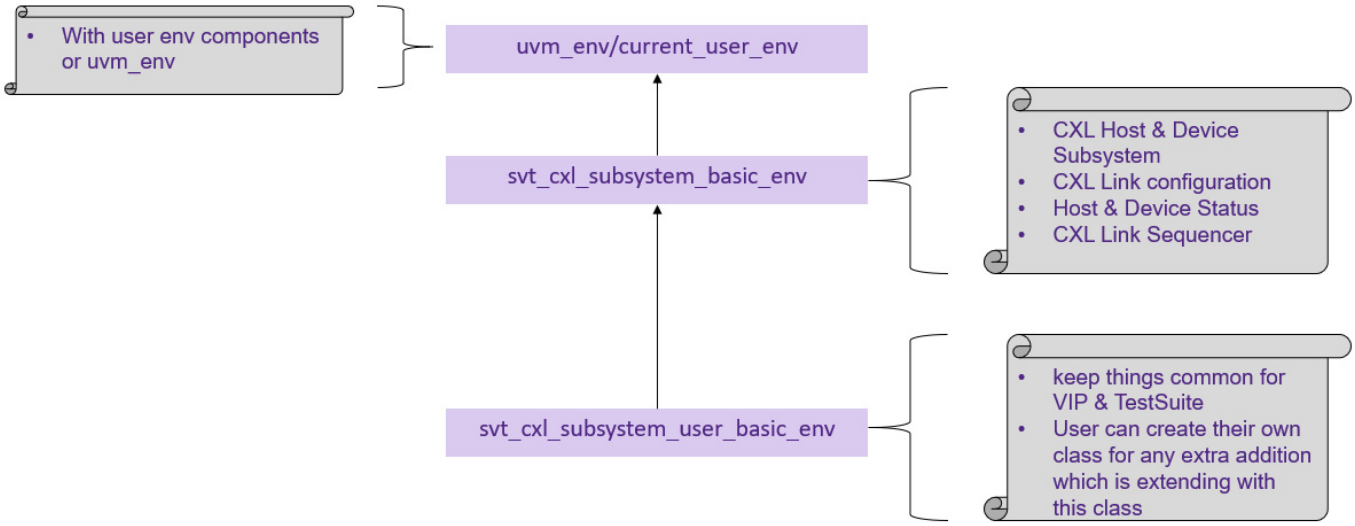
```
`ifndef SVT_CXL_SUBSYSTEM_TEST_BASE
  `define SVT_CXL_SUBSYSTEM_TEST_BASE svt_cxl_subsystem_user_base_test
`endif
```

And all the test cases would be extended from this macro - SVT_CXL_SUBSYSTEM_TEST_BASE.

```
class cxl_io_random_mem_wr_rd extends `SVT_CXL_SUBSYSTEM_TEST_BASE;
```

2.1.4.2 Creating/Updating User-Specific Basic Environment

To simplify the creation of environment by extending from the basic environment (svt_cxl_subsystem_user_basic_env) that includes all the environment settings required for both VIP and DUT, Synopsys CXL Subsystem VIP has implemented the following hierarchical flow for the basic environment.



Understanding the above implementation in the VIP Example

You can specify the user specific basic env / uvm_env as the parent class for the `svt_cxl_subsystem_basic_env` by defining `SVT_CXL_SUBSYSTEM_USER_BASE_ENV` macro in `svt_cxl_subsystem_user_defines.svi`.

```
`ifndef SVT_CXL_SUBSYSTEM_USER_BASE_ENV
`ifdef SVT_UVM_TECHNOLOGY
  `define SVT_CXL_SUBSYSTEM_USER_BASE_ENV uvm_env
```

You can see that `svt_cxl_subsystem_basic_env` is extended from the macro - `SVT_CXL_SUBSYSTEM_USER_BASE_ENV`.

```
class svt_cxl_subsystem_basic_env extends `SVT_CXL_SUBSYSTEM_USER_BASE_ENV;
```

The objective of this environment class- `svt_cxl_subsystem_basic_env` is a top level environment class to encapsulate CXL Subsystem VIP side components.

The `svt_cxl_subsystem_user_basic_env` is extended from the `svt_cxl_subsystem_basic_env`.

```
class svt_cxl_subsystem_user_basic_env extends svt_cxl_subsystem_basic_env;
```

The Objective of this env class-`svt_cxl_subsystem_user_basic_env` is to keep things common for VIP and TestSuite.

