Verification Continuum™

VC Verification IP
USB
UVM Getting Started Guide

Version V-2023.09, September 2023

SYNOPSYS®

Copyright Notice and Proprietary Information

© 2023 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

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

Disclaimer

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

Trademarks

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

Free and Open-Source Software Licensing Notices

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

Third-Party Links

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

www.synopsys.com

Contents

Preface

About This Document

This Getting Started Guide presents information about integrating the VC VIP for USB (referred to as VIP) into testbenches that are compliant with the SystemVerilog Universal Verification Methodology (UVM). You are assumed to be familiar with the USB protocol and UVM.

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.
  - 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.
- 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.

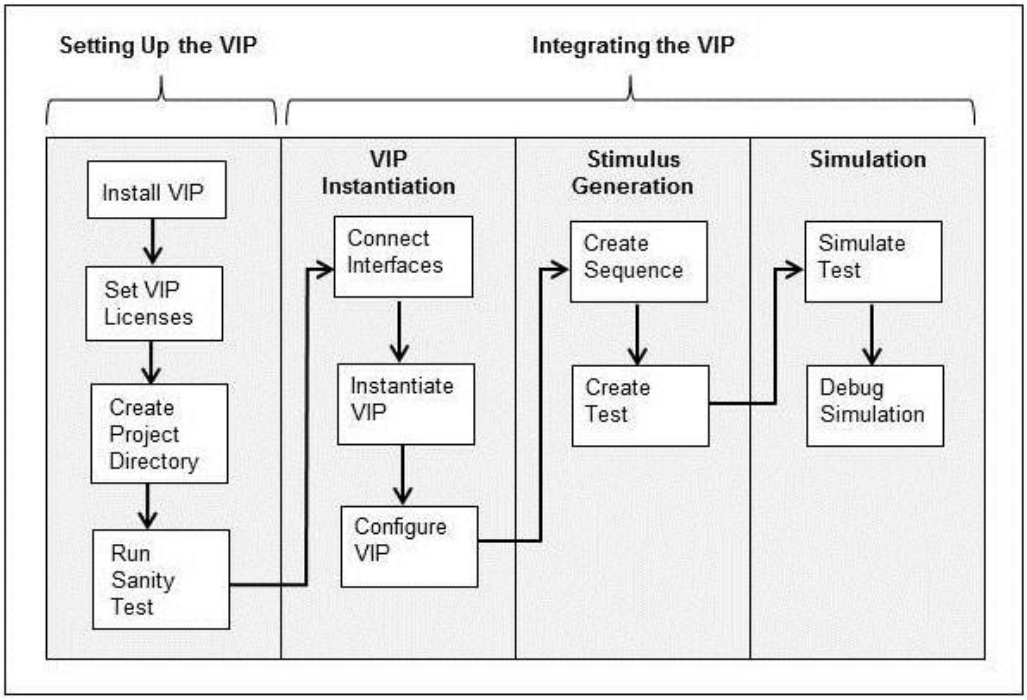
1 Overview of the Getting Started Guide

This Getting Started Guide presents information about integrating the VC VIP for USB (referred to as VIP) into testbenches that are compliant with the SystemVerilog Universal Verification Methodology (UVM). Figure 1-1 is the VIP integration and test work flow presented in this document. The steps for setting up the VIP are documented in the *VC Verification IP Installation and Setup Guide*. This guide is available on the SolvNetPlus Download Centre (click here -> VC VIP Library -> V-2023.09 -> Installation Guide) and in the VIP installation at the following location:

$DESIGNWARE_HOME/vip/svt/common/latest/doc/uvm_install.pdf

The VIP setup should be completed before executing the steps in this document.

