Verification Continuum™

VC Verification IP USB FAQ

Version V-2023.09, September 2023

SYNOPSYS®

Contents

Preface

About This Manual

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

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:

1. Go to https://solvnetplus.synopsys.com/ and open a case.
   - ✦ Enter the information according to your environment and your issue.
   - ✦ For simulation issues, provide a UVM_FULL verbosity log file of the VIP instance and a VPD or FSDB dump file of the VIP interface.
2. 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.
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:

     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.

USB Frequently Asked Questions (FAQ)

1 How do I access the VIP documentation and example?

All PDF documents are available on SolvNetPlus (click here).

USB VIP documentation directory:

$DESIGNWARE_HOME/vip/svt/usb_svt/latest/doc

VC VIP for USB UVM User Guide (PDF):

$DESIGNWARE_HOME/vip/svt/usb_svt/latest/doc/PDFs/usb_svt_uvm_user_guide.pdf

VC VIP for USB Getting Started Guide (PDF):

$DESIGNWARE_HOME/vip/svt/usb_svt/latest/doc/PDFs/usb_svt_uvm_getting_started.pdf

VC VIP for USB Test Suite User Guide (PDF):

$DESIGNWARE_HOME/vip/svt/usb_test_suite_svt/latest/doc/PDFs/usb_test_suite_svt_uvm_user_guide.pdf

VC VIP for USB Test Suite Release Notes (PDF):

$DESIGNWARE_HOME/vip/svt/usb_test_suite_svt/latest/doc/PDFs/usb_test_suite_svt_release_notes.pdf

VC VIP for USB QuickStart (HTML):

*$DESIGNWARE_HOME/vip/svt/usb_svt/latest/examples/sverilog/tb_usb_svt_uvm_basic_sys/doc/
tb_usb_svt_uvm_basic_sys/index_basic.html*

VC VIP for USB Class Reference (HTML):

$DESIGNWARE_HOME/vip/svt/usb_svt/latest/doc/class_ref/usb_svt_uvm_class_reference/html/index.html

VC VIP for USB Test Suite Class Reference (HTML):

$DESIGNWARE_HOME/vip/svt/usb_svt/latest/doc/usb_test_suite_svt_uvm_reference/html/index.html

VC VIP for USB Verification Plans (XML):

`$DESIGNWARE_HOME/vip/svt/usb_svt/latest/doc/VerificationPlans`

Directory that contains a list of USB VIP example directories:

`$DESIGNWARE_HOME/vip/svt/usb_svt/latest/examples/sverilog`

1.1 What is the directory structure of the VIP?

cc: (contains files for configuration creator)

pa: (contains files for protocol analyzer)

examples: (contains VIP examples)

sverilog: (contains files for VIP models)

Verilog: (contains files for VIP models)

Doc: (contains VIP documents)

The following directories contain `sverilog/src/<simulator>`:

```
usb_link_svt
usb_physical_svt
usb_ssic_physical_svt
usb_agent_svt
usb_monitor_checker_svt
usb_protocol_svt
usb_subenv_svt
```

1.2 What are the primary classes of the VIP?

- Configuration classes: The USB VIP defines the following configuration classes:

  Agent configuration (svt_usb_agent_configuration): This class provides settings for the basic testbench capabilities.

  Device configuration (svt_usb_device_configuration): This class provides device information for an individual USB device.

  Host configuration (svt_usb_host_configuration): The protocol component uses the information to support its breaking of transfers into individual transactions.

  Endpoint configuration (svt_usb_endpoint_configuration): This uvm_data class provides endpoint information for an individual USB endpoint.

  Ustream configuration (svt_usb_ustream_resource_configuration): This uvm_data class contains information regarding a 'USB SS stream resource'.

- Service classes: All layer specific services are defined in the service class.

  Link Service (svt_usb_link_service): These objects represent USB link service commands requested by the link service commands that are put into the link component.

Physical Service (svt_usb_physical_service): These objects represent USB physical service commands.

Protocol Service (svt_usb_protocol_service): These objects represent protocol service commands that flow between the Protocol layer and the testbench. Commands that Protocol Service objects support include: LMP Transactions, LPM Transactions and SOF Commands

- Exception classes: All layer specific error injection parameters are defined in the exception class.

1.3 How can I modify agent configuration attribute during run time in USB VIP?

To modify the configuration attribute during run time to save the compile time, you can use `$value$plusargs`.

For example:

```
svt_usb_agent_configuration dev_cfg;
int prot_xml_gen=0;
if($value$plusargs("PRO_XML_GEN=%d",prot_xml_gen))
begin
$display("SNPS Protocol xml generation is enabled prot_xml_gen %0d ",prot_xml_gen);
end
dev_cfg.enable_prot_xml_gen = prot_xml_gen;
```

The problem with the above approach is, it is difficult to add similar logic for all configuration attributes.

To solve the above problem VIP supports built in method to change the configuration attribute during run time.

```
function void svt_sequence_item_base::set_prop_val_via_plusargs ( string
plusarg_keyword )
```

This method takes advantage of the set_prop_val method to load a set of property values based on a command line plusarg value.

Note
`plusarg_keyword` is used to identify the command line plusargs.

For example,

```
dev_cfg.set_prop_val_via_plusargs("DEV_CFG");
```

During run time:

```
./simv +UVM_TESTNAME=basic_20_serial +DEV_CFG="enable_prot_xml_gen:1,inject_idle_cnt:1
```

Note
This method is applicable only for agent configuration attribute.

In case the user wants to modify multiple attributes during the same time, it is recommended to create a configuration which is loaded to the test itself with the following plusargs:

```
       function void build_phase(uvm_phase phase);
     //    `uvm_info ("build_phase", "Entered...", UVM_LOW)
          svt_usb_agent_configuration cfg;
           cfg=new();
```

```
                super.build_phase(phase);
               if ($value$plusargs("validation_cfg_filename=%s", filename))
                begin
                     if (cfg.load_prop_vals(filename))
                      begin
        ....
        ....
```

Then during the run time you can apply the following code:

```
./simv +UVM_TESTNAME=basic_20_serial +validation_cfg_filename=my_test_usb.cfg
```

Note
Similarly you can enable Coverage, PA, trace files, timers, and update all in the my_test_usb.cfg and load it again during run time.

Example:

```
$DESIGNWARE_HOME/vip/svt/usb_svt/latest/examples/sverilog/tb_usb_svt_uvm_cfg_validator/
tests/ts.cfg_validate.sv
```

1.4 How can I reconfigure the VIP?

You can use the REFRESH_CFG service in the agent service class `(svt_usb_general_agent_service_sequence)` to reconfigure the VIP configuration attributes in the configuration class.

UVM Example:

```
svt_usb_general_agent_service_sequence
dev_agent_service_sequence;

dev_cfg.speed = svt_usb_types::SS;
dev_cfg.usb_ss_signal_interface = svt_usb_configuration::USB_SS_SERIAL_IF;

dev_cfg.local_device_cfg[0].device_address = 0;
dev_cfg.local_device_cfg[0].connected_bus_speed = svt_usb_types::SS;

svt_config_object_db
#(svt_usb_agent_configuration)::set(null,my_dev_agent.get_full_name(),"cfg",dev_cfg);


`svt_xvm_do_on_with(dev_agent_service_sequence,
p_sequencer.dev_virt_sequencer.agent_service_sequencer,
{service_type == svt_usb_agent_service::REFRESH_CFG; });
```

1.5 How do I access and modify configuration settings during a simulation?

A USB SVT VIP Agent's configuration settings can be read and/or modified after simulation has begun (time > 0).

Following is an example of reading/accessing a VIP Agent's configuration settings.

```
svt_configuration vip_svt_cfg;
svt_usb_agent_configuration vip_agent_cfg;
```

/** Step 1: Get VIP's Configuration (of svt_configuration type) using get_cfg() method

Note
svt_configuration is a base class for svt_usb_configuration, which in turn is a base class for svt_usb_agent_configuration

```
        env.host_agent.get_cfg(vip_svt_cfg);
```

/** Step 2: Cast the type from `svt_configuration` to `svt_usb_agent_configuration` */

```
if (!$cast(vip_agent_cfg,vip_svt_cfg.clone()))
`uvm_fatal("body", "Cast of VIP's svt_configuration to svt_usb_agent_configuration
failed")
```

Following is a code example showing how to modify a VIP Agent's configuration settings:

/** Step 1: Access VIP's current configuration using the two steps mentioned above */

/** Step 2: Modify members of the local configuration object as desired */

/** Following is a sample code for updating Device VIP's device address (in `transfer_ended` callback) upon receiving `SET_ADDRESS` control transfer */

```
vip_agent_cfg.local_device_cfg[0].device_address =
transfer.get_setup_data_w_value_val();
```

/** Step 3: reconfigure VIP */

```
env.device_agent.reconfigure(vip_agent_cfg);
```

Note
Both Host or Device configuration values can be updated as shown in the previous examples.