Now, you need to create their own class for any extra addition which is extending with this class - svt_cxl_subsystem_user_basic_env. Also, you need to set SVT_CXL_SUBSYSTEM_TEST_ENV with your basic environment. By default, it is set to svt_cxl_subsystem_user_basic_env.

```
`ifndef SVT_CXL_SUBSYSTEM_TEST_ENV
  `define SVT_CXL_SUBSYSTEM_TEST_ENV svt_cxl_subsystem_user_basic_env
`endif
```

You can find the following implementation in svt_cxl_subsystem_base_test to instantiate the environment using the macro.

```
/**
 * Instance of CXL Subsytem native system environment
 */
`SVT_CXL_SUBSYSTEM_TEST_ENV env;
```

This ensures that user basic env which is set as the macro - SVT_CXL_SUBSYSTEM_TEST_ENV (present in env/svt_cxl_subsystem_user_defines.svi) would be instantiated in the base test .

Most of the things (instances and configurations) can be directly copied from the example area and can be used in your environment.

Update the define - SVT_CXL_SUBSYSTEM_SNPS_DUT_TYPE (present in env/svt_cxl_subsystem_user_defines.svi) file from VIP_DUT to GENERIC_IIP_DUT

Or

Add the following define in the compile file:

```
+ define+SVT_CXL_SUBSYSTEM_SNPS_DUT_TYPE=GENERIC_IIP_DUT
```

Note that if you are using Synopsys DWC CXL Controller as the DUT Type, then set this define SVT_CXL_SUBSYSTEM_SNPS_DUT_TYPE to SNPS_IIP_DUT.

Note When using CXL Subsystem VIP as 'Host', configure the subsystem topology type for Device as well (with the same value as Host) to avoid the run time error. The same approach is applicable when using CXL Subsystem VIP as 'Device'.

This can be done using the set_cxl_subsystem_env_device_cfg/set_cxl_subsystem_env_host_cfg methods available in the svt_cxl_subsystem_link_configuration class.

For example, if the CXL Subsystem VIP is being used as Host and the subsystem topology is set to PCIE_IO_FULL_STACK as shown below,

```
cust_cfg.set_cxl_subsystem_env_host_cfg(1,svt_cxl_subsystem_configuration::PCIE_IO_FULL
_STACK);
```

Then the same configuration must be set from the Device side as well as shown here:

```
cust_cfg.set_cxl_subsystem_env_device_cfg(1,svt_cxl_subsystem_configuration::PCIE_IO_FULL_STACK);
```

The intention is to make sure if one side of VIP is set with a certain value then the remote partner side should also be configured with same value.

This is applicable if DUT has full-stack. If a VIP instance is sitting on top of DUT side, then this is not applicable (for example, ARB/MUX + Logical PHY as DUT).

Note If you get any Error while doing the UVM get/set for the interface (pcie_if), match hierarchy using run-time switch +UVM_CONFIG_DB_TRACE. Check `update_if_variables` task in `hdl_interconnect_macros` file.

Example information signature:

UVM_INFO <$USER_PATH>/user_base_test.sv(80) @ 0.00000 ns: uvm_test_top [build_phase] vif_0[0] virtual interface handle(s) is/are null. Hence skipping the remaining build_phase

UVM_FATAL <$USER_PATH>/user_base_test.sv(167) @ 0.00000 ns: uvm_test_top [build_phase] Number of valid links is 0 should be at least 1

2.1.5 Run Sanity Test - Validate Basic Integration

Once the environment settings are done, run the sanity test to validate the integration. Here is the list of tests available in VIP example for validating the integration.

Test Case Name	Test Intention
ts.cxl_io_random_mem_wr_rd.sv	Test to generate CXL.io traffic
ts.cxl_tl_type3_mem_wr_rd.sv	Test to generate Type3 device CXL.mem traffic
ts.cxl_tl_type1_cache_wr_rd.sv	Test to generate Type1 device CXL.cache traffic
ts. cxl_tl_random_mem_wr_rd.sv	Test to generate Type 2 Device random traffic

For more details of installing and running the example, refer to the README file in the example, located at: *<design_dir>/examples/sverilog/cxl_subsystem_svt/tb_cxl_subsystem_uvm_basic_sys/README*

Command to run Sanity test:

For example, the following command can be used to run the test - `ts.cxl_io_random_mem_wr_rd.sv` for CXL.io traffic generation with the user defined topology.

```
gmake cxl_io_random_mem_wr_rd USE_SIMULATOR=vcsvlog SVT_CXL_SUBSYSTEM_COMPILE_FILE=<User define compile file>
SVT_CXL_SUBSYSTEM_TOPOLOGY_FILE=<User defined topology file>
```

You must take care of performing the DUT-specific steps while using the sanity tests. For example, bringing DUT out of reset or doing DUT specific configurations.

