Verification Continuum™

VC Verification IP
PCIe Test Suite Reference
External Application BFM

Version S-2021.06, June 2021

SYNOPSYS®

Contents

Preface

About This Document

This guide provides you the information on how to implement Application BFM implementation 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 Reference External Application BFM

This guide describes the steps to implement Application BFM implementation.

Attention A reference model of Application BFM for EP and RC DUT is present in *env/reference_app_bfm* directory in the `tb_dut_pcie` testbench directory. As the name suggests, this should be strictly used only as a reference to build your own Application BFM. You can use the code snippet or the entire code as per your requirements. Synopsys does not provide support for the user Application BFMs, which are developed and maintained by the users. The reference Application BFM is for AXI interface bus, there is no reference Application BFM available for any other application interface.

This chapter discusses the following topics:

- Introduction
- Integrating Basic Application BFM
- Application BFM Port Connections
- Writing Application BFM
- Validating the Integration

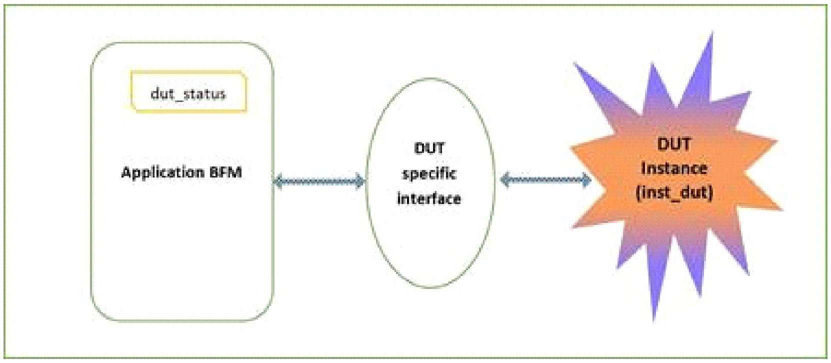
1.1 Introduction

An Application BFM is a component that interfaces with the DUT. Application BFM is used to:

- Make available inbound (VIP->DUT) transaction (by converting DUT application specific transactions to `svt_pcie_tlp` type) to test suite environment.
- The test scheduled outbound transaction is sent to DUT (by converting `svt_pcie_tlp` type to DUT-specific type) on the transmit path.
- Map DUT LTSSM state to `svt_pcie_status` object that is required by sequences and test cases

1.2 Integrating Basic Application BFM

Figure 1-1 Integrating Basic Application BFM



Note This section is also part of the initial integration steps described in Step 3 of *VC Verification IP PCIe Test Suite EP/RC DUT Integration Guide*. You can skip this step (section 1.2) if you have already completed.

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. This is part of module word. Therefore, it can be specified in user's *tb_top.sv* file (by default, this is not present in *tb_dut_pcie/top.sv*) file or topology file.

```
/** 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. Equivalent get call of `dut_specific_if` should be present inside user `external_app_bfm` class. By default, it is present in *basic_pcie_dut_external_app_bfm.sv*.

```
function void
`SVT_PCIE_TEST_SUITE_DUT_EXTERNAL_APP_BFM_TYPE::connect_phase(uvm_phase phase);
   if(!uvm_config_db#(virtual `SVT_PCIE_TEST_SUITE_DUT_SPECIFIC_IF)::get(this,
"", "dut_specific_if", dut_specific_if)) begin
     `uvm_fatal("connect_phase", "Did not received a dut_specific_if via
config_db, Must set dut_specific_if of type `SVT_PCIE_TEST_SUITE_DUT_SPECIFIC_IF
from top using uvm_config_db");
    end
endfunction
```

5. 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;
```