1.6 What is the usage of layer specific service classes?

USB VIP architecture is layered in accordance with USB Specifications. Layer specific service classes control the VIP operations according to functional definition of a feature as per Specification definitions in given layer. While using the VIP layer specific service classes it is required to know the relevant functionality of the layer that you intend to use in accordance with USB Specifications else incompatible service types does not work over VIP. These classes are used to initiate certain modes of operations.

For example:

To enable USB VIP `ping.LFPs` transmission required to use the service command `svt_usb_link_service`, where:

```
service_type == svt_usb_link_service::LINK_SS_PORT_COMMAND;
```

and

```
link_ss_command_type == svt_usb_link_service::USB_SS_TRANSMIT_PING_LFPS;
```

The following are classes with layer specific service definitions:

Layer	Service Definitions
Protocol Layer	svt_usb_protocol_service
Link Layer	svt_usb_link_service
Physical Layer	svt_usb_physical_service

1.7 What is the usage of status classes?

Layer specific status classes provide the protocol specific parameter status of each layer.

1.8 What are exceptions?

Exception classes are provided to inject error in specific fields of the associated transactions. This feature allows you to enable-1 or disable-0 big endian for data word before sending it to an analysis port.

1.9 What are the types of callback provided in the VIP and the usage of these callbacks?

Several callbacks are provided in each protocol layer. Virtual methods to override callbacks are defined in layer specific proxy classes. These virtual methods provide handles to received packets for validating protocol specific parameters in the packets, and handles to packets before transmission for injecting error.

1.10 What are the types of verbosity for messages issued by the VIP?

You can use the +uvm_set_verbosity option in the command line to specify the verbosity of the VIP messages.

The following is a mapping of the verbosity types between VIP and UVM.

VIP	UVM
`svt_fatal	UVM_FATAL
`svt_error	UVM_ERROR
`svt_warning	UVM_WARNING
`svt_note	UVM_INFO
`svt_trace	UVM_INFO
`svt_verbose	UVM_INFO

1.11 How to rename the default trace name?

USB VIP dumps the trace files, enabled during configuration.

Trace files contain information about the objects that have been transmitted across a particular port. There are different types of trace files, for example:

- Data trace files - Data objects (such as symbols) from physical (phys) component are available in the data trace files.
- Packet trace files - Packet objects from link component are available in packet trace files.
- Transaction trace files - Transaction objects from protocol (prot) component are available in transaction trace files.
- Transfer trace files - Transfer objects from prot component are available in packet, transaction, and transfer trace files.

In a typical VIP agent configuration, three components (protocol, link, and physical) are included with the SS operation, and the following trace files can be generated:

```
vip_agent.phys.SS.RX.data_trace
vip_agent.phys.SS.TX.data_trace
vip_agent.link.SS.RX.packet_trace
vip_agent.link.SS.TX.packet_trace
vip_agent.prot.SS.transaction_trace
vip_agent.prot.SS.transfer_trace
```

In a VIP agent configuration, a fourth component (phys representing remote PHY) is included, to generate two additional data trace files. The example names of the data trace files from the remote PHY are as follows:

```
vip_agent.remote_phys.SS.RX.data_trace
vip_agent.remote_phys.SS.TX.data_trace
```

For more information on trace files and how to enable them, see the Chapter 7 Troubleshooting section in the usb_svt_uvm_user_guide.pdf.

VIP dumps the trace files with default name.

Use the following method, to modify the file name:

```
/*Protocol layer Transfer report trace */
svt_usb_protocol_monitor_transfer_report_callback ddrvr_prot_xfer_report_cb;
svt_usb_protocol_monitor_transfer_report_callback hdrvr_prot_xfer_report_cb;
function void end_of_elaboration_phase(uvm_phase phase);
super.end_of_elaboration_phase(phase);
if(!$cast(ddrvr_prot_xfer_report_cb,dev_agent.prot_xfer_report_cb))
begin
      $display("SNPS $cast failed "); $finish;
end
if(!$cast(hdrvr_prot_xfer_report_cb,host_agent.prot_xfer_report_cb))
begin
       $display("SNPS $cast 1failed "); $finish;
end
// set_filename( default_trace_file_name, user_defined_file_name);
if(!ddrvr_prot_xfer_report_cb.local_xact_report.set_filename("env.dev_agent.prot_mon.SS
.xfer_trace","bulk_test-usb_dev_transfer_trace"))
begin
      $display("SNPS set_lone_filename failed "); $finish;
end

hdrvr_prot_xfer_report_cb.local_xact_report.set_filename("env.host_agent.prot_mon.SS.xf
er_trace","bulk_test-usb_host_transfer_trace");
endfunction
```

The above mentioned approach can be used for all the trace files from different layers.

```
/* Protocol layer  Service report  trace */
svt_usb_protocol_monitor_service_report_callback prot_svc_report_cb;
/* Protocol layer Transaction report  trace */
svt_usb_protocol_monitor_transaction_report_callback mon_prot_xact_report_cb;
/* Link layer service report  trace */
svt_usb_link_monitor_service_report_callback mon_link_svc_report_cb;
/*Link layer packet report  trace */
svt_usb_link_monitor_packet_report_callback mon_link_pkt_report_cb;
/* Physical layer data report  trace */
svt_usb_physical_monitor_data_report_callback mon_phys_data_report_cb;
/* Physical layer remote phy data report  trace */
```

```
svt_usb_physical_monitor_data_report_callback mon_remote_phys_data_report_cb;
/* Physical layer serivce report  trace */
svt_usb_physical_monitor_service_report_callback mon_phys_svc_report_cb;
```

1.12 How to set the maximum allowed UVM_ERROR count?

For more information, see the following SolvNetPlus article:

VC VIP: How to Set the Maximum Allowed UVM_ERROR Count for the USB?

1.13 How to set the maximum allowed error count while using VC VIP for USB in HDL Mode?

For more information, see the following SolvNetPlus article:

USB VIP: How to Set the Maximum Allowed Error Count in VC VIP for USB

1.14 How to check the USB VIP version and the available examples in an existing VIP installation using a single command?

Execute the following command to check the USB VIP version and the available examples in an existing VIP installation.

Command:

```
$DESIGNWARE_HOME/bin/dw_vip_setup -i home
```

Result:

The above command displays the following information on various Libraries, Licenses and Examples with their respective headers.

In the following usage note, the string next to usb_svt in LIBRARIES displays J-2014.12 as the usb_svt VIP version.

```
#----------------------------------------------------------------------#
# DesignWare VIP Setup; Copyright(C) 1994-2014 Synopsys, Inc. #
#----------------------------------------------------------------------#
#----------------------------------------------------------------------#
#
# Using svt version J-2014.12
#----------------------------------------------------------------------#
# LIBRARIES:
# common J-2014.12
# mphy_svt J-2014.12
# svt J-2014.12
# usb_svt J-2014.12
# MODELS:
# mphy_agent_svt J-2014.12
# mphy_monitor_svt J-2014.12
# mphy_mport_lm_agent_svt J-2014.12
# mphy_mport_lm_monitor_svt J-2014.12
# mphy_mport_lm_subenv_svt J-2014.12
# mphy_mport_lm_txrx_svt J-2014.12
# mphy_subenv_svt J-2014.12
# mphy_txrx_svt J-2014.12
# usb_agent_svt J-2014.12-T1125
# usb_link_svt J-2014.12-T1125
# usb_monitor_checker_svt J-2014.12-T1125
# usb_physical_svt J-2014.12-T1125
# usb_protocol_svt J-2014.12-T1125
# usb_ssic_physical_svt J-2014.12-T1125
# usb_subenv_svt J-2014.12-T1125
# EXAMPLES:
# mphy_svt/tb_mphy_lm_mport_svt_uvm_basic_sys
# mphy_svt/tb_mphy_model_lm_mport_uvm_sys
# mphy_svt/tb_mphy_svt_uvm_basic_sys
# usb_svt/tb_usb_svt_ovm_basic_sys
# usb_svt/tb_usb_svt_ovm_intermediate_sys
# usb_svt/tb_usb_svt_ovm_mon_chk_sys
# usb_svt/tb_usb_svt_uvm_basic_program_sys
# usb_svt/tb_usb_svt_uvm_basic_sys
# usb_svt/tb_usb_svt_uvm_cfg_validator
# usb_svt/tb_usb_svt_uvm_intermediate_sys
# usb_svt/tb_usb_svt_uvm_mon_chk_sys
# usb_svt/tb_usb_svt_vmm_advanced_sys
# usb_svt/tb_usb_svt_vmm_basic_sys
# usb_svt/tb_usb_svt_vmm_cfg_validator
# usb_svt/tb_usb_svt_vmm_intermediate_sys
# usb_svt/tb_usb_svt_verilog_sys
#----------------------------------------------------------------------#
# The 'info' operation has successfully completed.
#----------------------------------------------------------------------#
```

1.15 How can I demote a VIP error message in VMM for an expected error condition?

```
!ERROR![FAILURE] on svt_usb_subenv(device) at 215710ns:
Reset() - PROTOCOL still had transfer objects queued!
```

The above error message appears when VMM phase is moved from run_phase.