Note The above mentioned gmake command is provided assuming that your adopting the make of the VIP example.

If you want to create script for running the VIP example in User Test Bench area, refer to Step3 for all compile options to be added in the user script.

After successfully following the above steps without any errors or warnings, you must be able to see successful link up and the flits (converted into symbols) exchanged between VIP and DUT while running any one of the above tests or the tests developed by user.

Link up Message:

When integrating the VIP - DUT in serial/ PIPE5 mode, you will get the following message from VIP on successful link up:

```
UVM_INFO $USER_PATH/design_dir/src/sverilog/vcs/pciesvc_128b_130b_ltssm_incl.sv(1696) @ 98685924.10 ps: uvm_test_top.env.device_env.io[0].port0.pl0 [report_message]
 LTSSM: Link training completed.  The link is READY! Speed is 32Gb/s. Link width is 16. Link number is 4. Scrambling is ENABLED.
```

This is the basic message to check whether the link up happened properly.

This UVM_INFO gives information about the link speed, width, link number, and whether scrambling is enabled.

3 Understanding Compile and Topology Files

This chapter gives overview about various topology and compile files available in the CXL Subsystem VIP example area and describes various compile defines and HDL interconnect Macros used in the topology and compile files.

3.1 Overview of Topology and Compile Files Available in the CXL Subsystem VIP Example Area

Understand the various topology and compile files available in the VIP example - `tb_cxl_subsystem_uvm_basic_sys` area.

3.1.1 Topology Files

In the VIP back to back example, the top-level testbench (top.sv) file includes the file- `svt_cxl_subsystem_topology_file.svi` generated by prescript by copying the contents of any one of the following topology files based on the switch `SVT_CXL_SUBSYSTEM_TOPOLOGY_FILE =` passed along with the `gmake` command. If no file is passed along with the `gmake` command, the default `topology_snps_vip_cxl_b2b.svi` is picked up.

This file `svt_cxl_subsystem_topology_file.svi` contains the topology specific content which is a soft link of anyone of the following topology files.

Table 3-1 List of Topology Files Present in Back to Back Example

Topology File Name	Description
topology_snps_vip_cxl_b2b.svi	This is the base topology file Creates 2 VIP instances and connects them with Serial or PIPE5 link and CXL.io or CXL.Cache/mem or CXL.$ components will be enabled based on the compile file.
topology_snps_vip_cxl_b2b_lpif.svi	Creates 2 VIP instances and connects them at DL layers of each instance through the LPIF interface. CXL.io or CXL.Cache/mem or CXL.$ components are enabled based on the compile file.
topology_snps_vip_cxl_io_dl_b2b.svi	Creates 2 VIP instances (with CXL.io only) and connects them at the DL Layer level (using TLM Ports) with no PHY layer.

The definitions of Interconnect macros used in the above files are present in the env/hdl_interconnect_macro.sv file.

3.1.2 About hdl_interconnect_macro.sv File

This file contains definitions of macros used to create and connect the VIP module (and interface) instances. The macros are defined for all topologies and connection types. The usage of these macros will ease the integration process.

3.1.3 Compile Files

In VIP back to back example, the `vcs_build_options` file includes the compile file - `svt_cxl_subsystem_compile_file.f`.

This is generated by the prescript by copying the contents of any one of the following compile files based on the switch `SVT_CXL_SUBSYSTEM_COMPILE_FILE =` passed along with the `gmake` command. If no file is passed along with the `gmake` command, the default - `compile_snps_vip_pcie_serial.f` will be picked up.

This file - `svt_cxl_subsystem_compile_file.f` contains the compile options (based on which the topology will be created) which is a soft link of anyone of the following compile files.

