Verification Continuum™

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

Version S-2021.06, June 2021

SYNOPSYS®

Contents

Preface

About This Document

This guide provides you the information on how to integrate a RC/EP DUT in PIPE or Serial mode for VC VIP PCIe Test Suite.

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.
- ❖ 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.
- ❖ 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 RC/EP DUT Integration

This guide describes the steps to integrate a RC/EP DUT in PIPE or Serial mode to achieve the following objectives:

- To connect the DUT and VIP using macro and interface based approach.
- Configure VIP and get the link up by running sanity test `demo_pl_gen1_linkup_test- ep/rc`.

This chapter discusses the following topics:

- Integrating the Test Suite Testbench Components with a DUT
- Validating the Integration

1.1 Integrating the Test Suite Testbench Components with a DUT

Perform the following steps to integrate a DUT with VIP:

- Step 1: Top Module
- Step 2: Create Custom Topology Files (Used for VIP-DUT Connection)
- Step 3: Basic Application BFM Integration
- Step 4: Create a DUT ENV Wrapper
- Step 5: Create a Base Test by Extending from pcie_unified_base_test
- Step 6: Create a DUT Compile File List

Step 1: Top Module

This step is applicable for all the following cases:

Case 1: User *top* module instantiated in the test suite *top* module.

Case 2: Test suite *top* module instantiated in the user *top* module.

Case 3: Directly using the test suite *top* module for integrating the DUT.

The following snippet from the *top.sv* file lists the necessary major hooks. However, the entire `test_top` module is required.

```
`include "cust_pre_tb_top.svi"
`include "snps_pcie_test_suite.pkg"
// This file further include "cust_pcie_test_suite_pkg_files.svi"
module test_top();
import snps_pcie_test_suite_pkg::* ;
`include "svt_pcie_test_suite_topology_file.svi"
endmodule
```

Attention Any filename that starts with "*cust_**" will remain unchanged in the future releases. You must save a local copy of such files and make sure that your file is selected in the compile flow so that your changes are not overridden when the *tb* directory is upgraded with the new version of the test suite.

- Update the following files for Case 1 and Case 3.
  - *cust_pre_tb_top.svi*: This file is provided as a hook so that this can be populated to add DUT-specific defines or definitions of the modules to be used (DUT itself/other VIPs and so on). These files can be modified by the users.

    Few possible inclusions are:
    - DUT-related files, macros, constants, and so on
    - Clock reset module
    - Application interface (*axi.if*, *apb.if*, and so on)
    - Application VIP packages (*axi.uvm.pkg*, *appl.vip.uvm.pkg*, and so on)

      If you have done all the required inclusions in your *top* testbench, then this file does not require any changes. Existing test suite class packages will then be imported via the *snps_pcie_test_suite.pkg* hook. If the elements of user-defined packages are used in the *top*, then you must add import calls in the topology file.
  - *cust_pcie_test_suite_pkg_files.svi*: This file is provided as a hook so that you can populate this file to
    - Import the existing packages in user setup to this file (optional).
    - Add third-party DUT-specific classes as part of package `snps_pcie_test_suite`.
    - Include extended user env and base test files.

      If you have done all the required inclusions in your *top* testbench, then this file does not require any changes.
  - *svt_pcie_test_suite_topology_file*: This file includes VIP instantiation and connection to the DUT. If you have already done the connection in your connection file, then you can create a symbolic link of your DUT connection file with the name *svt_pcie_test_suite_topology_file.svi,* otherwise create a topology (connection) file as described in Step 2.

    For Case 2, you can either instantiate the test suite `test_top` module in your existing *top* module (*tb_top.sv*) or create your *top* file and/or copy all the contents of the test suite *top.sv* file to your *top* file.
- Override SNPS `test_top` module.

This step is applicable when the test suite *top* module is directly instantiated in the user *top* module instead of copying its necessary contents.

- ✦ Rename SNPS `test_top` module: You can override SNPS `test_top` module with any name by overriding macro `SVT_PCIE_TEST_SUITE_SNPS_TEST_TOP`. By default, the value of this macro is `test_top`.
- ✦ To invoke SNPS `test_top` as a child module in your `test_top,` you can invoke `run_phase` from SNPS *top* module with the `SVT_PCIE_TEST_SUITE_INVOKE_RUN_PHASE` parameter. The default value of parameter is 1, which implies that the `run_phase` is invoked from SNPS *top* module.

Step 2: Create Custom Topology Files (Used for VIP-DUT Connection)

Reference: *topology_dut_cust_controller.svi*

Note The *topology_dut_cust_controller.svi* file is an example which shows the steps for connections and integrations. You can use this file if you do not want to use macros to make the connections. If you want to use the macros method for the connections, then you can start with the *topology_dut_snps_pcie_ep_rc_vip.svi* file.

This file includes the following:

- ✦ SVT PCIe Unified VIP (module and interface) instances
- ✦ DUT instance (may or may not be part of the topology file)
- ✦ PCIe signal connections between VIP and DUT instance

❖ With `gmake/run_dut_pcie` option

The *svt_pcie_test_suite_topology_file.svi* file is created at runtime and is not part of the installation example. When you invoke the simulation using `gmake` command, `tb_dut_pcie/Makefile` creates a symbolic link named *svt_pcie_test_suite_topology_file.svi* pointing to the filename passed via switch `SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE`.

Usage:

```
    gmake USE_SIMULATOR=<vcsvlog> \
    <testname: class to be instantiated using 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=<tests/*.scr>
```

Example:

```
    gmake USE_SIMULATOR=vcsvlog  demo_pl_gen1_linkup_test-ep/rc\
    SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE=<dut_specific_compile_file.f> \
    SVT_PCIE_TEST_SUITE_TOPOLOGY_FILE=<dut_specific_topology.svi>
```

❖ Without `gmake/run_dut_pcie` option

You must create this file in your local directory and create a link (symbolic link) with *svt_pcie_test_suite_topology_file.svi* file.