Though some of the transfers are queued in the VIP, the VIP active transfer voter opposes the testcase to end and hence the error is reported.

In some scenarios it is an expected error. If it is an expected error, then the user can demote the error message using the following code snippet:

```
class err_catcher extends vmm_log_catcher;
```

```
    int num_errors = 0;
    int max_errors;

     function new(int max_errors);
      this.max_errors = max_errors;
     endfunction

    // Modifies the message and severity if num of errors is less than
    // maximum number of acceptable errors
     virtual function void caught(vmm_log_msg msg);
       if (num_errors < max_errors)
        begin
         msg.text[0] = {"ACCEPTABLE ERROR: ", msg.text[0]};
         msg.effective_severity = vmm_log::WARNING_SEV;
         issue(msg);
         num_errors++;
        end
       else
        begin
         msg.text[0] = {"ACCEPTABLE ERROR: ", msg.text[0]};
         msg.effective_severity = vmm_log::WARNING_SEV;
         issue(msg);
            //    throw(msg); If required you can throw the message
        end
     endfunction
    endclass
   err_catcher ctcher;
In build_phase:
   catcher = new(0);
         void'(device.log.catch(ctcher,.severity(vmm_log::ERROR_SEV),.text("PROTOCOL
still has transfer")));
```

1.16 How to enable debug verbosity based logs for a particular USB VIP class at runtime command line?

User can use `+vip_verbosity=<class_name>:debug` runtime option

For example, user intend to debug the USB 20 timers then, pass

`+vip_verbosity=svt_usb_link_20_timer:debug to the VCS runtime command line`

1.17 Does VC VIP USB support PIPE4?

Yes, VC VIP USB supports PIPE4.

For more information, see the test case in `DESIGNWARE_HOME` example.

```
$DESIGNWARE_HOME/vip/svt/usb_svt/latest/examples/sverilog/
tb_usb_svt_uvm_intermediate_sys/tests/ts.base_ess_pipe4_test.sv
```

1.18 What is the PIPE4 version that VIP supports?

VC USB VIP supports PIPE4 version 4.2.

1.19 How can I disable USB VIP license check?

Sometimes it is necessary for the user to compile all the SVT VIP title files and select the required VIP during run time.

For example, if the user environment uses multiple SVT VIP titles but only one or few of the VIPs are used by the testcase, then the user may not perform license polling or license checkout for unused VIP title or titles.

1.20 Does USB SVT VIP support this feature?

Compile all the SVT VIP titles and use it in your test bench and disable license check during run time using the test bench code. If you do not want the license to be checked or polled for unused title then it is recommended to avoid instantiation of the VIP agent or subenv during run time.

For example, the USB SVT VIP tries to check out the license, only if svt_usb_agent or svt_usb_subenv is instantiated.

Example usage as follows:

```
int usb_en=0;
if($value$plusargs("USB_EN=%d",usb_en))
begin
  $display("IS USB TRAFFICS ENABLED %0d \n",usb_en);
end
if(usb_en)
begin
   $display("USB traffic is enabled for this test")
  dev_agent = new();
end
```

1.21 How to resolve errors or slowness in VIP license checkout?

There are many steps to check for issues related to VIP license checkout, all are mentioned in the following SolvNetPlus article link:

https://solvnetplus.synopsys.com/s/article/All-VC-VIPs-License-Checkout-Error-or-Slowness-1576091490927

1.22 How to assign the dedicated payload in svt_usb_packet? How to modify svt_usb_packet content inside VIP?

There is a SolvNetPlus article which explains how to assign the dedicated payload in `svt_usb_packet`:

https://solvnetplus.synopsys.com/s/article/USB-SVT-VIP-Missing-Payload-in-Received-SS-Data-Packet-1576091471283

The following methods can be used to ssign the dedicated payload in `svt_usb_packet`:

- Related Sequences :

```
svt_usb_transfer  xfer;
xfer.payload.data.delete();
xfer.payload.byte_count = 18;   // Please fill the expected total payload byte count here
xfer.payload.data = new[18];
xfer.payload.data_generation_algorithm = svt_usb_payload::USER_DEFINED_ALGORITHM;
// User can start to fill the expected data byte content to be any meaningful content
xfer.payload.data[0] = 'h00;
xfer.payload.data[1] = 'h01;
xfer.payload.data[2] = 'h02;
…
xfer.payload_intended_byte_count = xfer.payload.data.size();
// Then user can deliver this xfer to VIP for execution
```

- Used Callback method :

  User can modify USB 2.0 packet by using `pre_usb_20_packet_out_port_put()`

  User can modify USB 3.x packet by using `pre_usb_SS_packet_out_port_put()`

  The following example can be used to modify USB 2.0 packet in the protocol layer before delivering to link layer.

```
class modify_ctrl_read_data_callback extends svt_usb_protocol_callback;

   function new(string name);
       super.new(name);
  endfunction


     function void pre_usb_20_packet_out_port_put (svt_usb_protocol component ,int chan_id
, svt_usb_packet packet, svt_usb_transaction transaction, int packet_ix , svt_usb_transfer
transfer , int transaction_ix , ref bit drop);

        $display($time,,"pre_usb_20_packet_out_port_put : packet");
        packet.print();
        $display($time,,"pre_usb_20_packet_out_port_put : transaction");
        transaction.print();
        $display($time,,"pre_usb_20_packet_out_port_put : transfer");
        transfer.print();

         // User can set the conditions to modify the expected packet
        if (  (transfer.xfer_type == svt_usb_transfer::CONTROL_TRANSFER) &
               (transfer.setup_data_bmrequesttype_dir == svt_usb_types::DEVICE_TO_HOST) &
               (packet.payload.byte_count == 18) ) begin
            packet.payload.data[0]  = 'h97;
            packet.payload.data[1]  = 'h9f;
            packet.payload.data[2]  = 'h15;
            packet.payload.data[3]  = 'h7a;
            packet.payload.data[4]  = 'hce;
            packet.payload.data[5]  = 'h5b;
            packet.payload.data[6]  = 'h0e;
            packet.payload.data[7]  = 'h80;
            packet.payload.data[8]  = 'h83;
            packet.payload.data[9]  = 'hb2;
            packet.payload.data[10] = 'hd9;
            packet.payload.data[11] = 'hcd;
            packet.payload.data[12] = 'hd6;
            packet.payload.data[13] = 'he6;
            packet.payload.data[14] = 'h11;
            packet.payload.data[15] = 'h51;
            packet.payload.data[16] = 'h1a;
           packet.payload.data[17] = 'h67;
        end
    endfunction
endclass
```

1.23 Why do I see the following message at the beginning of the simulation when the simulator is loading the VIP instance?

```
Encountered SLI error 'Override feature not allowed to authorize'
License checkout failure. Authorization not granted for suite 'usb'.
```

This message is displayed because a special environment variable `DW_LICENSE_OVERRIDE` recognized by Synopsys VIP is being used. Check if this variable `DW_LICENSE_OVERRIDE` is set. If it is set and VIP is unable to use it to check out the required license, then the error message is displayed and simulation aborts or does not run. For example, if `DW_LICENSE_OVERRIDE="DesignWare-USB-VIP"` and the design has both PCIe and USB VIP, then the simulation aborts because the PCIe VIP did not obtain the required license. If the license for PCIe VIP is also available, then change the environment variable to `DW_LICENSE_OVERRIDE="DesignWare-USB-VIP DesignWare-PCIe-VIP"`.

Another example is for using VIPs where the environment variable must be set as `DW_LICENSE_OVERRIDE="DesignWare_Regression VIP_LIBRARY_SVT"`

Note
You can find the information about license feature names for VIP in the respective VIP release notes.