Table 3-2 List of Compile Files Present in back to back Example

Compile File Name	Description
compile_snps_vip_pcie_serial.f	Contains the compile-time options for creating a connection with serial interface with all componets enabled i.e. CXL.$
compile_snps_vip_io_lpif.f	Contains the compile-time options for creating the topology with only CXL.io component enabled and the connection at LPIF.
compile_snps_vip_cache_mem_only_dl_tl_lpif.f	Contains the compile-time options for creating the topology with only Cache/mem components enabled and the connection at LPIF.
compile_snps_vip_cxl_cache_mem.f	Contains the compile-time options for creating the topology with only Cache/mem components enabled and connection at DL layer through TLM ports.
compile_snps_vip_cxl_io_only_pcie_pipe.f	Contains the compile-time options for creating the topology with only CXL.io component enabled and the connection with PIPE interface
compile_snps_vip_cxl_io_only_pcie_serial.f	Contains the compile-time options for creating the topology with only CXL.io component enabled and the connection with PIPE interface
compile_snps_vip_cxl_cache_mem_tl.f	Contains the compile-time options for creating the topology with only Cache/mem components enabled and connection at TL layer through TLM ports
compile_snps_vip_cxl_io_tl_dl_only.f	Contains the compile-time options for creating the topology with only CXL.io component enabled and connection at DL layer through TLM ports

3.1.4 Desciption About Various Compile Defines and HDL Interconnect Macros

You can choose the compile options based on the following description for creating the desired topology.

Note Some of the defines are containing the PCIe naming conventions as in CXL Subsystem SVT, PCIe VIP is being used as CXL.io

Table 3-3 Description of Compile Defines

Compile Define	Description	Used for Interface Type
SVT_CXL_SUBSYSTEM_PCIE_IF_TOPOLOGY	Enables connection at PCIe interface	Serial/ PIPE/ PIPE5
SVT_PCIE_TEST_SUITE_SERDES_TB	Enables connection at PCIe Serial topology	Serial
SVT_PCIE_TEST_SUITE_PIPE_SPEC_VER_5_1	To select the PIPE Spec. version 5.1	PIPE5
SVT_PCIE_TEST_SUITE_PIPE_SPEC_VER_4_2 _PCLK_INPUT	To select the PIPE Spec. version 4.2 with PCLK as PHY input	PIPE
SVT_PCIE_ENABLE_SERDES_ARCH	To enable SERDES Architecture mode	PIPE
SVT_PCIE_TEST_SUITE_ENABLE_GEN5	To enable Gen5 Support	Serial/ PIPE5
SVT_PCIE_TEST_SUITE_VIP_DUT	To use one of the VIP instance as DUT. This will defined when running in VIP back to back setup. Must not be used when running with actual DUT.	Serial/PIPE / PIPE5/LPIF
SVT_PCIE_TEST_SUITE_EP_IS_DUT	To set the Device VIP instance as DUT when running in VIP back to back setup	Serial/PIPE/PIPE5/ LPIF
SVT_PCIE_TEST_SUITE_EXCLUDE_APP_ INTERFACE	To bypass the App interface	Serial/PIPE/PIPE5/ LPIF
SVT_PCIE_ENABLE_PIPE5	To enabled PIPE5 support	PIPE5
SVT_CXL_SUBSYSTEM_CXL_IO_ONLY	Disables CXL.Cache/Mem agent creation	Serial/PIPE5
SVT_CXL_LPIF_LCLK_FREQ_NS	To set LPIF clock (lclk) frequency	LPIF
SVT_ENABLE_XCHECK	To enable XCHECKER	LPIF
SVT_CXL_SUBSYSTEM_CACHE_MEM_LPIF_ TOPOLOGY	To enable the connection at b2b Cache/Mem at DL via LPIF	LPIF
SVT_CXL_SUBSYSTEM_PHY_LPIF_OVER_ DEVICE_SUBSYSTEM	To configure the Device to drive the PL signal	LPIF
SVT_LPIF_DEASSERT_PL_TRDY_ INTERMITTENTLY	To De-assert PL TRDY signal intermittently	LPIF

Compile Define	Description	Used for Interface Type
SVT_CXL_SUBSYSTEM_IO_LPIF_TOPOLOGY	To enable the connection at VIP b2b IO at DL via LPIF	LPIF
SVT_CXL_SUBSYSTEM_TOP_LAYER_DL	To set the top layer as Data Link Layer	--
SVT_CXL_SUBSYSTEM_CXL_CACHE_MEM_TOPOLOGY	To enable the VIP Back2Back connection at DL	--
SVT_CXL_SUBSYSTEM_CXL_CACHE_MEM_TL_TOPOLOGY	To enable the VIP Back2Back connection at TL	--

Here is the description of HDL Interconnect macros used in the topology files (available in the VIP example area) for creating VIP instances and connecting the ports and so on.