Figure 1-1 VIP Integration and Test Work Flow



You are assumed to be familiar with the USB protocol and UVM. For more information on the VIP, see the *VC Verification IP USB UVM User Guide* on SolvNetPlus (click here) or in the VIP installation at the following location:

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

2 Integrating the VIP into a User Testbench

The VC VIP for USB provides a suite of advanced SystemVerilog verification components and data objects that are compliant to UVM. Integrating these components and objects into any UVM compliant testbench is straightforward. For a complete list of VIP components and data objects, see the main page of the *VC VIP USB Class Reference* (only in HTML format) at the following location:

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

2.1 VIP Testbench Integration Flow

The USB agent (`svt_usb_agent`) is the top-level component provided by the VIP for modeling USB hosts, devices and hubs. This generic agent encapsulates the following components:

- Sequencer
- Monitor
- Driver

A USB environment is a user-defined UVM environment that encapsulates all of the VIP components, and implicitly constructs the required number of USB host and USB device agents as specified by its system configuration object. You can instantiate and construct the USB environment in the top-level environment of your UVM testbench.

Figure 2-1 Top-level Architecture of a USB VIP Testbench

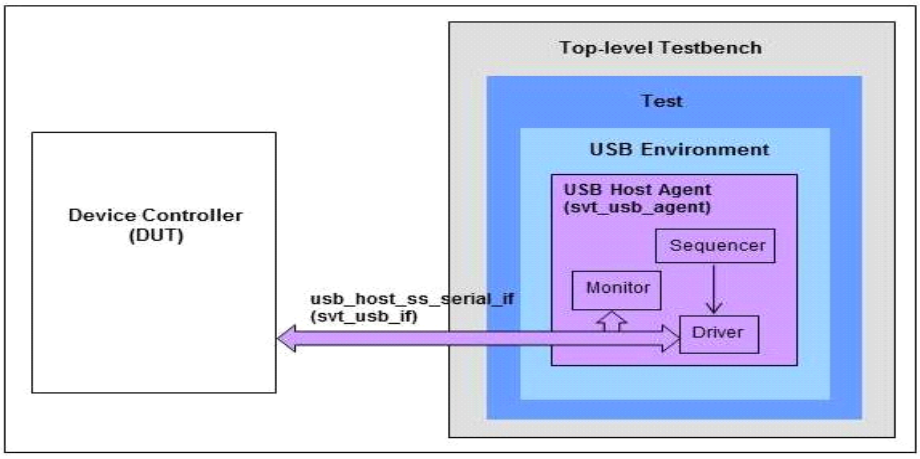


Figure 2-1 is a top-level architecture of a simple VC VIP for USB testbench. The testbench includes an instance of a USB host agent connecting through the USB SuperSpeed (SS) serial interface to an instance of a device controller that represents the DUT. The steps for integrating the VIP into a UVM testbench are described in the following sections:

- Connecting the VIP to the DUT
- Instantiating and Configuring the VIP
- Creating a Test Sequence
- Creating a Test

These steps are documented for a VIP configured as a USB host verifying a device controller (DUT) connected through a SuperSpeed serial interface. Similar steps can be followed for other serial interfaces. The code snippets presented in this chapter are generic and can be applied to any UVM compliant testbench. For more information on the code usage, refer to the following example:

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

2.1.1 Connecting the VIP to the DUT

The following are the steps to establish a connection between the VIP to the DUT in your top-level testbench:

- Include the standard UVM and VIP files and packages.

```
`include "svt_usb_defines.svi"
`include "svt_usb.uvm.pkg"    //VIP package
`include "svt_usb_if.uvm.svi" //top-level USB interface

import uvm_pkg::*;
`include "uvm_macros.svh"
import svt_uvm_pkg::*;

import svt_usb_uvm_pkg::*;
```

Note You must compile all VIP files with the timescale 1ps/1ps. You can use the timescale option at compile-time or add the `timescale 1ps/1ps statement before including the VIP files.