1. How to disable `svt_usb_agent` ?

   Use system reset link service to disable the agent ..

```
   svt_usb_link_service req;
    svt_usb_sequence_policy::sequence_policy_pre(sequence_policy);
     req = this.new_item("req");

req.service_type.rand_mode(0);
req.link_ss_command_type.rand_mode(0);
req.prereq_ltssm_state.rand_mode(0);
req.prereq_ltssm_substate.rand_mode(0);
req.ltssm_state.rand_mode(0);
req.ltssm_substate.rand_mode(0);
req.device_address.rand_mode(0);

`svt_xvm_do_with(req,    {service_type           ==
svt_usb_link_service::LINK_SS_PORT_COMMAND;
                           link_ss_command_type   ==
svt_usb_link_service::USB_SS_SYSTEM_RESET_ON ;
                           prereq_ltssm_state     == svt_usb_types::RX_DETECT;
                           prereq_ltssm_substate  == svt_usb_types::NO_SUBSTATE;
                           ltssm_state            == svt_usb_types::NO_LTSSM_STATE;
                           ltssm_substate         == svt_usb_types::NO_SUBSTATE;});
```

   System reset needs to be enabled using the following configuration before executing service sequence.

```
cfg.host_cfg.drive_initial_pipe_reset = 1'b1;
cfg.dev_cfg.drive_initial_pipe_reset = 1'b1;
```

1.24 How to enable Verdi+PA

```
setenv DESIGNWARE_HOME <VIP installation path>
setenv TRANSACTION_PROTOCOL_SUPPORT 1
```

compile option : `+define+SVT_FSDB_ENABLE`

Run time option : `+svt_enable_pa=fsdb`

For more information, see the USB UVM User Guide.

1.25 tb_usb_svt_uvm_basic_sys example testcase run not dumping fsdb with IUS by default. Is it expected?

Yes, the reason is `VERDI_HOME` needs to be added in `LD_LIBRARY_PATH`. Script does not define a specific `LD_LIBRARY_PATH`. To dump the fsdb add your `VERDI_HOME` PLI path to `LD_LIBRARY_PATH` in Makefile or `run_usb_svt_uvm_*_sys` script.

```
setenv LD_LIBRARY_PATH $VERDI_HOME/share/PLI/IUS/LINUXAMD64:$LD_LIBRARY_PATH
```

1.26 How to resolve VPI AGSOPRO and NAPROT error?

Set the following mentioned environment variable to resolve this error

```
setenv FSDB_EVENT_BEGIN_CALLSTACK 0
setenv FSDB_EVENT_END_CALLSTACK 0
ERROR:          VPI       AGSOPRO
```

Note

Cannot get string properties from objects within a protected region.

```
<$DESIGNWARE_HOME>/vip/svt/common/N-2017.12/sverilog/src/ncv/svt_vip_writer.svp,
1432: $fsdbTrans_begin(stream_id,"+type+transaction")
ERROR:          VPI       NAPROT
Not authorized to access protected object.
<DESIGNWARE_HOME>/vip/svt/common/N-2017.12/sverilog/src/ncv/svt_vip_writer.svp,
1432: $fsdbTrans_begin(stream_id,"+type+transaction")
```

1.27 Does Protocol analyzer supports performance analyzer?

Yes, to invoke performance analyzer please open fsdb with `-lca` option

```
-Verdi -ssf test.fsdb -lca
```

1.28 How to enable debug port?

```
cfg.host_cfg.enable_debug_ports =1;
cfg.dev_cfg.enable_debug_ports =1;
```

1.29 Does USB VIP support hub model?

Yes.

For more information on usage details, see the *$DESIGNWARE_HOME/vip/svt/usb_svt/latest/examples/sverilog/tb_usb_svt_uvm_basic_router_sys/* and *$DESIGNWARE_HOME/vip/svt/usb_svt/latest/doc/PDFs/usb_router_svt_uvm_user_guide.pdf*.

1.30 Is eUSB2 supported ?

Yes, USB SVT VIP supports e USB2. Draft Specification version 1.07 is supported.

2 USB 2.0

2.1 PROTOCOL LAYER

2.1.1 How can I program the Host VIP to generate SOF packets?

By default the Start Of Frame (SOF) generation is disabled in the HOST VIP. You should use protocol service commands and a channel or port to have the VIP generate the SOF. Protocol Service (`svt_usb_protocol_service`) objects represent protocol service commands that flow between the protocol layer and the testbench. One of the supported protocol service command is the SOF Command.

For more information, see the following SolvNetPlus article:

VC VIP: SOF Generation in USB

2.2 LINK LAYER

2.2.1 How can I bypass the 20 reset?

For more information, see the following SolvNetPlus article:

USB-SVT: How to Bypass or Skip the Setup Phase (speed negotiation) and Directly do Transfers from a Testcase?

2.2.2 What are the VIP timers used in 20 reset?

For more information, see the following SolvNetPlus article:

USB-SVT: Timers for High Speed Reset

2.2.3 How do I know the current 20 state of VIP?

VIP prints its 20 state by default.

VIP acting as A-Device (Host in non OTG mode) prints 20 state with the search string `L20_DEV_A_SM`.

<-: Current state <- Previous state

->: Current state -> Next state.

```
UVM_EXAMPLE:

UVM_INFO /vip/svt/usb_svt/4.35a/usb_link_svt/sverilog/src/vcs/
svt_usb_link_20_device_a_sm.svp(1564) @ 416600:
uvm_test_top.env.host_agent.link.usb_20_device_a [device_attached] L20_DEV_A_SM:
DEVICE_ATTACHED <- DISCONNECTED

UVM_INFO /vip/svt/usb_svt/4.35a/usb_link_svt/sverilog/src/vcs/
svt_usb_link_20_device_a_sm.svp(1756) @ 100441649:
uvm_test_top.env.host_agent.link.usb_20_device_a [device_attached] L20_DEV_A_SM:
DEVICE_ATTACHED -> RESETTING

VIP as B-Device(Device in non otg mode ) prints 20 states with search string
"L20_DEV_B_SM".

/vip/svt/usb_svt/4.35a/usb_link_svt/sverilog/src/vcs/
svt_usb_link_20_device_b_sm.svp(1650) @ 1089798073:
uvm_test_top.env.dev_agent.link.usb_20_device_b [receiving_is_state] L20_DEV_B_SM:
RECEIVING_IS <- BUS_RESET

UVM_INFO
```

```
UVM_INFO /vip/svt/usb_svt/4.35a/usb_link_svt/sverilog/src/vcs/
svt_usb_link_20_device_b_sm.svp(1650) @ 1100664305:
```

2.2.4 How can I control the duration for UTMI reset?

Use `utmi_reset_duration` configuration attribute to modify the duration.

```
dev_cfg.utmi_reset_duration = 1000;
```

real attribute

```
svt_usb_configuration::utmi_reset_duration =
SVT_USB_20_HS_FS_8_BIT_INTERFACE_UTMI_CLOCK_PERIOD*10
```

Note
The number of time the PHY drives the reset pin on the UTMI interface, the recommended value needs to be greater than 0.

2.2.5 How can I delay the VIP attachment?

VIP uses `poweron_auto_attach_delay` configuration attribute to delay the attachment. Before the delay VIP will be in un-connected state.

```
real  attribute
```

```
svt_usb_configuration::poweron_auto_attach_delay = 0
```

Determines the starting attachment state of the VIP. This value, which defines a delay when greater than 0, identifies one of three possible attachment startup conditions:

- If = 0 (the default), the VIP starts in an attached state.
- If > 0, the VIP starts in a detached state and automatically attaches following the delay specific by this value.
- If < 0, the VIP starts in a detached state and must be manually directed to an attached state using an `ATTACH_DEVICE` physical service command.

2.3 PHYSICAL LAYER

2.3.1 Is there any specific clocking requirement for using VIP in the 20 serial interface?

Yes, the testbench is expected to provide the VIP with an input clock running at a base frequency of 1,920 Mhz (four times the fundamental HS clock frequency of 480 Mhz).

2.3.2 What is the signal strength levels used, for the VIP 20 serial interface?

The VIP models the DP and DM signals using wired-or outputs in combination with 9-state logic.

The following VIPs drive the strength properties:

- HS drive strength on DP/DM is "supply"
- FS/LS drive strength on DP/DM is "strong"
- Pull-down strength on DP/DM is "weak"
- Pull-up strength on DP/DM is "pull"
- DP/DM signals declared as "wor"

Note
For proper simulation, the DUT/testbench is recommended to use the same set of criteria for modeling the DP/DM signals. If the set of criteria does not match the signaling criteria, then the simulation anomalies are reported.

2.3.3 How to remove `UVM_FATAL` error related to interface ?

`UVM_FATAL remote_cfg` specified with `usb_20_signal_interface = UTMI_IF` but `usb_20_if` not provided with the config db or `remote_cfg`.