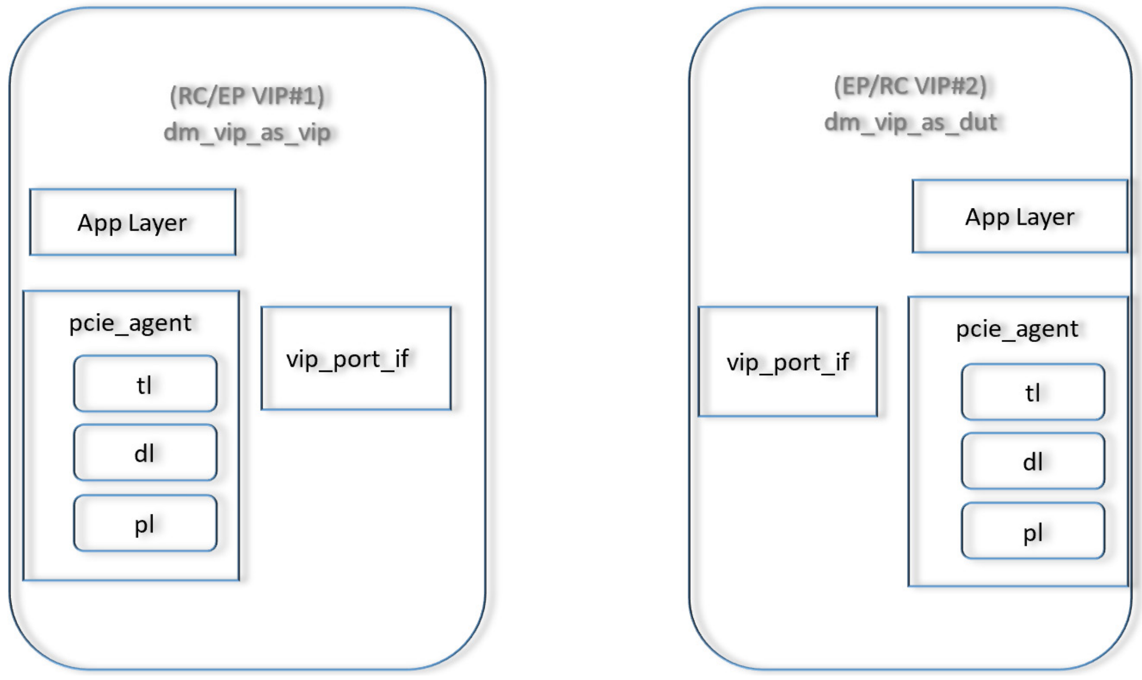
VIP Instantiation and DUT Connections in a Topology File

The reference topology file *topology_dut_cust_controller.svi* shows the recommended approach for connecting the VIP and DUT. The VIP instantiation and connection to DUT can be made by customizing helper macros or by directly instantiating the VIP. In the reference file, creating instances of PCIe VIP HDL agent and its associated interface and forming the link using two instances of interface is done directly in the file without the use of helper macros, whereas the cross-connecting signals of one interface instance to another interface instance is done using the helper macro `SVT_PCIE_ICM_SER_SER_LINK` for serial interface and `SVT_PCIE_ICM_PIPE_PIPE_LINK` for PIPE interface.

❖ Connections in a topology file
  - ✦ Instantiate two instances of dual mode VIP supporting various PCIe signal interconnects.

```
// This is the interface of VIP acting as EP/RC for RC/EP DUT

svt_pcie_if port_if_0(clkreq_n[0], wake_n);
svt_pcie_single_port_device_agent_hdl dm_vip_as_vip(port_if_0);

// This is second VIP instance, will act as Application VIP

svt_pcie_if port_if_1(clkreq_n[0], wake_n);
svt_pcie_single_port_device_agent_hdl dm_vip_as_dut(port_if_1);
```

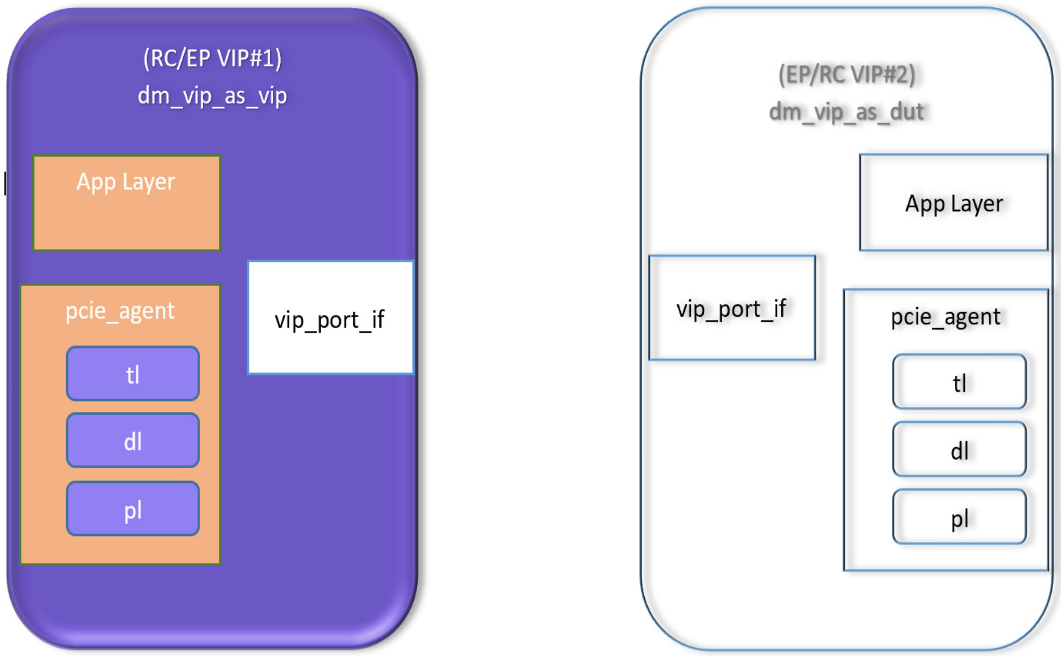
Figure 1-1 VIP and Interface Instantiation