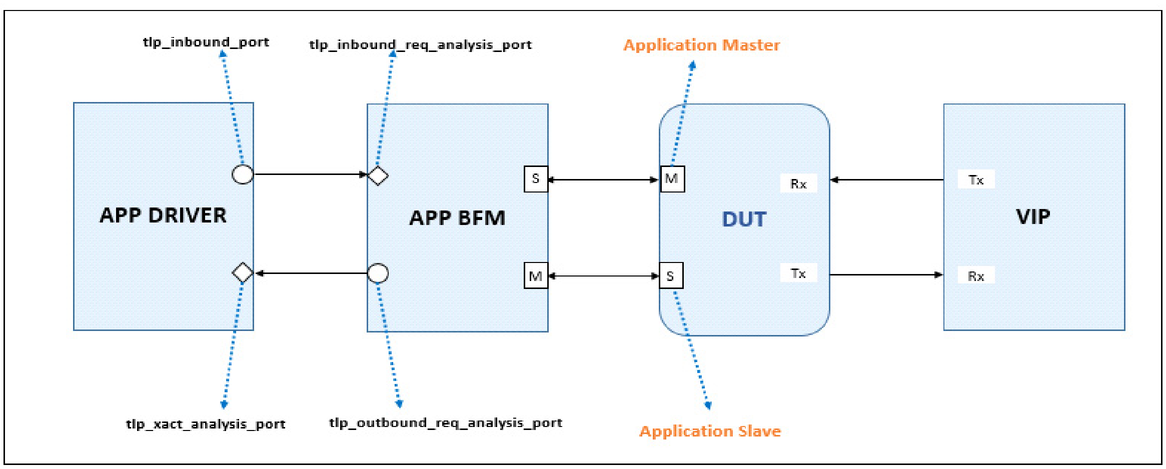
6. Create an env class which extends from `snps_pcie_dm_dut_env`. Save this file in your local directory. This class encapsulates an `external_app_bfm`.
7. Include this file in *cust_pcie_test_suite_pkg_files.svi* to set the correct compilation order.
8. Instantiate the basic Application BFM in env class as described in Step 4 of *VC Verification IP PCIe Test Suite EP/RC DUT Integration Guide.* Create the `external_app_bfm` instance in the `build_phase` of extended env class.
9. Create a user base test extending from `pcie_unified_base_test` and instantiate the basic Application BFM in it.
10. Re-implement the `pcie_unified_base_test::set_env_override` method in your extended base test to override the `snps_pcie_device_env` instance of `snps_pcie_link_env` with the extended env class. This method is invoked in `pcie_unified_base_test` at the appropriate point during the `build_phase`.
11. 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 of *VC Verification IP PCIe Test Suite EP/RC DUT Integration Guide*.

    For example,

```
+define+SVT_PCIE_TEST_SUITE_EP/RC_DUT_TEST=pcie_dut_controller_base_test
```

1.3 Application BFM Port Connections

Figure 1-2 Application BFM Port Connections



This section describes the Application BFM port connections in the environment.

Figure 1-2 shows the following analysis ports in the Application BFM component:

- `tlp_inbound_req_analysis` port is used to pass the inbound packets to the upper layer for end-to-end scoreboarding and optionally for response generation.
- `tlp_outbound_req_analysis` port is used for outbound transfers, it takes the PCIe request from the upper layer driver sequencer so that it can be transformed to application type and given to the controller.

The tlp_inbound_req_analysis should be connected to the tlp_inbound_port of the dm_driver instance (type snps_pcie_dm_dut_driver) in the connect_phase of extended env class in which the Application BFM is instantiated.

Similarly, the tlp_outbound_req_analysis port should be connected to the tlp_xact_anlaysis_port of the dm_driver instance.

```
function void dut_controller_env::connect_phase(uvm_phase phase);
 super.connect_phase(phase);
 // Outbound port from DUT Driver to External APP BFM
 dm_driver.tlp_xact_analysis_port.connect(external_app_bfm.tlp_outbound_req
 _analysis_port);

 // Inbound port from External APP BFM to DUT Driver
 external_app_bfm.tlp_inbound_req_analysis_port.connect(dm_driver.tlp_inbou
 nd_port);

 // Analysis port connection between driver and external_app_bfm for backdoor access of DUT registers
 // cfg_database_service transactions used for configuring DUT through backdoor (optional for DUT)
 dm_driver.cfg_database_service_analysis_port.connect(external_app_bfm.cfg_
 database_service_req_analysis_imp);

 // Response port for the cfg_database_service transaction is communicated back to sequence through this port
 external_app_bfm.cfg_database_service_response_analysis_port.connect(dm_dr
 iver.cfg_database_service_response_analysis_imp);
 ……
endfunction
```

Optionally, the Application BFM should also implement the backdoor read and write access of DUT registers by implementing cfg_database_service_req_analysis_imp TLM port that gets connected to the dm_driver. The dm_driver takes svt_pcie_cfg_database_service request from sequences and puts it on its cfg_database_service_analysis_port which is connected to cfg_database_service_req_analysis_imp port of Application BFM.

The cfg_database_service_response_analysis_port of Application BFM is used to send the DUT response to backdoor configuration access back to the sequence that made the request. This analysis port is connected to cfg_database_service_response_analysis_imp port of dm_driver.

1.4 Writing Application BFM

The section describes the steps and design guidelines for the analysis ports use model to integrate the DUT with the PCIe test suite environment.