The fatal error is due to interface not assigned to virtual interface of the agent. To resolve this error you need to assign actual interface to virtual interface.

```
uvm_config_db#(svt_usb_if)::set(this, "host_agent", "usb_20_if", this.host_usb_if);
```

2.3.4 Which timer controls the Host VIP downstream resume duration?

Tdrsmdn is the timer used by the VIP to control the downstream resume duration.

```
real  attribute  svt_usb_configuration::tdrsmdn = SVT_USB_TDRSMDN_MIN
```

Note
Duration of driving resume to a downstream port

2.3.5 How to program VIP to tolerate extra EOP bits sent by the DUT? By default VIP looks for 8 eop bits. If it receives more than 8, then reports `UVM_ERROR`. However, it is acceptable for DUT to send more than 8 bits. How to solve this error?

Add the following mentioned macro in your compile script it will modify the default 8 eop bits 9 eop bits.

```
+define+SVT_USB_20_USER_MAX_HS_EOP_LENGTH=9
```

2.3.6 Is it possible to modify the drive strength of dp/dm during reset and connection?

Yes, you can modify the default drive strength mentioned in 2.3.2 by using the following mentioned user defines.

```
 `define SVT_USB_DRIVE_STRENGTH_USER_DEFINED_PULLDOWN0  <weak0/supply0 etc..>
 `define SVT_USB_DRIVE_STRENGTH_USER_DEFINED_PULLDOWN1  <weak0/supply0 etc..>


 `define SVT_USB_DRIVE_STRENGTH_USER_DEFINED_PULLUP0  <weak0/supply0 etc..>
 `define SVT_USB_DRIVE_STRENGTH_USER_DEFINED_PULLUP1  <weak0/supply0 etc..>
 `define SVT_USB_DRIVE_STRENGTH_USER_DEFINED_HS_TERMINATION0 <weak0/supply0 etc..>
`define SVT_USB_DRIVE_STRENGTH_USER_DEFINED_HS_TERMINATION1 <weak0/supply0 etc..>
```

2.3.7 Which timer controls the upstream port resume duration?

Tdrsmup timer is used by the device VIP to drive the resume.

```
real  attribute
```

```
svt_usb_configuration::tdrsmup = 1ms
```

Note
Duration of driving resume upstream

2.3.8 What is the timer name for usb 2.0 protocol reset?

Tdrst is the timer used by the host VIP to control protocol reset duration.

```
real  attribute
svt_usb_configuration::tdrst = 10ms
```

Note
Duration of driving reset to a downstream facing port.

2.3.9 How to change the default sync length?

Default sync length of the VIP can be changed using `usb_20_hs_sync_length_min` configuration attribute:

- `cfg.dev_cfg.usb_20_hs_sync_length_min =30;`
- `cfg.dev_cfg.usb_20_fs_sync_length_min =4;`
- `cfg.dev_cfg.usb_20_ls_sync_length_min =4;`

2.3.10 How to change the default EOP length?

Use macro `SVT_USB_20_USER_MAX_HS_EOP_LENGTH` to define the max HS EOP length

```
+define+SVT_USB_20_USER_MAX_HS_EOP_LENGTH=9
```

3 USB 3.0

3.1 PROTOCOL LAYER

3.1.1 How can I modify the device VIP data response?

For more information, see the USB-SVT: Modifying Device Data Responses SolvNetPlus article.

USB-SVT: Modifying Device Data Responses

3.1.2 How can I modify the device default ACK response?

For more information, see the USB-SVT: USB 2.0 Device VIP Response Modification SolvNetPlus article.

USB-SVT: USB 2.0 Device VIP Response Modification

3.1.3 Does VIP support concurrent multi end points bulk in and out transfers?

Yes, VIP by default supports no special handling is needed.

3.1.4 Need sequence names for some error sceanrios?

```
Disparity error:
svt_usb_20_na_30_ss_link_errors_received_dpp_10_bit_disparity_error_system_virtual_sequ
ence
CRC-5 error:
svt_usb_20_hs_fs_30_na_bulk_in_crc5_error_system_virtual_sequence
CRC-16 error:
svt_usb_20_hs_fs_30_na_bulk_in_crc16_error_system_virtual_sequence
HSEQ# error:
svt_usb_20_na_30_ss_link_errors_rx_hp_hseq_and_rx_hdr_seq_num_mismatch_error_system_vir
tual_sequence
CRC-32 error:
svt_usb_20_na_30_ss_bulk_in_dpp_crc32_error_system_virtual_sequence
DPP abort:
svt_usb_protocol_service_abort_current_transfer_sequence ( Need to execute this
sequence protocol service sequencer.)
DPP missing :
svt_usb_20_na_30_ss_bulk_out_invalid_dpp_due_to_dpp_missing_random_endpoint_system_virt
ual_sequence
```

3.1.5 Does VC VIP USB support dynamic reconfiguration?

Yes, you can dynamically reconfigure the VIP to change the device address, number of devices, endpoints and so on. It will not modify the existing device behavior, if the device address is not modified.

3.1.6 How to send next TP_ACK with Retry bit set when Host VIP sends a TP_ACK with Rty=1 and NumP=0 to end burst and request a retired DP?

A callback is used to modify Rty bit of USB 3.x transaction by using `randomized_transaction()`, the following example specifies Rty bit of USB 3.x transaction in the protocol layer before delivering to the link layer:

```
class retry_ack_callback extends svt_usb_protocol_callback;
```

```
      function new(string name);
        super.new(name);
      endfunction

      function void randomized_transaction ( svt_usb_protocol component , svt_usb_transfer
    transfer ,
                                  int transaction_ix ,
                      svt_usb_types::protocol_randomization_point_enum rand_point )

        svt_usb_transaction  xact;
        foreach( transfer.implementation[i] )begin
        $cast(xact, transfer.implementation[i]);
        if( xact.last_pkt_sent != null && xact.last_pkt_sent.tp_subpacket_type ==
    svt_usb_packet::TP_ACK &&
         (xact.last_pkt_sent.rty_bit == 1 || xact.resend_ack_after_retry_ack == 1 )) begin
           xact.rty_bit_0_in_resend_of_ack_after_retry_ack = 0;
           $display($time, "randomized_transaction:
                  Set xact.rty_bit_0_in_resend_of_ack_after_retry_ack = 0");
        end
      endfuncation

    endclass
```

3.2 LINK LAYER

3.2.1 How can I bypass the link training?

By configuring VIP initial ltssm state as U0 will bypass the link training and directly stat from U0 state.

```
host_cfg.usb_ss_initial_ltssm_state = svt_usb_types::U0;
dev_cfg.usb_ss_initial_ltssm_state = svt_usb_types::U0;
```

3.2.2 How do I know the current LTSSM state of VIP?

Search the LTSSM state is string in the log.

```
UVM_INFO /vip/usb_svt/usb_link_svt/src/svt_usb_link_ss_ltssm.sv(7806) @ 151156200:
uvm_test_top.env.dev_agent.link.ss.ltssm [Polling.Idle] LTSSM state is Polling.Idle
UVM_INFO /vip/usb_svt/usb_link_svt/src/svt_usb_link_ss_ltssm.sv(7806) @ 151212200:
uvm_test_top.env.host_agent.link.ss.ltssm [Polling.Idle] LTSSM state is Polling.Idle
UVM_INFO /vip/usb_svt/usb_link_svt/src/svt_usb_link_ss_ltssm.sv(13695) @ 151284200:
uvm_test_top.env.host_agent.link.ss.ltssm [u0_state] LTSSM state is U0
UVM_INFO /vip/usb_svt/usb_link_svt/src/svt_usb_link_ss_ltssm.sv(13695) @ 151292200:
uvm_test_top.env.dev_agent.link.ss.ltssm [u0_state] LTSSM state is U0
```

3.2.3 How can I skip the receiver equalization training?

TSEQ receiver equalization training sequence (65536 times TSEQ ordered set) used by the receiver, trains the receiver equalizer. VIP supports skipping this for faster simulation.

You can skip by configuring VIP to skip.

```
cfg.dev_cfg.ltssm_skip_polling_rxeq = <value>; // 0 -don not skip 1- skip
cfg.host_cfg.ltssm_skip_polling_rxeq = <value>; // 0 -don not skip 1- skip
```

3.2.4 What are all the timers used in LTSSM state transitions?.

For more information, see the ts.basic_additional_ss_ltssm.sv test in DESIGNWARE_HOME. It describes all the timers and configurations related to LTSSM state transitions.

```
$DESIGNWARE_HOME/vip/svt/usb_svt/latest/examples/sverilog/tb_usb_svt_uvm_basic_sys/
tests/ts.basic_additional_ss_ltssm.sv
```

3.2.5 How to debug LTSSM state stuck in one state problem?

VIP is configured for timers for the correct LTSSM state transitions. Sometimes VIP LTSSM state is stuck in one state.