- Instantiate the top-level USB interface (`svt_usb_if`) and connect the serial signals to the DUT.

```
svt_usb_if usb_host_ss_serial_if();

.ssrxp_device (usb_dev_if.usb_host_ss_serial_if.ssrxp),
.ssrxm_device (usb_dev_if.usb_host_ss_serial_if.ssrxm),
.sstxp_device (usb_dev_if.usb_host_ss_serial_if.sstxp),
.sstxm_device (usb_dev_if.usb_host_ss_serial_if.sstxm),
.ssvbus_device (usb_dev_if.usb_host_ss_serial_if.vbus),

//Bi-directionals Connected by Transmission Gates
inout ssvbus_device;
```

```
tran usb_ss_vbus_xmit(usb_host_if.usb_host_ss_serial_if.vbus,
                      ssvbus_device);
//OR
dut dut_inst(tx_p,tx_m,rx_p,rx_m);

assign usb_host_ss_serial_if.usb_ss_serial_if.ssrxp = tx_p;
assign usb_host_ss_serial_if.usb_ss_serial_if.ssrxm = tx_m;
assign rx_p = usb_host_ss_serial_if.usb_ss_serial_if.sstxp;
assign rx_m = usb_host_ss_serial_if.usb_ss_serial_if.sstxm;
assign usb_host_ss_serial_if.usb_ss_serial_if.vbus = 1'b1;
```

- Connect the top-level USB interface to a system clock.

```
//USB SS 5.0GT/s clock period is 200 ps and the unit of the
//testbench timescale is 1ps.
parameter   usb_ss_simulation_cycle = 200;
bit         ss_serial_clock_for_VIP;

initial begin
    ss_serial_clock_for_VIP = 0;
    #(usb_ss_simulation_cycle);     //No clock edge at T=0
    forever begin
      ss_serial_clock_for_VIP = ~ss_serial_clock_for_VIP;
      #(usb_ss_simulation_cycle/8);
      ss_serial_clock_for_VIP = ~ss_serial_clock_for_VIP;
      #(usb_ss_simulation_cycle/8);
      ss_serial_clock_for_VIP = ~ss_serial_clock_for_VIP;
      #(usb_ss_simulation_cycle/8);
      ss_serial_clock_for_VIP = ~ss_serial_clock_for_VIP;
      #((usb_ss_simulation_cycle/2) -
                          (3*(usb_ss_simulation_cycle/8)));
      ss_serial_clock_for_VIP = ~ss_serial_clock_for_VIP;
      #(usb_ss_simulation_cycle/8);
      ss_serial_clock_for_VIP = ~ss_serial_clock_for_VIP;
      #(usb_ss_simulation_cycle/8);
      ss_serial_clock_for_VIP = ~ss_serial_clock_for_VIP;
      #(usb_ss_simulation_cycle/8);
      ss_serial_clock_for_VIP = ~ss_serial_clock_for_VIP;
      #(((usb_ss_simulation_cycle)-(usb_ss_simulation_cycle/2))-
                          (3*(usb_ss_simulation_cycle/8)));
    end
  end

//Assign generated clock to the SS clock input of the VIP SS serial
//interface (ssclk)

assign usb_host_ss_serial_if.usb_ss_serial_if.ssclk = ss_serial_clock_for_VIP;
```

Note For VIP operations with serial interfaces, the VIP requires a 4x USB speed since clock recovery is enabled by default. This requirement applies to all VIP operations regardless of the VIP's configuration as a USB host, device or hub.

For SuperSpeed (SS) operations, the input clock of the VIP (`usb_ss_serial_if.ssclk`) requires to be 20 GT/s (i.e. 4x 5 GT/s).

For USB 2.0 (HS or FS or LS) operations, the input clock of the VIP (`usb_20_serial_if.clk`) requires to be 1.92 GT/s (i.e. 4x 480 MT/s).

- Connect the top-level USB interface to the virtual interface of the USB environment.

```
intial begin
  uvm_config_db#(virtual svt_usb_if)::set(uvm_root::get(),
  "uvm_test_top.env", "host_usb_if", usb_host_ss_serial_if);
end
```

The `uvm_config_db` command connects the top-level USB interface to the virtual interface of the USB environment. The `uvm_test_top` represents the top-level module in the UVM environment. The `env` is an instance of your testbench environment.