✦ Compile-time settings of first VIP instance.

```
defparam dm_vip_as_vip.SVT_PCIE_UI_DISPLAY_NAME = "dm_vip_as_vip.";
defparam dm_vip_as_vip.SVT_PCIE_UI_ENABLE_CFG_BLOCK= 0;
defparam dm_vip_as_vip.SVT_PCIE_UI_PCIE_SPEC_VER =
`SVT_PCIE_UI_PCIE_SPEC_VER_4_0;
defparam dm_vip_as_vip.SVT_PCIE_UI_PIPE_SPEC_VER =
`SVT_PCIE_UI_PIPE_SPEC_VER_4_2;
defparam dm_vip_as_vip.SVT_PCIE_UI_NUM_PHYSICAL_LANES= 1;
defparam dm_vip_as_vip.SVT_PCIE_UI_HIERARCHY_NUMBER= 0;
defparam dm_vip_as_vip.SVT_PCIE_UI_MPIPE= 0;
`ifdef SERDES_TB
defparam dm_vip_as_vip.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES;
`else
defparam dm_vip_as_vip.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.SVT_PCIE_UI_DEVICE_IS_ROOT= 1; // for EP DUT, VIP is acts as a root
complex.
`else
defparam dm_vip_as_vip.SVT_PCIE_UI_DEVICE_IS_ROOT= 0;// for RC DUT, VIP is Endpoint
`endif

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

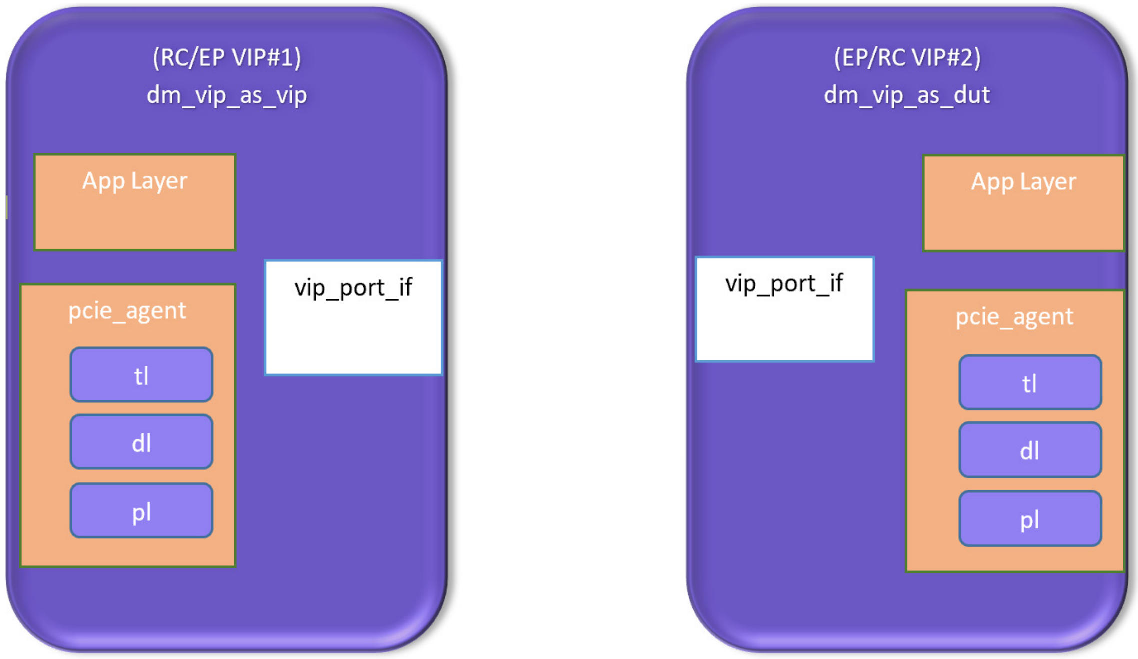
Figure 1-2 VIP#1 Settings



❖ Compile-time settings of second VIP Instance.

```
defparam dm_vip_as_dut.SVT_PCIE_UI_DISPLAY_NAME = "dm_vip_as_dut.";
defparam dm_vip_as_dut.SVT_PCIE_UI_ENABLE_CFG_BLOCK= 0;
defparam dm_vip_as_dut.SVT_PCIE_UI_PCIE_SPEC_VER =
`SVT_PCIE_UI_PCIE_SPEC_VER_4_0;
defparam dm_vip_as_dut.SVT_PCIE_UI_PIPE_SPEC_VER =
`SVT_PCIE_UI_PIPE_SPEC_VER_4_2;
defparam dm_vip_as_dut.SVT_PCIE_UI_NUM_PHYSICAL_LANES= 1;
defparam dm_vip_as_dut.SVT_PCIE_UI_HIERARCHY_NUMBER= 0;
defparam dm_vip_as_dut.SVT_PCIE_UI_MPIPE= 1;
`ifdef SERDES_TB
defparam dm_vip_as_dut.SVT_PCIE_UI_PHY_INTERFACE_TYPE=
`SVT_PCIE_UI_PHY_INTERFACE_TYPE_SERDES;
`else
defparam dm_vip_as_dut.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_dut.SVT_PCIE_UI_DEVICE_IS_ROOT = 0;
`else
defparam dm_vip_as_dut.SVT_PCIE_UI_DEVICE_IS_ROOT = 1;
`endif

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

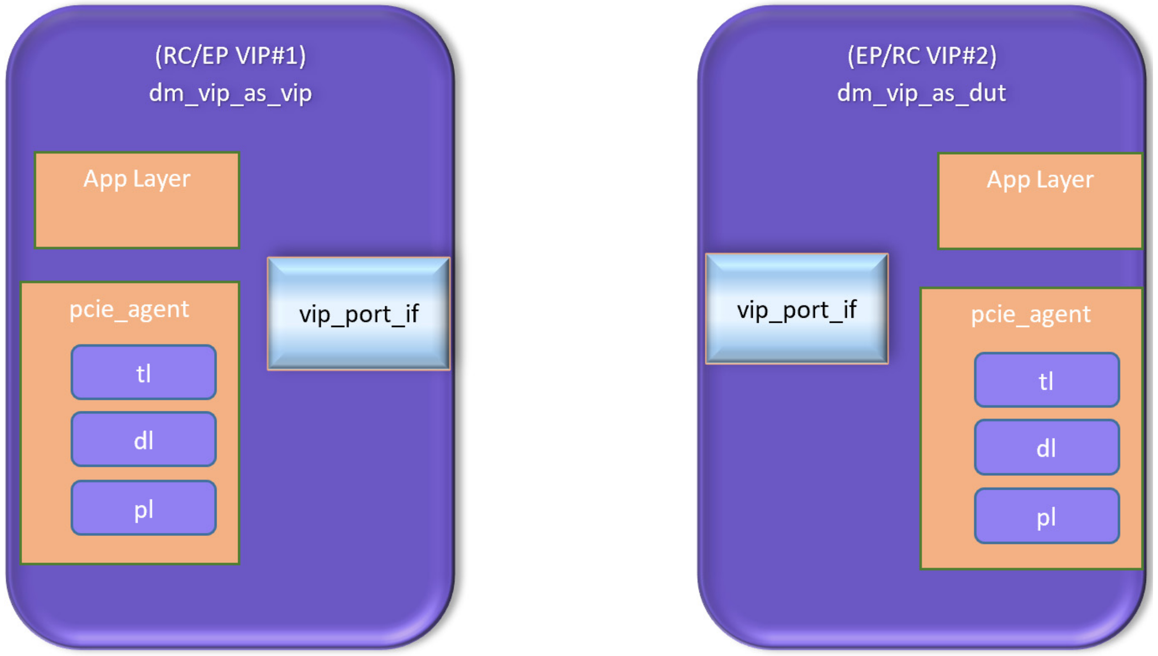
Figure 1-3 RC and EP VIP Settings



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

```
initial begin
  dm_vip_as_vip.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 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_dut.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 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
                            );
end
```