For example: VIP LTSSM state is not moving from Polling to Active. You can debug this issue by enabling VIP LTSSM class verbosity. The timer may have issues if the VIP does not receive the expected pattern from the DUT.

```
+vip_verbosity=svt_usb_link_ss_ltssm:verbose,,svt_usb_link_ss_ltssm_base:verbose
```

3.2.6 How to inject 8b/10b disparity error?

For more information, see the USB VIP: Inject 8b/10b Disparity Error in VC VIP for USB UVM Environment SolvNetPlus article.

USB VIP: Inject 8b/10b Disparity Error in VC VIP for USB UVM Environment

3.2.7 Is there any general usage article on the inter packet delays?

For more information, see the following SolvNetPlus article:

USB SVT VIP: Inter Packet Delays

3.2.8 How to override the end-to-end delays?

For more information, see the following SolvNetPlus article:

USB-SVT: Overriding Packet Delay Timer Value

3.2.9 When can I ignore `skp_symbol_ratio_high_check` or `skp_symbol_ratio_low_check` error in USB VIP?

As per Section 6.4.3 Elasticity Buffer and SKP Ordered Set, SKP Ordered Sets can be used to reconciledreconcile for frequency differences between the two ends of the link. The transmitter sends SKP ordered sets at an average of every 354 symbols.

Link-layer-checks `skp_symbol_ratio_high_check` and `skp_symbol_ratio_low_check` are included in the USB VIP to check if there is a violation of Section 6.4.3 specification. In case of a violation, these checks will result in errors during simulation. However, for the following topologies, the user may choose to demote or ignore the `skp_symbol_ratio_high_check` and `skp_symbol_ratio_low_check` errors.

As per Section 7.14 Clock Tolerance Compensation of the PIPE3 specification, the PHY may remove or add a SKP symbol for Clock Correction. So, the SKP-symbol ratio on the RX side (`usb_agent.link.link_ss_rx`), may differ from that on the TX side (DUT). Also, it is possible that the SKP-symbol ratio on the RX side can be lower or higher than 354 depending on removal or addition of SKP by PHY.

In such case, the user shall not consider the SKP-symbol ratio error as an issue with VIP or DUT. The user may safely demote this UVM_ERROR as it has occurred on the RX side.

Example Error Message:

```
UVM_ERROR:  uvm_test_top.tb.usb_module.usb_agent.link.link_ss_rx [register_fail:LINK
RECEIVER:skp_symbol_ratio_high_check]
```

```
Description: Check if the SS Rx calculates an appropriate SKP to other-symbol ratio.
Reference: USB3.0 Spec v1.0: 6.4.3 - DEVICE Super-Speed Rx Skip-symbol ratio is not
correct.
```

3.2.10 Can I control SKP symbol interval?

Yes, by default VIP inserts SKP symbols after every 354 symbols. You can override this value, using the following mentioned configuration attribute:

```
dev_cfg. link_ss_tx_symbols_per_skp_pair = 400;
```

int unsigned attribute

```
svt_usb_configuration::link_ss_tx_symbols_per_skp_pair =
SVT_USB_LINK_SS_NOMINAL_SYMBOLS_SENT_PER_SKP_PAIR
```

Note Controls the ratio of SKP ordered sets, for the other-symbols produced by the VIP super-speed link transmitter. This defaults to the specification value, but can also be customized.

3.2.11 What is the Difference Between receiver_detect_time and rx_detect_quiet_timeout in USB?

With reference to USB3.0 Specification Section 6.11 and Section 7.5.3.7, the timer attributes `receiver_detect_time` and `rx_detect_quiet_timeout` have been defined in the USB_SVT VIP respectively.

Note The `receiver_detect_time` is the time required for the PHY to perform receiver detection.

For more information, see the following solvet article:

VC VIP: What is the Difference Between receiver_detect_time and rx_detect_quiet_timeout in USB?

3.2.12 How can I direct a VIP to move LTSSM state from SS.Disabled to Rx.Detect?

Use built in link service sequence to direct the VIP to LTSSM state to move from SS.Disable to Rx.Detect

```
svt_usb_link_service_ss_disabled_rx_detect_sequence
dev_ss_disabled_rx_detect_sequence;
```

3.3 Is dynamic reconfiguration of LTSSM parameters supported by VIP?

Yes, during the middle of simulation Execute system reset ("How to enable Verdi+PA") before changing the ltssm parameter.

For example,

```
wait for ltssm reach U0
execute system reset
change ltssm timing parameter
refresh the configuration ..
```

3.3.1 How to change the default value of PIPE reset?

```
cfg.host_cfg.drive_initial_pipe_reset =0;
bit  attribute
svt_usb_configuration::drive_initial_pipe_reset = 1
```

This variable determines the value to be driven on `Reset_n` initially When set to 1, the `Reset_n` is driven high When set to 0, the `Reset_n` is drops down until the `drive_reset_time` timer expires.

3.3.2 How to control pipe reset duration?

```
cfg.host_cfg.drive_reset_time =10us;
real  attribute
svt_usb_configuration::drive_reset_time = SVT_USB_PHYSICAL_DRIVE_RESET_TIME
```

Note
Length of time Reset(l) is asserted

3.3.3 How to change default electrical idle value?

```
cfg.dev_cfg.tx_elec_idle_val = svt_usb_types::LINESTATE_SE0;
rand svt_usb_types :: linestate_value_enum  attribute
svt_usb_configuration::tx_elec_idle_val = svt_usb_types::LINESTATE_Z
```

Note Transmit Electrical Idle value -- Any of the three valid values can be selected to tansmit as Electrical Idle. The valid values are Z, SE0, SE1 Default value is Z. On the Receive path if any of the values are detected as electrical idle at all times

3.4 PHYSICAL LAYER

3.4.1 Does VC VIP USB support automatic jitter behavior?

Yes, you can enable the jitter using the following configuration attributes:

```
cfg.dev_cfg.enable_tx_clock_jitter = 1'b1 ;
// Enables jitter in tx clock
cfg.dev_cfg.jitter_type = svt_usb_configuration::CONSTANT_NEGATIVE_JITTER;
//You can change it to POSTIVE or RANDOM
cfg.dev_cfg.number_of_clocks_with_jitter = 12'd10 ;
cfg.dev_cfg.number_of_clocks_without_jitter = 12'd0 ;
//Always Generates clock with jitter
```

4 USB 3.1

4.1 PROTOCOL LAYER

4.2 LINK LAYER

4.2.1 How can I bypass the link training?

See Question 3.1.1

4.2.2 How do I know the current LTSSM state of VIP?

See Question 3.1.2

4.2.3 For SCD1/SCD2 Polling, LFPS, Tburst is 80 ns, but spec defines 0.6 ~ 1.4 us and Trepeat is

0.7us and 1.2us, but spec defines (6 ~ 9us) and (11 ~ 14 us), why?

In the example Test case we have shorten the values to save the simulation Time. And the default values of these SCD1/SCD2 are as per spec.

4.2.4 Does the VIP support the PIPE4 signal, BlockAlignControl

No not yet.

4.2.5 Do we support Lane polarity inversion for Gen 1 and Gen 2 operation?

For Lane polarity inversion in Gen1 operation: We have this 3.0 test, wherein Host VIP is sending TSEQ OS with inverted Identifier so that DUT upon receiving it senses the reverse polarity and then after keeps on inverting the bits which it is receiving.

This don't require the connecting the signals inversely. Please check by using this `test.usb_20_na_30_ss_polarity_inversion_detection_using_tseq`

For Lane polarity inversion in Gen2 operation: The inverted polarity is detected using SYNC Ordered sets.

4.2.6 Which timer controls the LBPM timings?

```
Tx path : polling_lfps_pwm_time
Rx path :  polling_lfps_pwm_min and polling_lfps_pwm_max
```

4.3 PHYSICAL LAYER

4.3.1 Why VIP does not drive the PIPE4 Wdith[1:0]?

The PIPE4 Width Signal would not be driven in case of USB3.1, only the DataBusWidth shall be driven and set to 0 if width is 32 bit.

The PIPE4 spec says, this signal shall be implemented and driven only if port is capable of changing the signaling rate on selected DataBusWidth, that is, if 3.1 MAC is capable of deciding that it wants to use, only 8 bits or 16 bits out of the 32 bit wide PIPE then we expect this signal to be set to 0(for 8bits) or 1(for 16 bits).

If we suppose the DataBusWidth is selected as 32 lanes means in 1 PCLK 32 bits can go but 3.1 MAC decides to send only 8 or 16 bits i.e it decides to use only 8 or 16 lanes then Width[1:0] should be 0 or 1.

But 3.1 MAC doesn't have this flexibility of choosing lanes out of total available lanes, may be PCIe Gen3 has that.

Note
So we should not expect Width to be driven.

4.3.2 What is the difference between 3.0 Host PHY and Device PHY?

The difference in 3.0 is with respect to VBUS and we have additional scenarios at Device DUT side that verifies whether device DUT is able to detect absent VBUS in any power state P0, P1, P2 or P3.