Table 3-4 Description of HDL Interconnect Macros

Macro	Description
SVT_PCIE_ICM_CREATE_PORT_INST (link_num, port_num)	To Create port/VIP instances and its associated interface object. Also, set the default parameter settings of the VIP instance. **Note**: These settings can be modified further based on the usage
SVT_PCIE_ICM_CREATE_LINK(link_id, spd_a, spd_b)	To form a link between 2 VIP instances using the interface objects. This link can be uniquely identified using the link_id argument.
SVT_PCIE_ICM_DO_CONDITIONAL_INTERCONNECT(spd_a_port_num, spd_a, spd_b_port_num, spd_b)	To do certain signals inter-connections which are dependent on VIP parameter settings.
SVT_PCIE_ICM_SER_SER_LINK	To cross-connect signals of one serdes interface instance with another serdes interface instance.
SVT_PCIE_ICM_PIPE_PIPE_LINK	To cross-connect signals of SPIPE interface with signals of MPIPE interface to use PIPE interface
SVT_PCIE_ICM_PIPE5_PIPE5_LINK	To cross-connect signals of SPIPE interface with signals of MPIPE interface to use PIPE5 interface
SVT_LPIF_IF_SIGNAL_CONNECT_1_0	To cross-connect signals (for LPIF 1.0) of one LPIF interface instance with another LPIF interface instance
SVT_LPIF_IF_SIGNAL_CONNECT_1_1	To cross-connect signals (for LPIF 1.1) of one LPIF interface instance with anoher LPIF interface instance

3.1.5 Understanding VIP Back to Back LPIF Topology File in the CXL Subsystem VIP Example

You can see how the LPIF topology file is created for VIP back to back setup.

You can refer to the topology file `topology_snps_vip_cxl_b2b_lpif_reference.svi`. In this topology file, both IO and Cache.mem LPIF are instantiated in a single subsystem.

1. Instantiated svt_cxl_if for both the VIPs

```
//Instantiating cxl_if for host/device
  svt_cxl_if lpif_host_if();
 svt_cxl_if lpif_device_if();
```

2. Assigned lclk,reset,tb_clk to each interface

```
assign lpif_host_if.lclk  = local_clk;
   assign lpif_host_if.reset = `SVT_CXL_SUBSYSTEM_SNPS_TEST_TOP.reset;
   assign lpif_host_if.tb_clk  = local_clk; // For now, keeping both lclk and
tb_clk same. Updae if separate sources are required for these.
   assign lpif_device_if.lclk  = local_clk;
   assign lpif_device_if.reset = `SVT_CXL_SUBSYSTEM_SNPS_TEST_TOP.reset;
   assign lpif_device_if.tb_clk  = local_clk; // For now, keeping both lclk
and tb_clk same. Updae if separate sources are required for these.
```

3. Set interface for host/device via set_config_db

```
svt_config_vif_db#(virtual svt_cxl_if)::set(null,  "*_test_top.env.host_env",
"vif",  lpif_host_if);
svt_config_vif_db#(virtual svt_cxl_if)::set(null,  "*_test_top.env.device_env"
,  "vif",  lpif_device_if);
```

4. Fed the signals among interfaces through the following macro present in hdl_interconnect_macros.sv file

```
`SVT_LPIF_IF_SIGNAL_CONNECT_1_0(pl_if_inst,lp_if_inst) - Used to connect LPIF
1.0 signals
`SVT_LPIF_IF_SIGNAL_CONNECT_1_1(pl_if_inst,lp_if_inst) - Used to connect LPIF
1.1 signals
```

   First input instance must be the interface for subsystem over which PHY LPIF is hooked.

   Second input instance should be interface for subsystem over which LINK LPIF is hooked.

   This analogy must be followed for correct connections.

   First set interface handles for IO LPIF components, and then Cache/Mem

```
//Conncetion for IO lPIF
`SVT_LPIF_IF_SIGNAL_CONNECT_1_0(lpif_device_if.lpif_if[0],lpif_host_if.lpif_if
[0])
`SVT_LPIF_IF_SIGNAL_CONNECT_1_1(lpif_device_if.lpif_if[0],lpif_host_if.lpif_if
[0])
 //Conncetion for cache.mem lPIF
`SVT_LPIF_IF_SIGNAL_CONNECT_1_0(lpif_device_if.lpif_if[1],lpif_host_if.lpif_if
[1])
`SVT_LPIF_IF_SIGNAL_CONNECT_1_1(lpif_device_if.lpif_if[1],lpif_host_if.lpif_if
[1])
```