1. Application BFM must keep a reference of the DUT environment (extended from snps_pcie_dm_dut_env) object named dut_env. This reference can be used to access the APIs implemented in that class. Currently, this is commented in the basic Application BFM.

```
/** Handle to DUT environment.
 * Create an instance of the dut environment that will be extended from snps_pcie_dm_dut_env.
 */
 // snps_pcie_dm_dut_env dut_env;
```

2. Application BFM must receive the configuration object of type `svt_pcie_device_configuration` with the following method and use it to configure the core before initiating a link training. The configuration can be obtained using the following in the build phase.

```
uvm_config_db
void'(uvm_config_db#(svt_pcie_device_configuration)::get(this,"","dut
_cfg", dut_cfg));
```

The configuration object must be passed to BFM in the build phase of the extended env.

```
uvm_config_db#(svt_pcie_device_configuration)::set(this,
"external_app_bfm", "dut_cfg", active_cfg);
```

3. Use of scoreboard during initial stages of Application BFM development.

   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 in the initial phase of Application BFM development, until the inbound/outbound flow and port connections are verified. This can be done in the extended base test as shown below:

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

   OR

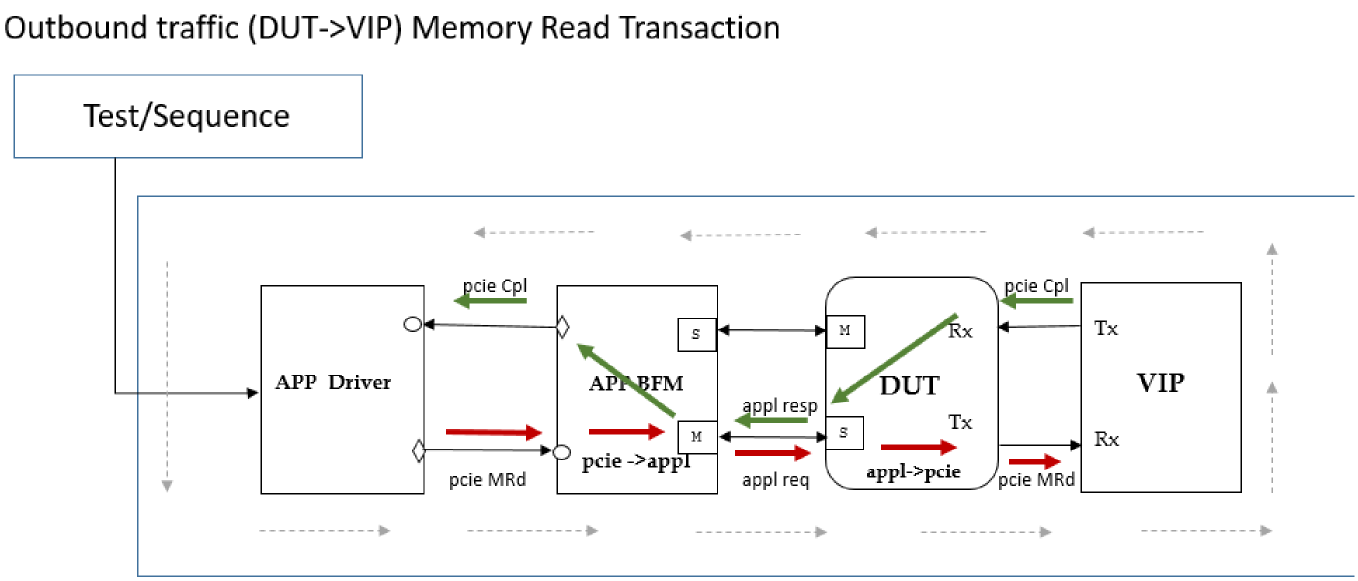
   It can be set using the .txt file that is passed during runtime.

   ```
   cfg[0].enable_scoreboard=0
   ```

4. Outbound Transactions (DUT->VIP).

   Application BFM must provide the `write_dut_outbound_tlp_xact_req()` method for the analysis port implementation (`tlp_outbound_req_analysis_port`) to import transactions of type `svt_pcie_tlp` from tests/sequences and drive it onto the DUT core application interface.

The request and response flow is shown in the below diagram.



Transactions initiated from the Application agent sequencer by the test/sequence reaches the Application driver which puts the PCIe transaction on the outbound analysis port. This is picked up by the Application BFM at the `tlp_outbound_req_analysis_port` and stored in separate queues for outbound request and outbound completions.