4.3.3 Is there any callback mechanism or configuration that provides VIP's view on LFPS signaling?

No but VIP has checks (some of them enlisted below) present in both active and passive that monitors TX and RX LFPS and shouts if VIP sees wrong LFPS.

`unrecognized_lfps_check`: This check fails when VIP sees the received LFPS not falling under valid state specific LFPS, for example, if VIP does not receive a valid U1/U2 exit LFPS while in U1 state then this check shall fire stating the expected and actual received LFPS."`lfps_handshake_failed_check`","`u1/u2/u3_exit_signaling_failed_check`","`ss_disabled_lfps_reception_disabled_check`" ...

4.3.4 What are the configurations that you need to consider for SCD1/SCD2 transmission/ reception?

- To control the VIP SCD1/2 transmit:

```
polling_lfps_scd_burst_time
polling_lfps_scd_logic0_repeat_time
polling_lfps_scd_logic1_repeat_time
polling_lfps_scd_end_repeat_time
```

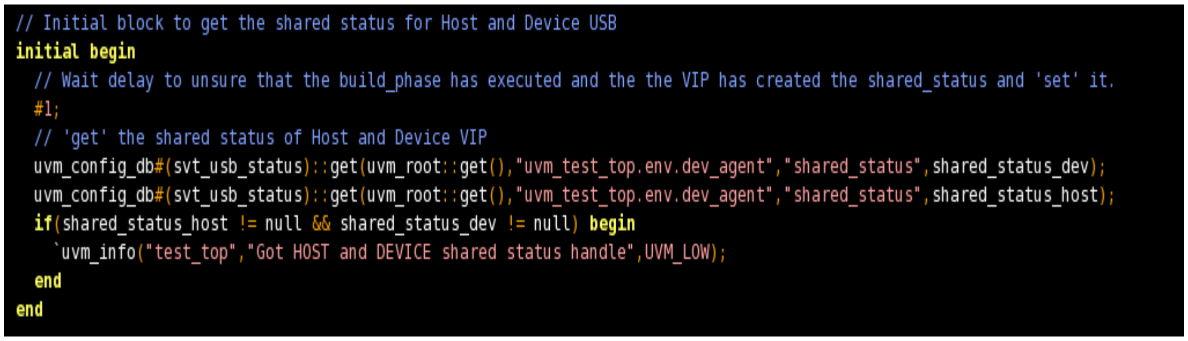
- To control the VIP SCD1/2 receive:

```
polling_lfps_scd_burst_min
polling_lfps_scd_burst_max
polling_lfps_scd_logic0_repeat_min
polling_lfps_scd_logic0_repeat_max
polling_lfps_scd_logic1_repeat_min
polling_lfps_scd_logic1_repeat_max
```

4.3.5 Do we support the following “RX detection model? *“When DUT RX Term is on, there is a “verilog weak 0" assigned on the VIP's TX differential pair. When DUT RX Term is off, the VIP's TX differential pair is floating (High-Z) The VIP should check the status of TX differential pair and do RX detection as Spec define”*

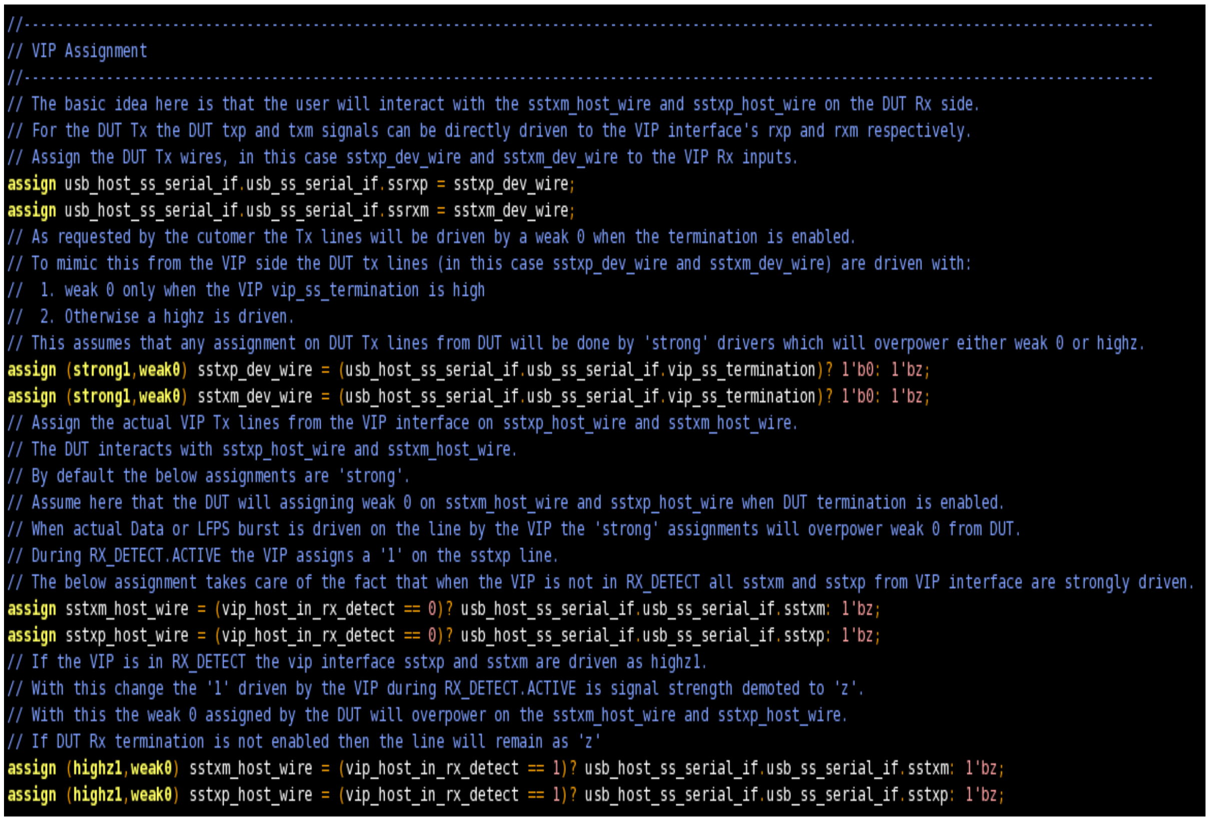
Yes, by breaking the strategy in three parts:

1. Getting the `shared_status` in `test_top` to monitor the LTSSM state



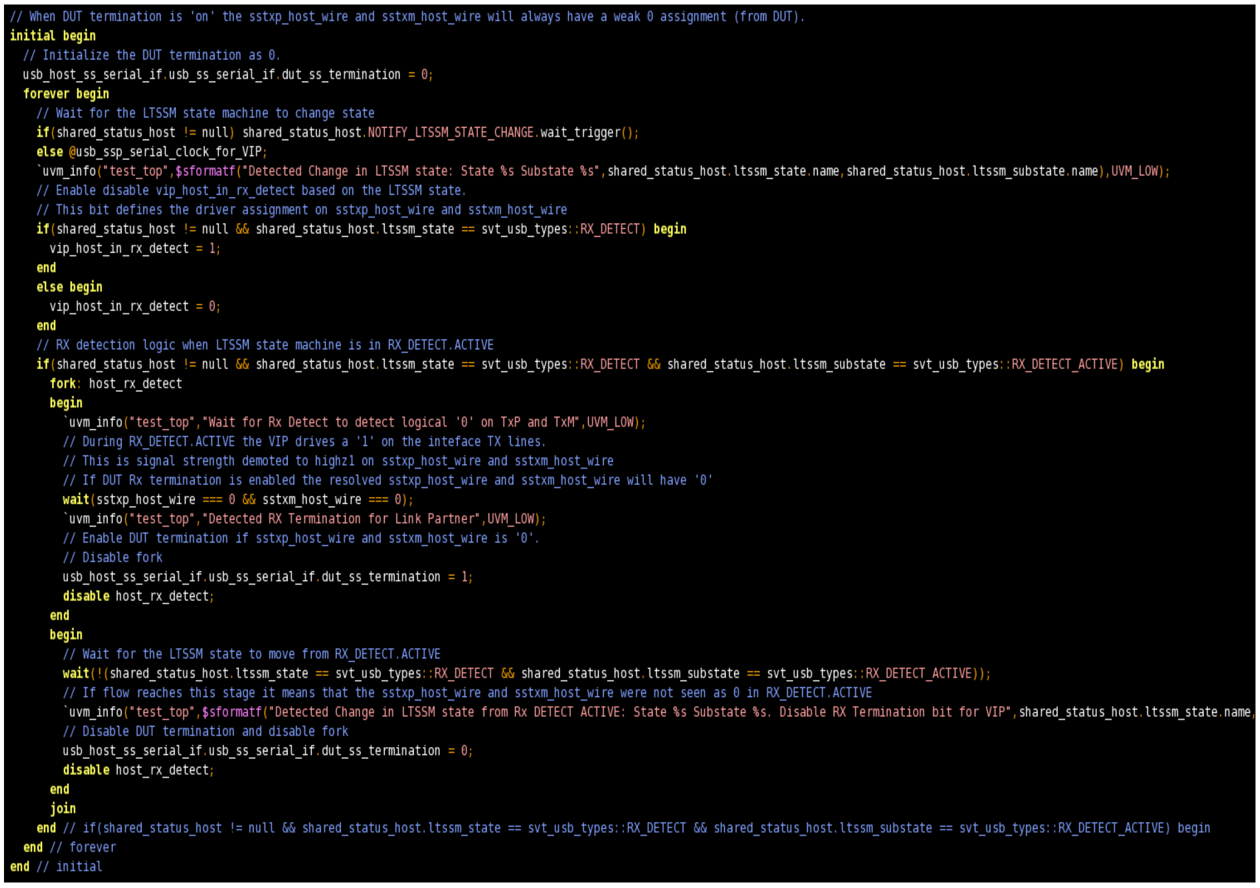
```
// Initial block to get the shared status for Host and Device USB
initial begin
  // Wait delay to unsure that the build_phase has executed and the the VIP has created the shared_status and 'set' it.
  #1;
  // 'get' the shared status of Host and Device VIP
  uvm_config_db#(svt_usb_status)::get(uvm_root::get(),"uvm_test_top.env.dev_agent","shared_status",shared_status_dev);
  uvm_config_db#(svt_usb_status)::get(uvm_root::get(),"uvm_test_top.env.dev_agent","shared_status",shared_status_host);
  if(shared_status_host != null && shared_status_dev != null) begin
    `uvm_info("test_top","Got HOST and DEVICE shared status handle",UVM_LOW);
  end
end
```