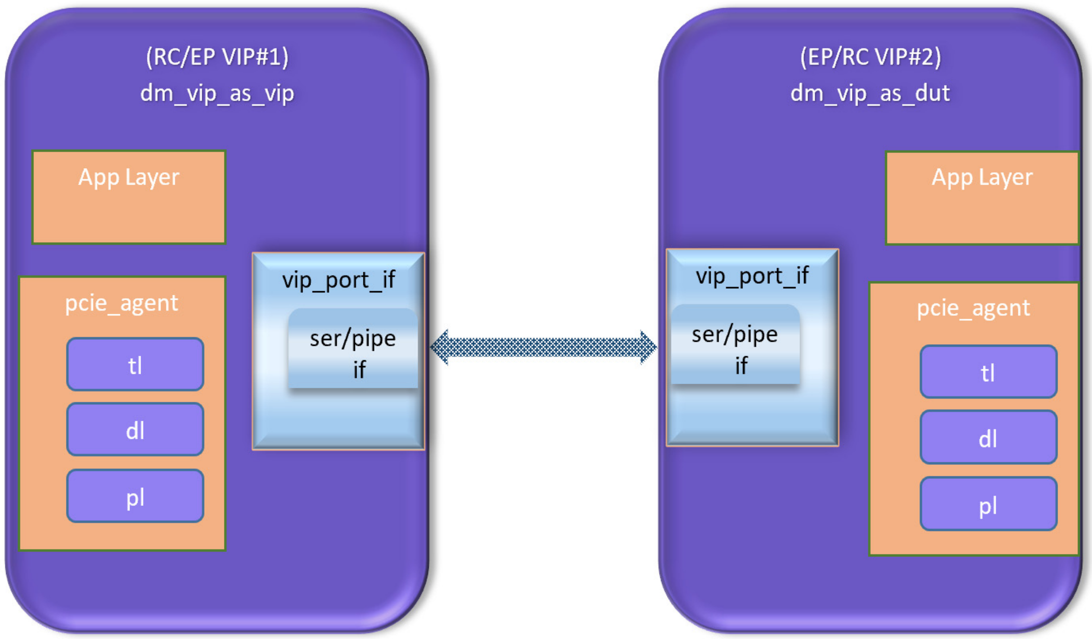
Figure 1-4 VIP Settings Mapped to Interface Variables



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

```
`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,dm_vip_as_dut)
`else

//cross connect PCIe SERIAL interfaces of VIP instances to form PCIe PIPE 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_PIPE_PIPE_LINK(0,dm_vip_as_vip,dm_vip_as_dut)
`endif
```

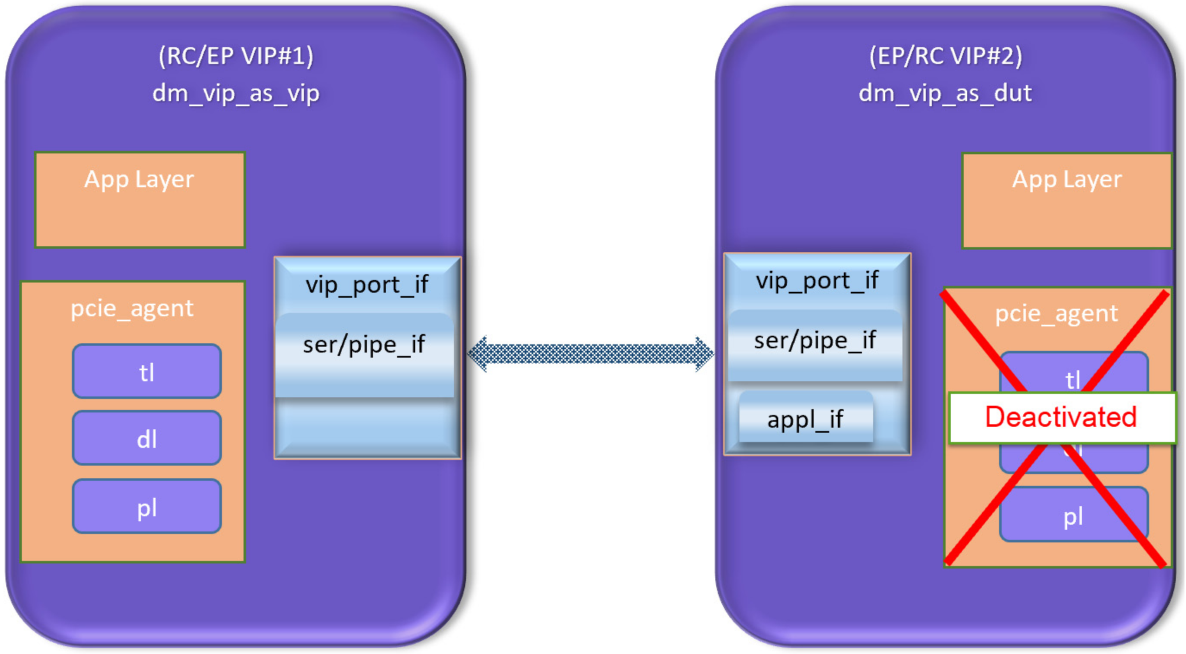
Figure 1-5 RC and EP VIP Interface Cross Connected



- Deactivate the TL, DL and PL stack of EP/RC 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 dm_vip_as_dut.phy_interface_type = `SVT_PCIE_UI_PHY_INTERFACE_TYPE_APP;
```

Figure 1-6 VIP#2 Converted to Application VIP



  - This setting will disable the TL, DL and PL stack of the VIP#2 that is acting as a DUT (`dm_vip_as_dut`).
  - With this change, the VIP#2 now becomes an Application VIP which is connected to the DUT through application interface `app_if`. The Application interface contains clock and reset only.
- Supply clock and reset to 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.vip_port_if.app_if.reset =
`CUSTOMER_DUT_INSTANCE.app_or_tl_reset; // input to VIP instance to reset the app layer which
is driving the DUT TL,DL,PL stack

always @(*) dm_vip_as_dut.app_if.appl_clk =
`CUSTOMER_DUT_INSTANCE.free_running_or_gated_clock; //input to VIP instance for app
layer which is driving the DUT TL,DL,PL stack.
```

- The RC/EP 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 Figure 1-7).