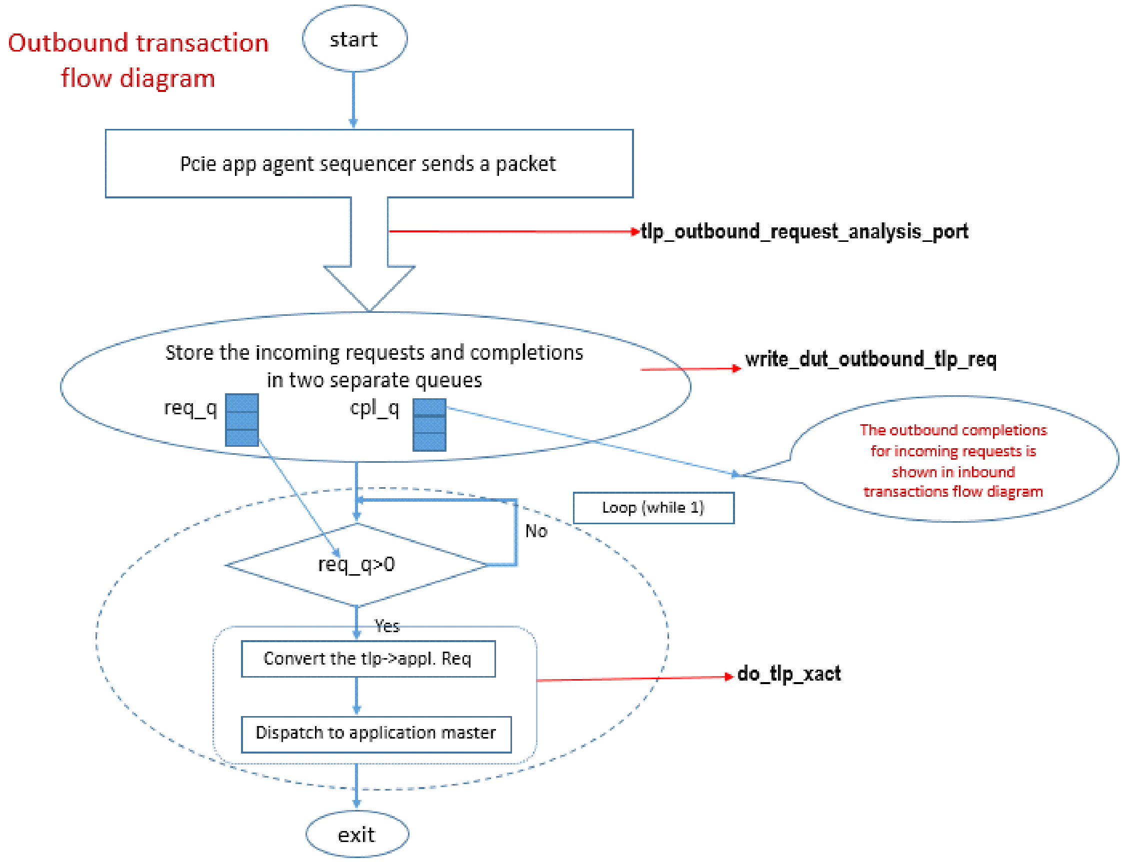
This outbound PCIe TLP (`svt_pcie_tlp`) requests must be converted to application transactions and sent to the application master to drive it on to the core. The Application BFM must implement the `svt_pcie_tlp` to Application Layer transaction conversion logic. The conversion from `svt_pcie_tlp` to application level where the application interface is AXI is shown in the `tlp_to_axi_conversion` method of the reference Application BFM (*pcie_ep_dut_external_app_bfm.sv*/*pcie_rc_dut_external_app_bfm.sv*).

Note This should be used as reference only as the actual conversion is not generic even in cases where the application is same as the reference Application BFM interface (AXI), The conversion logic is purely DUT dependent.

The read requests must be stored in a read queue to later track the completions associated to the requests.

Figure 1-3 shows the flow chart for outbound requests.

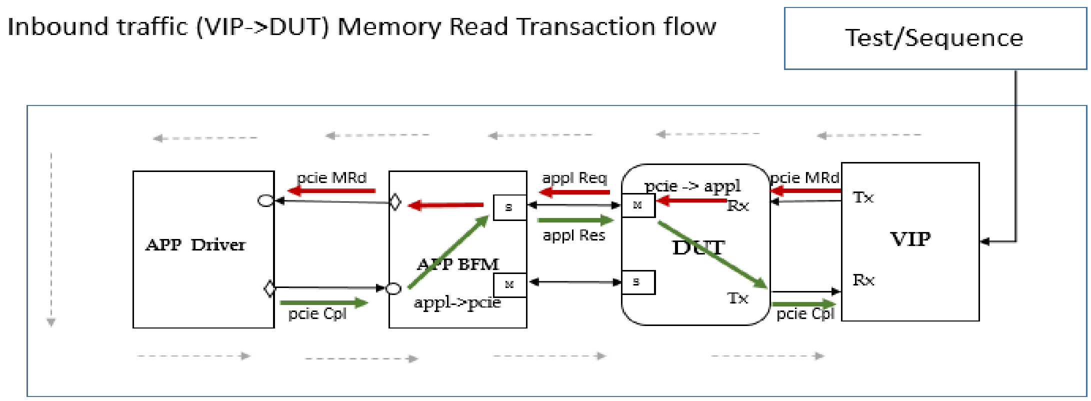
Figure 1-3 Outbound Transaction Flow Diagram



Note The outbound CplD from upper layer can be dropped if the DUT is capable to generate completions for inbound read requests from VIP. To prevent scoreboard from giving false failures for these completions, the scoreboard comparison of outbound completions should be stopped. This can be done by setting the `svt_pcie_test_suite_configuration` class attribute `enable_sb_dut_tx_cpl_compare` to 0.