2.1.2 Instantiating and Configuring the VIP

The following are steps to instantiate and configure the USB agent (`svt_usb_agent`) in your testbench environment.

- Instantiate the USB environment in the build phase of your testbench environment.

```
usb_basic_serial_env env;

env = usb_basic_serial_env::type_id::create("env", this);
```

- Instantiate the USB agent (`svt_usb_agent`) in the build phase of your testbench environment and bind the environment virtual interface to the agent virtual interface

```
svt_usb_agent host_agent;
svt_usb_if host_usb_if;

host_agent = svt_usb_agent::type_id::create("host_agent", this);
void'(uvm_config_db#(virtual svt_usb_if)::get(null,get_full_name(),
        "host_usb_if", host_usb_if));

if (this.dev_usb_if != null)
begin
  if (cfg.host_cfg.usb_ss_signal_interface !=
                          svt_usb_configuration::NO_SS_IF)
begin
   uvm_config_db#(USB_IF)::set(this, "host_agent", "usb_ss_if",
                          this.host_usb_if);
end
end
```

- Create a basic configuration class by extending the `uvm_object` class. This configuration class specifies the USB protocol layer configuration for the USB host and device.

  For example,

```
class usb_shared_cfg extends uvm_object;
  svt_usb_agent_configuration host_cfg;
      ...
  function new(string name = "usb_shared_cfg_inst");
    begin
super.new(name);
```

```
host_cfg = new();
    host_cfg.component_type = svt_usb_types::HOST;
    host_cfg.top_layer = svt_usb_agent_configuration::PROTOCOL;
    host_cfg.capability = svt_usb_configuration::PLAIN;
    host_cfg.speed = svt_usb_types::SS;

    host_cfg.usb_ss_signal_interface =
                        svt_usb_configuration::USB_SS_SERIAL_IF;

    host_cfg.usb_capability = svt_usb_configuration::USB_SS_ONLY;
    host_cfg.usb_ss_initial_ltssm_state = svt_usb_types::U0;
    host_cfg.usb_20_signal_interface = svt_usb_configuration::NO_20_IF;
    host_cfg.local_device_cfg = null;

    host_cfg.local_host_cfg_size = 1;
    host_cfg.local_host_cfg[0] = new();

    host_cfg.remote_host_cfg_size = 0;
    host_cfg.remote_host_cfg = null;
    host_cfg.remote_device_cfg_size = 1;
    host_cfg.remote_device_cfg[0] = new();
    host_cfg.remote_device_cfg.capability = svt_usb_configuration::PLAIN;
    host_cfg.remote_device_cfg[0].speed = svt_usb_types::SS;
    host_cfg.remote_device_cfg[0].usb_ss_signal_interface =
                         svt_usb_configuration::USB_SS_SERIAL_IF;

    host_cfg.remote_device_cfg[0].device_address = 0;
    host_cfg.remote_device_cfg[0].connected_bus_speed = svt_usb_types::SS;
    host_cfg.local_device_cfg[0].connected_hub_device_address = 0;

    host_cfg.remote_device_cfg[0].functionality_support =
                                                svt_usb_types::SS;

    host_cfg.remote_device_cfg[0].num_endpoints = 1;

    host_cfg.remote_device_cfg[0].usb_capability =
                         svt_usb_configuration::USB_SS_ONLY;

    host_cfg.remote_device_cfg[0].usb_ss_initial_ltssm_state =
                                                svt_usb_types::U0;
  host_cfg.remote_device_cfg[0].usb_20_signal_interface =
                               svt_usb_configuration::NO_20_IF;

    host_cfg.remote_device_cfg[0].endpoint_cfg[0] = new();
    host_cfg.remote_device_cfg[0].endpoint_cfg[0].ep_number = 0;

    host_cfg.local_device_cfg[0].endpoint_cfg[0].direction =
                                                svt_usb_types::IN;

    host_cfg.remote_device_cfg[0].endpoint_cfg[0].ep_type =
                                           svt_usb_types::CONTROL;
  host_cfg.remote_device_cfg[0].endpoint_cfg[0].max_burst_size = 0;
  host_cfg.remote_device_cfg[0].endpoint_cfg[0].max_packet_size =
                               `SVT_USB_SS_CONTROL_MAX_PACKET_SIZE;
```

```
        host_cfg.remote_device_cfg[0].endpoint_cfg[0].supports_ustreams
        =1'b0;
        end
        endfunction
            ...
endclass
```