Note The above connection is equivalent to EP/RC DUT directly connected to RC/EP VIP.

- This connection practice ensures 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.

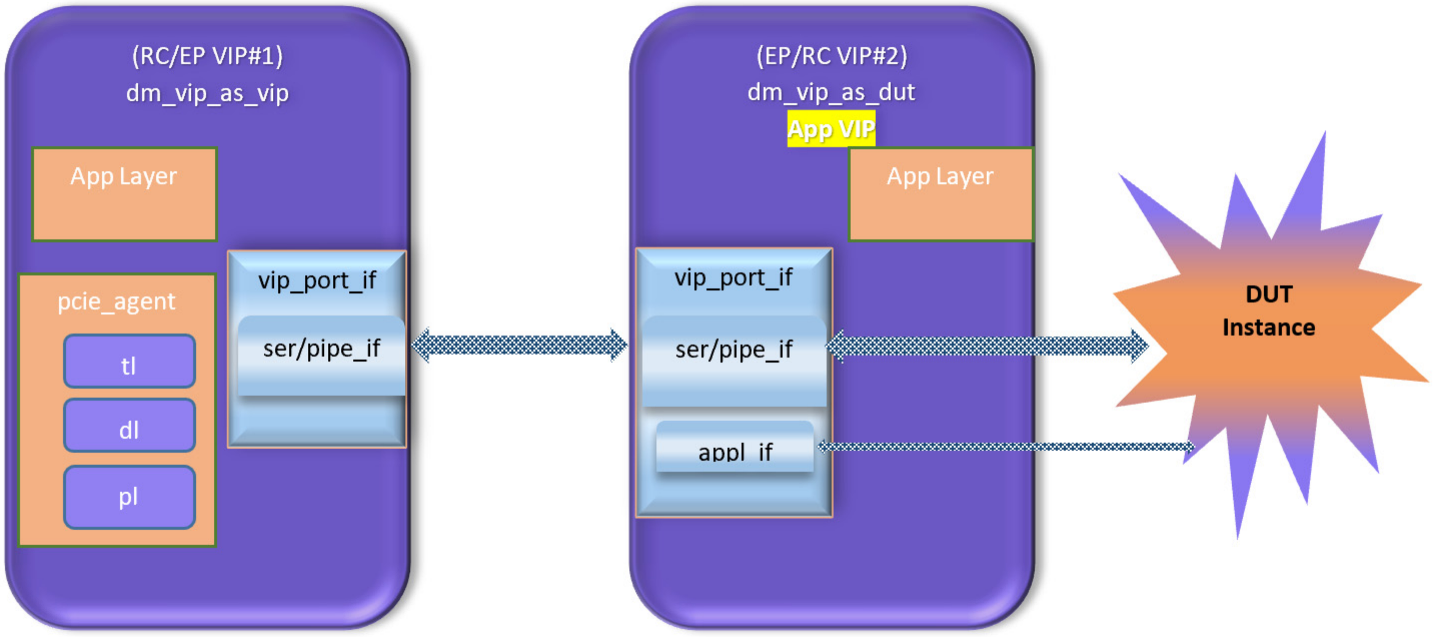
Note The Tx signals of VIP#2 interface (`dm_vip_as_dut.vip_port_if`) must be connected to Tx signals of the DUT. Similarly, Rx of VIP#2 interface (`dm_vip_as_dut.vip_port_if`) to Rx signals of the DUT. The Tx-Rx cross-connection is already done in the macro (`SVT_PCIE_ICM_SER_SER_LINK/SVT_PCIE_ICM_PIPE_PIPE_LINK`) used to connect the two interfaces.

- Connect the EP/RC DUT to RC/EP VIP through `dm_vip_as_dut` instance of VIP.

```
 `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 =
dm_vip_as_dut.vip_port_if.ser_if.rx_datap_0 ;
always @(*) `CUSTOMER_DUT_INSTANCE.rx_datan_0 =
dm_vip_as_dut.vip_port_if.ser_if.rx_datan_0 ;
always @(*)dm_vip_as_dut.vip_port_if.ser_if.tx_datap_0 =
`CUSTOMER_DUT_INSTANCE.tx_datap_0 ;

   always @(*)dm_vip_as_dut.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 =dm_vip_as_dut.vip_port_if.pipe_if.pclk;
always @(*)`CUSTOMER_DUT_INSTANCE.max_pclk =
dm_vip_as_dut.vip_port_if.pipe_if.max_pclk;
always @(*)`CUSTOMER_DUT_INSTANCE.reset = common_pwr_on_reset;
always @(*)dm_vip_as_dut.vip_port_if.pipe_if.rate= `CUSTOMER_DUT_INSTANCE.rate;
always @(*)dm_vip_as_dut.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 =
dm_vip_as_dut.vip_port_if.pipe_if.rx_data_0 ;
always @(*)`CUSTOMER_DUT_INSTANCE.rx_data_k_0 =
dm_vip_as_dut.vip_port_if.pipe_if.rx_data_k_0 ;
always @(*)`CUSTOMER_DUT_INSTANCE.rx_status_0 =
dm_vip_as_dut.vip_port_if.pipe_if.rx_status_0 ;
always @(*)`CUSTOMER_DUT_INSTANCE.rx_valid_0 =
dm_vip_as_dut.vip_port_if.pipe_if.rx_valid_0 ;
. . . . . .
. . . . . .
```