5. Inbound transactions (DUT<-VIP): The inbound (VIP->DUT) transaction flow in the environment

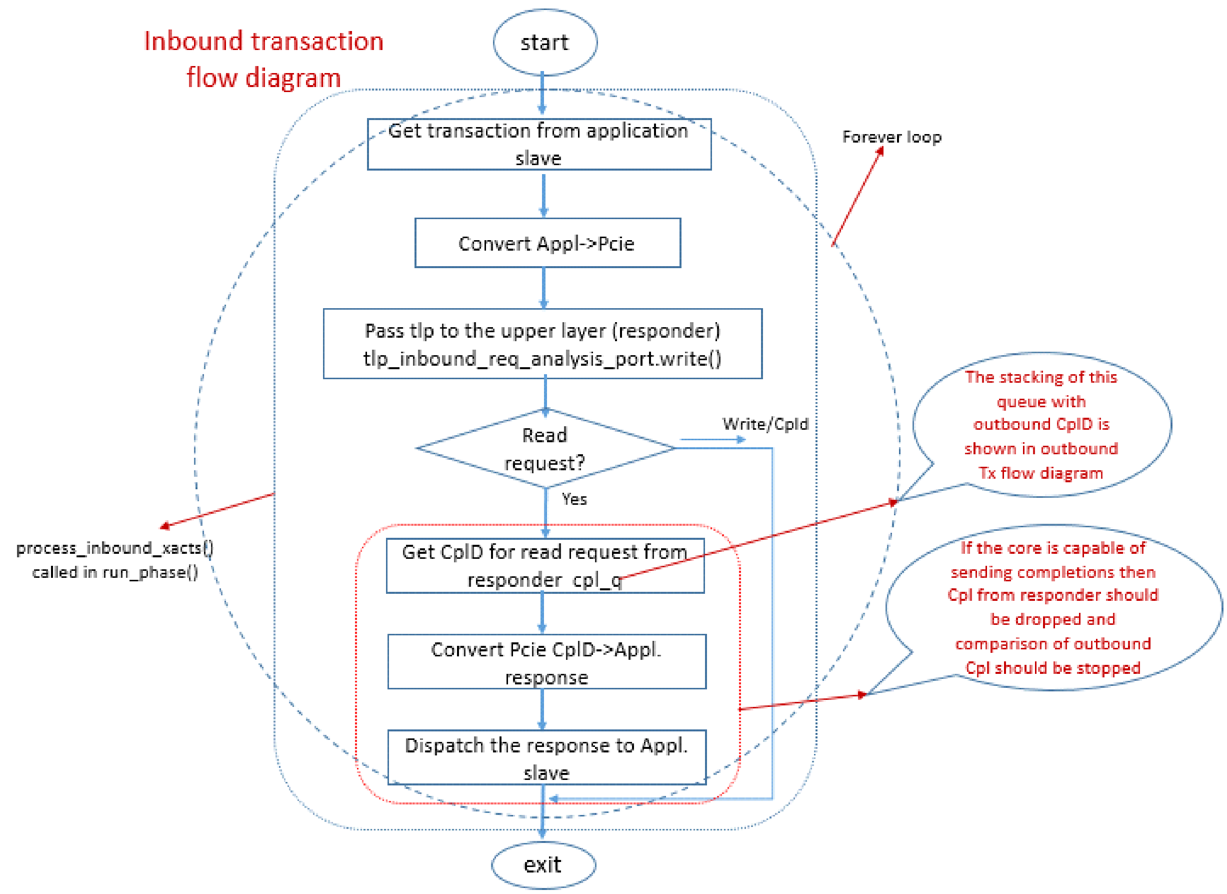
for Inbound request is shown below.



Application BFM must implement an inbound analysis port (`tlp_inbound_req_analysis_port`) of type `svt_pcie_tlp` which is imported by the DM driver (`tlp_inbound_port`) and passed to other components in the environment. Therefore, the Application BFM must form a transaction of type `svt_pcie_tlp` when an inbound transaction happens at the core's interface and invoke the write() function of the inbound TLP analysis port with a reference of the transaction object.