2. Defining the signal strength drivers for the tx lines which the DUT Rx interacts with



```
//-------------------------------------------------------------------------------------------------------------------------------------------
// VIP Assignment
//-------------------------------------------------------------------------------------------------------------------------------------------
// The basic idea here is that the user will interact with the sstxm_host_wire and sstxp_host_wire on the DUT Rx side.
// For the DUT Tx the DUT txp and txm signals can be directly driven to the VIP interface's rxp and rxm respectively.
// Assign the DUT Tx wires, in this case sstxp_dev_wire and sstxm_dev_wire to the VIP Rx inputs.
assign usb_host_ss_serial_if.usb_ss_serial_if.ssrxp = sstxp_dev_wire;
assign usb_host_ss_serial_if.usb_ss_serial_if.ssrxm = sstxm_dev_wire;
// As requested by the cutomer the Tx lines will be driven by a weak 0 when the termination is enabled.
// To mimic this from the VIP side the DUT tx lines (in this case sstxp_dev_wire and sstxm_dev_wire) are driven with:
//  1. weak 0 only when the VIP vip_ss_termination is high
//  2. Otherwise a highz is driven.
// This assumes that any assignment on DUT Tx lines from DUT will be done by 'strong' drivers which will overpower either weak 0 or highz.
assign (strong1,weak0) sstxp_dev_wire = (usb_host_ss_serial_if.usb_ss_serial_if.vip_ss_termination)? 1'b0: 1'bz;
assign (strong1,weak0) sstxm_dev_wire = (usb_host_ss_serial_if.usb_ss_serial_if.vip_ss_termination)? 1'b0: 1'bz;
// Assign the actual VIP Tx lines from the VIP interface on sstxp_host_wire and sstxm_host_wire.
// The DUT interacts with sstxp_host_wire and sstxm_host_wire.
// By default the below assignments are 'strong'.
// Assume here that the DUT will assigning weak 0 on sstxm_host_wire and sstxp_host_wire when DUT termination is enabled.
// When actual Data or LFPS burst is driven on the line by the VIP the 'strong' assignments will overpower weak 0 from DUT.
// During RX_DETECT.ACTIVE the VIP assigns a '1' on the sstxp line.
// The below assignment takes care of the fact that when the VIP is not in RX_DETECT all sstxm and sstxp from VIP interface are strongly driven.
assign sstxm_host_wire = (vip_host_in_rx_detect == 0)? usb_host_ss_serial_if.usb_ss_serial_if.sstxm: 1'bz;
assign sstxp_host_wire = (vip_host_in_rx_detect == 0)? usb_host_ss_serial_if.usb_ss_serial_if.sstxp: 1'bz;
// If the VIP is in RX_DETECT the vip interface sstxp and sstxm are driven as highz1.
// With this change the '1' driven by the VIP during RX_DETECT.ACTIVE is signal strength demoted to 'z'.
// With this the weak 0 assigned by the DUT will overpower on the sstxm_host_wire and sstxp_host_wire.
// If DUT Rx termination is not enabled then the line will remain as 'z'
assign (highz1,weak0) sstxm_host_wire = (vip_host_in_rx_detect == 1)? usb_host_ss_serial_if.usb_ss_serial_if.sstxm: 1'bz;
assign (highz1,weak0) sstxp_host_wire = (vip_host_in_rx_detect == 1)? usb_host_ss_serial_if.usb_ss_serial_if.sstxp: 1'bz;
```

3. The DUT receiver detection logic



```
// When DUT termination is 'on' the sstxp_host_wire and sstxm_host_wire will always have a weak 0 assignment (from DUT).
initial begin
  // Initialize the DUT termination as 0.
  usb_host_ss_serial_if.usb_ss_serial_if.dut_ss_termination = 0;
  forever begin
    // Wait for the LTSSM state machine to change state
    if(shared_status_host != null) shared_status_host.NOTIFY_LTSSM_STATE_CHANGE.wait_trigger();
    else @usb_ssp_serial_clock_for_VIP;
    `uvm_info("test_top",$sformatf("Detected Change in LTSSM state: State %s Substate %s",shared_status_host.ltssm_state.name,shared_status_host.ltssm_substate.name),UVM_LOW);
    // Enable disable vip_host_in_rx_detect based on the LTSSM state.
    // This bit defines the driver assignment on sstxp_host_wire and sstxm_host_wire
    if(shared_status_host != null && shared_status_host.ltssm_state == svt_usb_types::RX_DETECT) begin
      vip_host_in_rx_detect = 1;
    end
    else begin
      vip_host_in_rx_detect = 0;
    end
    // RX detection logic when LTSSM state machine is in RX_DETECT.ACTIVE
    if(shared_status_host != null && shared_status_host.ltssm_state == svt_usb_types::RX_DETECT && shared_status_host.ltssm_substate == svt_usb_types::RX_DETECT_ACTIVE) begin
      fork: host_rx_detect
      begin
        `uvm_info("test_top","Wait for Rx Detect to detect logical '0' on TxP and TxM",UVM_LOW);
        // During RX_DETECT.ACTIVE the VIP drives a '1' on the inteface TX lines.
        // This is signal strength demoted to highz1 on sstxp_host_wire and sstxm_host_wire
        // If DUT Rx termination is enabled the resolved sstxp_host_wire and sstxm_host_wire will have '0'
        wait(sstxp_host_wire === 0 && sstxm_host_wire === 0);
        `uvm_info("test_top","Detected RX Termination for Link Partner",UVM_LOW);
        // Enable DUT termination if sstxp_host_wire and sstxm_host_wire is '0'.
        // Disable fork
        usb_host_ss_serial_if.usb_ss_serial_if.dut_ss_termination = 1;
        disable host_rx_detect;
      end
      begin
        // Wait for the LTSSM state to move from RX_DETECT.ACTIVE
        wait(!(shared_status_host.ltssm_state == svt_usb_types::RX_DETECT && shared_status_host.ltssm_substate == svt_usb_types::RX_DETECT_ACTIVE));
        // If flow reaches this stage it means that the sstxp_host_wire and sstxm_host_wire were not seen as 0 in RX_DETECT.ACTIVE
        `uvm_info("test_top",$sformatf("Detected Change in LTSSM state from Rx DETECT ACTIVE: State %s Substate %s. Disable RX Termination bit for VIP",shared_status_host.ltssm_state.name,
        // Disable DUT termination and disable fork
        usb_host_ss_serial_if.usb_ss_serial_if.dut_ss_termination = 0;
        disable host_rx_detect;
      end
      join
    end // if(shared_status_host != null && shared_status_host.ltssm_state == svt_usb_types::RX_DETECT && shared_status_host.ltssm_substate == svt_usb_types::RX_DETECT_ACTIVE) begin
  end // forever
end // initial
```