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

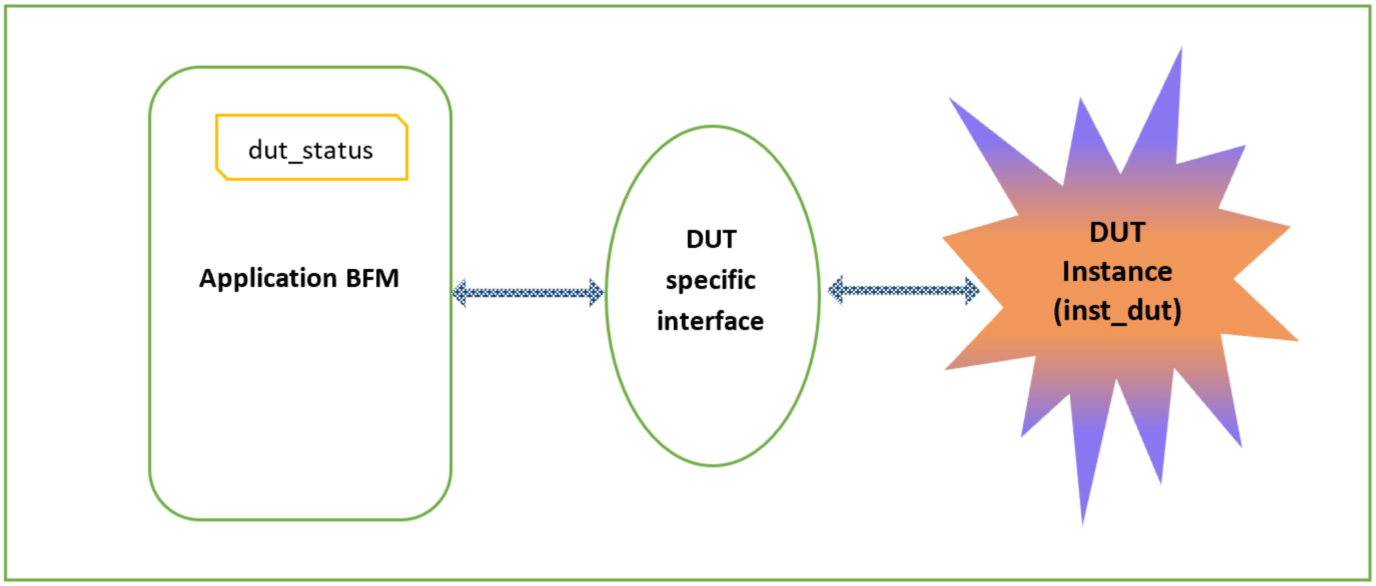
Figure 1-7 VIP-DUT Connection



Step 3: Basic Application BFM Integration

Note The Application BFM provided with the testbench is only a reference model. If the DUT Application interface is AXI, then you can take this as a reference and implement/enhance it as per your DUT requirement. The user should own this component and maintain it. Synopsys does not provide reference Application BFM for any other interface.

Figure 1-8 DUT Controller ENV with Basic Application BFM Instantiation



In the initial phase, Application BFM is used to communicate the DUT status to the sequences and test cases. A basic Application BFM *basic_pcie_dut_external_app_bfm.sv* is available in *env/reference_app_bfm* directory of the `tb_dut_pcie` testbench. In the absence of your custom Application BFM, you can use this basic external Application BFM with few updates and connections.

This basic Application BFM will not compile as-is in your testbench environment. The `update_ltssm_state` method contains the example code for mapping of DUT LTSSM state to `svt_pcie_status` object of the VIP. Similarly, you must map your DUT LTSSM states to the `ltssm_state` object of `svt_pcie_pl_status` class of the VIP. The test suite sequences and test cases rely on this status object to make progress.

The Application BFM connects to the DUT via the `dut_specific_if` interface. A basic DUT-specific interface is also part of this testbench in *env/reference_app_bfm* directory. It includes the basic signals like DL and PL link up and DUT LTSSM states that are required to communicate the DUT status to tests/sequences.

To check the initial link up, you must run PL layer tests, these tests depend on DUT status to complete the sequence.

Note With the basic Application BFM, you can only run PL layer tests. For DL and TL layers, you must implement the complete Application BFM.

To instantiate and connect the Application BFM to the DUT interface signal, perform the following steps:

1. Copy the *basic_pcie_dut_external_app_bfm.sv* to any other location, rename it and implement the `update_ltssm_state` method to map the DUT LTSSM states with VIP LTSSM states. Include this file in *cust_pcie_test_suite_pkg_files.svi* for compilation.

```
function void
`SVT_PCIE_TEST_SUITE_DUT_EXTERNAL_APP_BFM_TYPE::update_ltssm_state(bit [5:0]
encoded_state);
 case(encoded_state)
    6'h00,6'h05: begin dut_status.pcie_status.pl_status.ltssm_state =
svt_pcie_types::DETECT_QUIET ;            end
    6'h01: begin dut_status.pcie_status.pl_status.ltssm_state =
svt_pcie_types::DETECT_ACTIVE;                  end
…
…
  endcase
endfunction
```

2. Review the contents of the file *tb_dut_pcie/env/reference_app_bfm/dut_specific_interface.svi*. Copy this file into any other directory, include this interface file and define the macro `SVT_PCIE_TEST_SUITE_DUT_SPECIFIC_IF` in *cust_pre_tb_top.svi.*

```
`define SVT_PCIE_TEST_SUITE_DUT_SPECIFIC_IF dut_specific_interface
`include "dut_specific_interface.svi"
```

3. Instantiate the DUT-specific interface file in your topology file and pass the interface to the `external_app_bfm` instance of env through `uvm_config_db#()::set` utility.

```
/** DUT specific signals */
dut_specific_interface dut_specific_if();
initial
   uvm_config_db#(virtual dut_specific_interface)::set(uvm_root::get(),
"uvm_test_top.env. external_app_bfm ", "dut_specific_if", dut_specific_if);
```

4. Connect the interface signals to appropriate signals of the DUT in the topology file.

   For example,

```
assign dut_specific_if.core_clk        = dut_core_clk;
assign dut_specific_if.pl_link_up      = dut_pl_link_status[3];
assign dut_specific_if.dut_ltssm_state = dut_ltssm_info[5:0];
assign dut_specific_if.dl_link_up      = dut_dl_link_up;
```

5. Instantiate the basic Application BFM in env class as described in Step 4.
6. Create the `external_app_bfm` instance in the `build_phase` of env class extended class from `snps_pcie_dm_dut_env` as shown in Example 1-1.

Step 4: Create a DUT ENV Wrapper

The instructions given in this step are mandatory even though you have your own env (`uvm_env`).

❖ Create a DUT env file extending from `snps_pcie_dm_dut_env`. During initial integration phase, this class is required to instantiate Application BFM.