Figure 1-4 shows the inbound (VIP->DUT) transactions flow diagram for Inbound request.

Figure 1-4 Inbound Transaction Flow Diagram

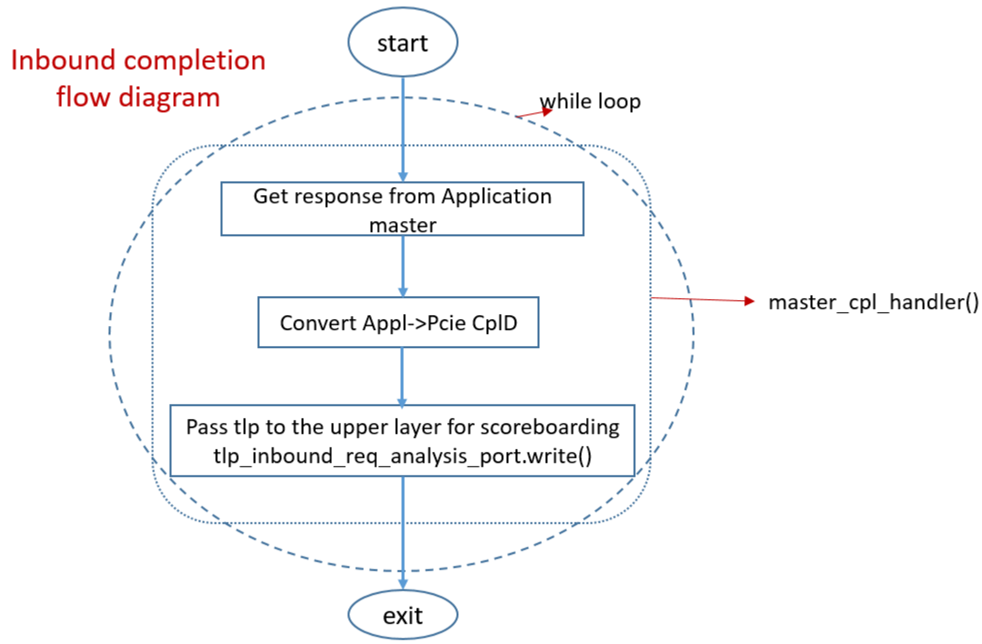


When the inbound TLP that is originated from VIP reaches the slave port of application BFM, it should be picked and converted to `svt_pcie_tlp` transactions and sent to the upper layer for end-to-end scoreboarding by writing it on the `tlp_inbound_req_analysis_port`. The conversion from application level to `svt_pcie_tlp`, where the application interface is AXI is shown in the `axi_to_tlp_conversion` method of the reference Application BFM (*pcie_ep_dut_external_app_bfm.sv*/ *pcie_rc_dut_external_app_bfm.sv*).

Note This should be used as reference only as the actual conversion is not generic even in cases where the application is same as the reference Application BFM interface (AXI), The conversion logic is purely DUT dependent.

If the DUT cannot generate completions for incoming read requests, then the Application BFM should wait for completions from the upper layer which acts as a responder, convert this outbound response for incoming read to application level transaction and send it back to the application Slave as response which puts it on the application Master port of DUT.

6. The Inbound completion flow is explained in the following flow diagram.



The inbound response for outbound request are picked at the Master of port of application Master.

The responses which are in application interface type transactions should be converted to PCIe format completions and sent to the upper layer through `tlp_inbound_analysis_port`. These will be picked up by the other components like scoreboard for end-to-end scoreboarding.

Hint You can refer to the PCIe <-> AXI conversion logic in files *pcie_ep_dut_external_app_bfm.sv* for EP DUT and file *pcie_rc_dut_external_app_bfm.sv* for RC DUT.

7. Configuration space access via backdoor.

Application BFM also provides a utility to configure PCIe configuration space via backdoor. For EP DUT, it is optional because configuration space is accessible via front door (via `CfgWr`/`CfgRd` transaction). But for RC DUT it is mandatory to implement this method as front door access to RC DUT configuration space is not possible. In case of EP DUT, this API is required for the optional feature of routing all configuration accesses required by the test suite tests to DUT via backdoor. You can set the use model by setting the TS configuration class `svt_pcie_test_suite_configuration` attribute `backdoor_cfg_req` and implement logic to write/read configuration registers present inside the DUT as indicated in the imported transaction objects. The implementation can be left blank in case it is desired to exchange configuration transactions on PCIe link only.

a. This mechanism is implemented via `cfg_database_service_req_analysis_imp` port