❖ Other DUT application specific components like application specific interface, application VIP agents, application VIP configuration instance, RAL Model and so on can also be directly instantiated in this env or user env class. Alternatively, user testbench env can be instantiated in user base test from which all the test suite tests will be extended. For more details, see Step 5.

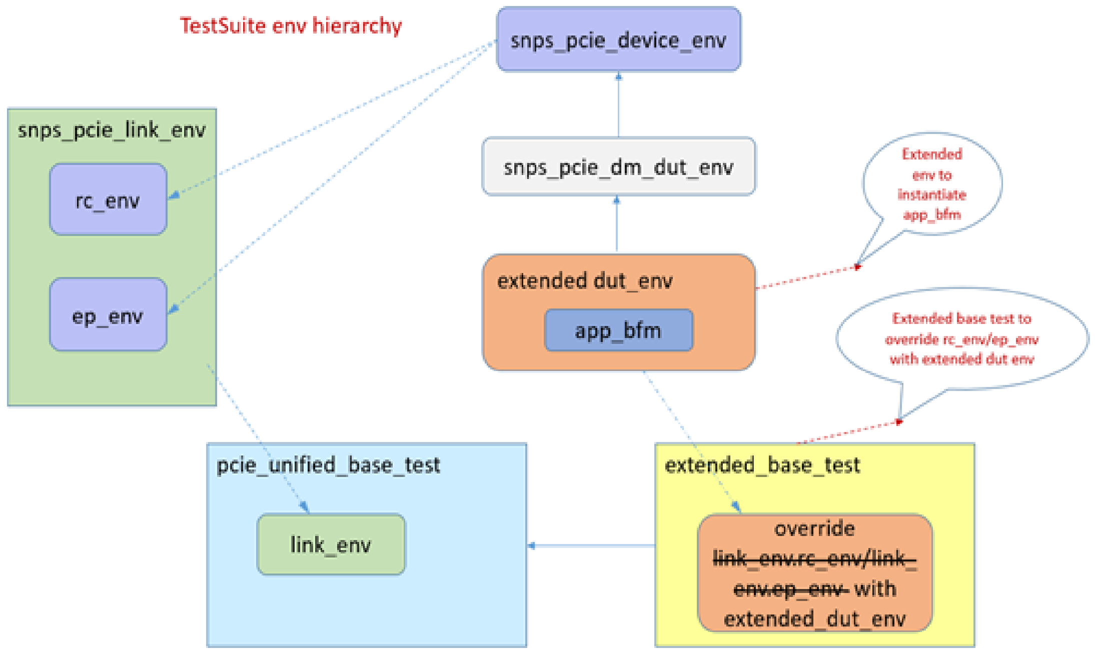
Example 1-1 Create DUT ENV Extended from `snps_pcie_dm_dut_env` and Instantiate Basic Application BFM in it

```
class dut_controller_env extends snps_pcie_dm_dut_env;
  `uvm_component_utils(dut_controller_env)
……
…
/** Instantiating Application BFM*/
   pcie_dut_external_app_bfm external_app_bfm;
function void dut_controller_env::build_phase(uvm_phase phase);
….
external_app_bfm =
pcie_dut_external_app_bfm::type_id::create("external_app_bfm",this);
endfunction
endclass
```

The DUT env file can be included in *cust_pcie_test_suite_pkg_files.svi* for maintaining the correct compilation order. You must override the `snps_pcie_device_env` instance (instantiated in `snps_pcie_link_env` as `rc_env/ep_env`) by this class in the `set_env_overrides` method of extended base test as described in Step 5.

Figure 1-9 shows the env class hierarchy.

Figure 1-9 ENV Class Hierarchy


TestSuite env hierarchy
snps_pcie_device_env
snps_pcie_link_env
rc_env
ep_env
snps_pcie_dm_dut_env
Extended env to instantiate app_bfm
extended dut_env
app_bfm
Extended base test to override rc_env/ep_env with extended dut env
pcie_unified_base_test
link_env
extended_base_test
override ~~link_env.rc_env/link_env.ep_env~~ with extended_dut_env

The `snps_pcie_link_env` class contains an object of `snps_pcie_device` env for `rc_env` and `ep_env`. You can create your DUT env class extending from `snps_pcie_dm_dut_env` which is extended from `snps_pcie_device_env`.

The `snps_pcie_link_env` is instantiated in the `pcie_unified_base` test which you must extend to create `user_base_test` and override the `ep_dut/rc_dut` instance in `link_env` with the user-extended env in the `set_env_override` method of the extended `base_test`.

Step 5: Create a Base Test by Extending from pcie_unified_base_test

❖ Re-implement `pcie_unified_base_test::set_env_override` method in your extended base test. This method is invoked in `pcie_unified_base_test` at the appropriate point during the `build_phase`.

❖ Override the default env for `ep/rc_env` present in `snps_pcie_link_env` with the env class name that was created in Step 4.

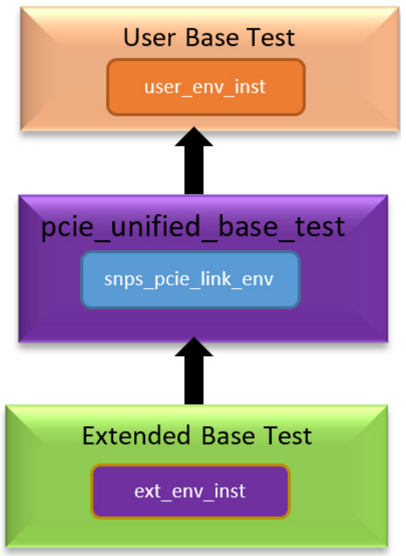
```
virtual function void set_env_overrides(`SVT_PCIE_TEST_SUITE_CONFIGURATION_TYPE
link_cfg);
  //ep_env type override
    set_inst_override_by_type({link_cfg.path_to_pcie_test_suite_env, ".ep_env" },
                               snps_pcie_device_env::get_type(),
                 `ifdef VIP_AS_EP_DUT
                              snps_pcie_dm_vip_env::get_type());
                  `else
                               dut_controller_env::get_type());
                  `endif
  //rc_env type override
    set_inst_override_by_type({ link_cfg.path_to_pcie_test_suite_env, ".rc_env"
},
                              snps_pcie_device_env::get_type(),
                  `ifdef VIP_AS_RC_DUT
                              snps_pcie_dm_vip_env::get_type());
                  `else
                               dut_controller_env::get_type());
                  `endif
  endfunction
```

All VIP configuration settings as per the DUT configuration can be done in this file or using *src/txt* file approach. For example, clock tolerance setting in case of SERDES interface, LTSSM state timeout settings, speed, link width, specification version and so on.

Note If your env has an instance of VIP agent, then it needs to be removed as the required VIP instance is already present in the TS env (`snps_pcie_link_env`). Retaining additional instances of VIP agent will lead to the uvm_fatal—`could not register err_check` or `Display name does not match`.

If you already have settings in your base test which is also instantiating user env, then the recommended use model is as follows:

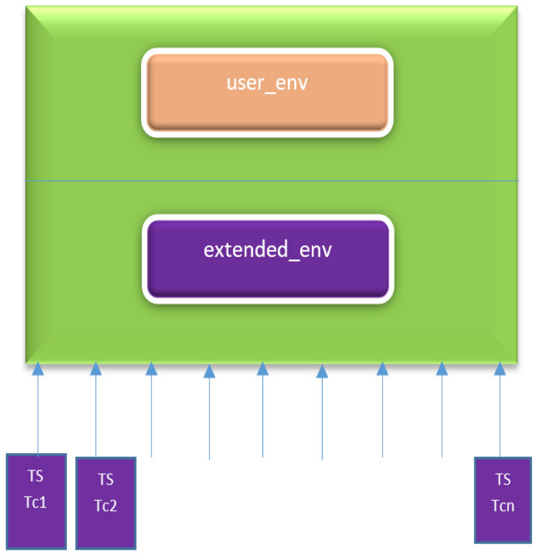
Figure 1-10 Base Test Hierarchy



- Extend the `pcie_unified_base_test` from `user_base_test`.

  This can be done by setting the macro `SVT_PCIE_TEST_SUITE_USER_BASE_TEST` to user base test name. By default, the `unified_base_test` is extended from `uvm_test`.
- As the extended base test will now be extended from unified base test, it includes both user env instance and extended DUT env instance that was created in Step 3.
- All test suite tests will be extended from this extended base test as shown in Figure 1-11.

Figure 1-11 Extended Base Test



You can make/add implementation-specific customizations in this base test class by referring to file *pcie_e/rcp_dut_cust_base_test.sv*. It is recommended that, in this file, you can add/modify only those configurations that are generic, applicable to all test and are not dynamic. For all test-specific configurations, follow the *src/txt* file approach.

Note All the mandatory code is present only in `pcie_unified_base_test`, therefore you must extend your base test from `pcie_unified_base_test` depending on the requirement even for meeting multi-DUT requirement in a single test. The `pcie_ep/rc_dut_cust_base_test` class must be used as a reference. It is recommended not to extend this class, use it only as a reference.

All test suite tests are extended from `pcie_unified_base_test`. To replace the test suite base test with your extended base test, you must add the following macros in the compile file discussed in Step 6.

```
+define+SVT_PCIE_TEST_SUITE_EP/RC_DUT_TEST=<extended_base_test_name>
```

For example,

```
+define+SVT_PCIE_TEST_SUITE_EP/RC_DUT_TEST=pcie_ep_dut_controller_base_test
```

❖ Use of scoreboard during initial integration: Before enabling scoreboard, make sure that the inbound/outbound flow is correct to avoid unnecessary errors from the scoreboard. It is recommended to disable scoreboard by setting the `enable_scoreboard` attribute of `pcie_test_suite_configuration_svt` class to 0 during the initial integration of DUT and initial phase of Application BFM development, until the inbound/outbound flow and port connections are verified.

```
cfg.enable_scoreboard = 1'b0;
```

Step 6: Create a DUT Compile File List

You can use your compile setup to make the test suite inclusions and settings.

Reference

- *compile_dut_cust_controller.f*
- *compile_dut_snps_pcie_ep_pue_iip_pipe.f*
- *compile_dut_snps_pcie_ep_pue_iip_serial.f*
- *compile_dut_snps_pcie_rc_pue_iip_pipe.f*
- *compile_dut_snps_pcie_rc_pue_iip_serial.f*
- The compile file includes compiler directives (`+incdir`, `-y` `+libext+` and so on) required to compile the DUT files.
- If you run the test inside `tb_dut_pcie` directory, then you must create a compile file.
  - If you run the test outside `tb_dut_pcie` directory, then you can skip creating this file.

❖ With `gmake` option

The DUT compile file list can be specified either via make option `SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE` or without it. The *svt_pcie_test_suite_dut_compile_file.f* file is created at runtime and is not part of the installation example. When you invoke the simulation using `gmake` command, `tb_dut_pcie/Makefile` creates a symbolic link named *svt_pcie_test_suite_dut_compile_file.f* pointing to the filename passed via `gmake` switch `SVT_PCIE_TEST_SUITE_DUT_COMPILE_FILE`.

Usage:

```
    gmake USE_SIMULATOR=<vcsvlog> \
    <testname: class to be instantiated using 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=<tests/*.scr> \
```

❖ Without `gmake` option

You must create this file in your local directory and create a link (symbolic link) with *svt_pcie_test_suite_dut_compile_file.f* file.

Note You must modify the VIP configuration attributes as per the DUT requirement to get the link up to the desired speed of operation–that is, Gen1/Gen2/Gen3/Gen4.

1.2 Validating the Integration

Table 1-1 lists the tests required for validating the integration.

Table 1-1 Test Cases for Validating the Integration

Layer	Speed	Test Case	Description
PL	GEN-1	`demo_pl_gen1_linkup_test-ep/rc`	This test case will check the LTSSM flow up to L0 in Gen1 speed.
PL	GEN-2	`demo_pl_gen1_to_gen2_speed_change_test-ep/rc`	This test case will change the speed to Gen2 and check the LTSSM flow up to L0 in Gen2 speed.
PL	GEN-3	`demo_pl_gen1_to_gen3_speed_change_test-ep/rc`	This test case will change the speed to Gen3 and check the LTSSM flow up to L0 in Gen3 speed.
PL	GEN-2/3	`demo_gen3_pl_all_supported_speed_change_test-ep/rc`	This test case validates multiple speed changes and verifies if the LTSSM is successfully achieving all speed changes. The flow is as follows: Gen1 -> Gen-2 -> Gen-3 -> Gen-2 -> Gen1 -> Gen3 -> Gen1

Note By following the instructions (6 steps) for initial phase of integration, you should be able to get at least the link up at Gen1.