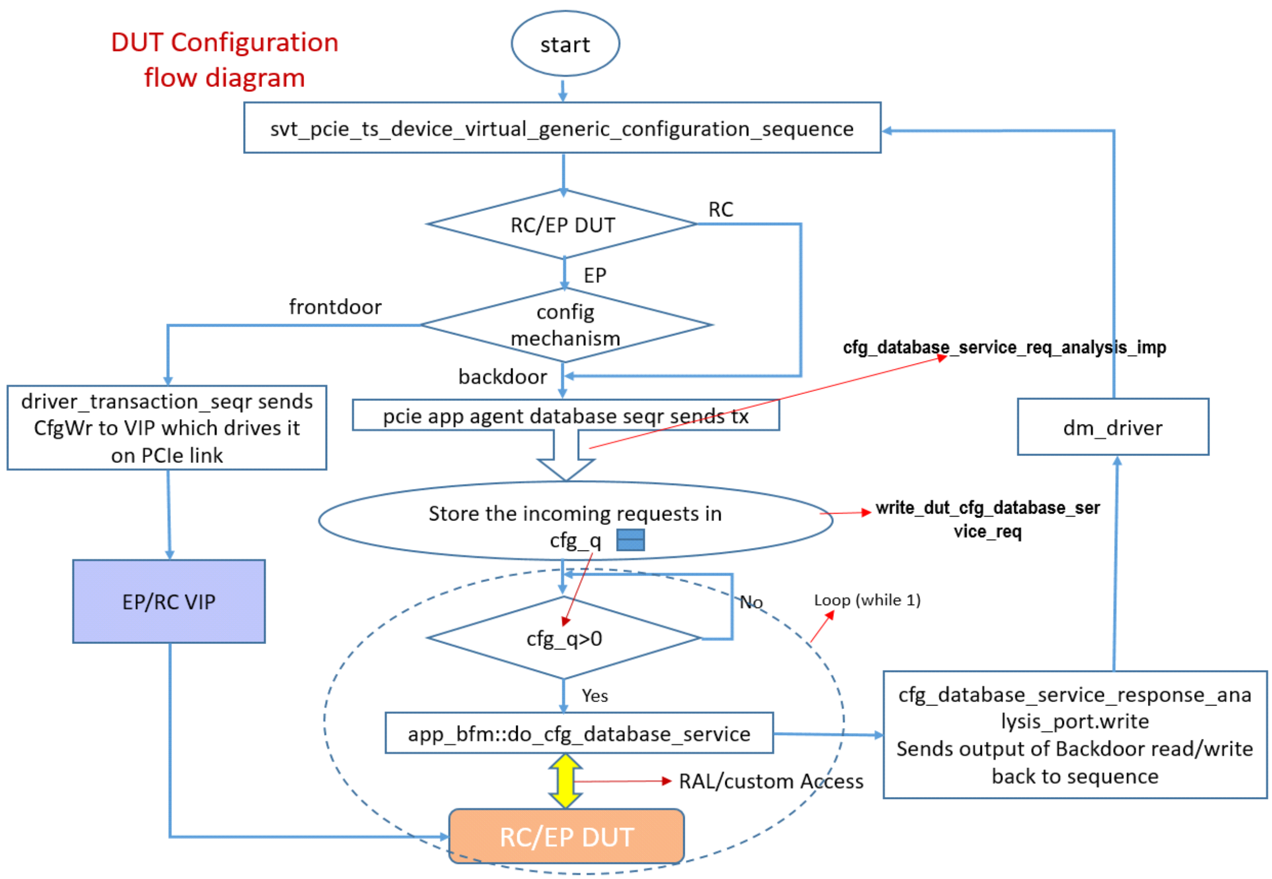
The `cfg_database_service_req_analysis_imp` port is used to import transaction items of type `svt_pcie_cfg_database_service` from the sequence.

The Application BFM should provide the `write_dut_cfg_database_service_req()` method for analysis port implementation (optional for EP DUT).

b. Application BFM must implement an analysis port `cfg_database_service_response_analysis_port` to drive response transactions of type `svt_pcie_cfg_database_service` is populated with DUT's response to the backdoor configuration access. You must write to this port with reference of the response transaction.

Figure 1-5 shows the flow diagram for writing into DUT registers.

Figure 1-5 DUT Configuration Flow Diagram



8. The DUT register access mechanism which is set via the test suite configuration class `svt_pcie_test_suite_configuration` attribute `enable_backdoor_cfg_updates` is read by the `virtual_generic_configuration` sequence and decides the configuration mechanism. For Front door access, the sequence initiates `CfgWr/Rd` TLP through VIP's `driver_trasaction` sequencer which sends it to the DUT via the PCIe link. In case of Backdoor access, the generic configuration sequence initiates a `svt_pcie_cfg_database_service` transaction on `dut_seqr.cfg_database_seqr` with `service_type` (read,write), `bdf, register_number, first_dw_be` and `payload` as shown in Example 1-1.

Example 1-1 Code Snippet: svt_pcie_ts_device_system_virtual_generic_configuration_sequence

```
// fire backdoor transaction to DUT Driver in DUT mode
`svt_xvm_do_on_with(backdoor_cfg_req,
dut_seqr.cfg_database_seqr, {

service_type == ((transaction_type ==
svt_pcie_driver_app_transaction::CFG_RD) ?
svt_pcie_cfg_database_service::READ_CFG_DWORD :
svt_pcie_cfg_database_service::WRITE_CFG_DWORD);

        function_num == bdf[7:0];
        dword_addr == register_number;
        byte_enables == first_dw_be;
        dword_data == payload;
      })
```

This transaction is picked by the `dm_driver` and put on the `cfg_database_service_req_analysis_imp` port of APP BFM. The Application BFM stores this request in a queue `cfg_database_service_req_queue`.

It checks for transactions in the queue and pass the `svt_pcie_cfg_database_service` transaction to the `do_cfg_database_service` method.

This method should take the `svt_pcie_cfg_database_service` transaction and convert it to DUT access type format to access the DUT registers. In the reference application BFM (*pcie_ep_dut_external_app_bfm.sv*/ *pcie_rc_dut_external_app_bfm.sv*), RAL is used to access DUT registers. For users not using RAL, they should convert the `svt_pcie_cfg_database_service` transaction to the DUT specific access type and sent to the DUT. The `svt_pcie_cfg_database_service` type to RAL conversion is shown in Example 1-2. This code is taken from the reference Application BFM.

Example 1-2 svt_pcie_cfg_database_service transaction to RAL conversion in Reference APP BFM::do_cfg_database_service method

```
svt_pcie_cfg_database_service::WRITE_CFG_DWORD : begin
     foreach(reg_q[i]) begin
       if(address == reg_q[i].get_address() && sucess == 0) begin
        `uvm_info(method_name,$sformatf("Writing to DUT register (%s) of
         function no %0d through
         RAL",reg_q[i].get_name(),xact.function_num),UVM_MEDIUM);
         reg_q[i].read(status,data, .map(reg_map));
         for(int idx = 0; idx < 4 ;idx ++)
         if(xact.byte_enables[idx] == 1'b1)  data[idx*8 +:8] =
         xact.dword_data[idx*8 +:8];
        reg_q[i].write(status,data, .map(reg_map));
       end
     end
   end
   svt_pcie_cfg_database_service::READ_CFG_DWORD  : begin
     foreach(reg_q[i]) begin
       if(address == reg_q[i].get_address() && sucess == 0) begin
        `uvm_info(method_name,$sformatf("Reading from DUT register (%s)
         of function no %0d through
         RAL",reg_q[i].get_name(),xact.function_num),UVM_MEDIUM);
         //Read from DUT
        reg_q[i].read(status,data,.map(reg_map));
        for(int idx = 0; idx < 4 ;idx ++)
         if(xact.byte_enables[idx] == 1'b1)   xact.dword_data[idx*8 +:8]
         = data[idx*8 +:8] ;
        `uvm_info(method_name,$sformatf("Value read back to VIP - DATA =
         0x%0x, address = 0x%0x",xact.dword_data, address),UVM_LOW);
       end
      end
```

The `svt_pcie_cfg_database_service` type transaction should be populated with DUT's response to the backdoor configuration accesses and write to `cfg_database_service_response_analysis_port` with reference of the response transaction, this takes the response back to the requesting sequence

1.5 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
TL	GEN-1	`demo_tl_gen1_linkup_followed_by_tlps-rc/ep`	This test case validates a few TLPs (MRD) flow from VIP to DUT and expects the completion for those in Gen1 speed.
TL	GEN-2	`demo_tl_gen2_speed_change_followed_tlps-rc/ep`	This test case validates a few TLPs (MRD) flow from VIP to DUT and expects the completion for those in Gen2 speed.

Table 1-1 Test Cases for Validating the Integration

Layer	Speed	Test Case	Description
TL	GEN-3	`demo_tl_gen3_speed_change_followed_tlps-rc/ep`	This test case validates a few TLPs (MRD) flow from VIP to DUT and expects the completion for those in Gen3 speed.
TL	GEN-2/3	`tl_rx_mem_mapped-rc/ep`	This test case validates a few TLPs (MRD) flow from VIP to DUT and expects the completions in response to each TLP.
TL	GEN-2/3	`tl_tx_mem_mapped-rc/ep`	This test case schedules memory writes and reads from the DUT. Simulation ends only when all TL traffic has completed.