Note The USB host configuration must include the configuration of a USB device that the host is communicated with. Hence, the remote_device_cfg is constructed.

For more information on the configuration class, see the `svt_usb_configuration` and `svt_usb_endpoint_configuration` Class References at the following locations:

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

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

- Construct the customized USB configuration and pass the configuration to the USB environment (instance of `usb_basic_serial_env`) in the build phase of your testbench environment.

```
usb_shared_cfg cfg;

cfg = usb_shared_cfg::type_id::create("cfg", this);
cfg.setup_usb_ss_defaults();
uvm_config_db#(usb_shared_cfg)::set(this, "env", "cfg", this.cfg);
```

The `usb_shared_cfg` is the customized USB configuration as defined in the previous step. The `cfg` is an instance of this configuration.

- Assign configuration to the agent object in the build phase of your testbench environment.

```
usb_shared_cfg cfg;

void'(uvm_config_db#(usb_shared_cfg)::get(get_parent(), get_name(),"cfg", cfg));

if (cfg == null) begin
  `uvm_info("build_phase", "External cfg not provided, creating
            default cfg.", UVM_LOW)
  cfg = usb_shared_cfg::type_id::create("cfg");
end
else
begin
  `uvm_info("build_phase", "Using externally provided cfg.",
            UVM_LOW)
end
if (this.cfg.is_valid(0))
begin
`uvm_info("build_phase", $sformatf("cfg passed is_valid() check.
            Contents:\n%0s", this.cfg.sprint()), UVM_LOW)
end
else
begin
```

```
        `uvm_fatal("build_phase", "cfg failed is_valid() check. Unable to continue.")
        end
          uvm_config_db#(svt_usb_agent_configuration)::set(this, "host_agent", "cfg",
        cfg.host_cfg);
```

2.1.3 Creating a Test Sequence

The VIP provides a base sequence class for the USB host agent (`svt_usb_host_base_sequence`) and the USB device agent (`svt_usb_device_base_sequence`). You can extend these base sequences to create test sequences for the USB host and device agents.

For more information on the USB host and device base sequences, and the VIP sequence collection, refer to the Sequence Page of the VC VIP USB Class Reference at the following location:

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

In addition, a list of random and directed sequences are available in the VIP examples. For more information on the example sequences, refer to the example directories at the following location:

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

Note
You must set a device response sequence for active devices in the run phase.

2.1.4 Creating a Test

You can create a VIP test by extending the `uvm_test` class. In the build phase of the extended class, you construct the testbench environment and set the respective USB host and device sequences.

```
        class basic_ss_serial extends usb_base_test;

          // build_phase

        uvm_config_db#(uvm_object_wrapper)::set(this,"env.host_agent.xfer_sequencer.main_p
        hase","default_sequence",usb_bulk_out_bulk_in_random_sequence::type_id::get());
```

2.2 Compiling and Simulating a Test with the VIP

The steps for compiling and simulating a test with the VIP are described in the following sections:

- "Directory Paths for VIP Compilation"
- "VIP Compile-time Options"
- "VIP Runtime Option"

2.2.1 Directory Paths for VIP Compilation

You need to specify the following directory paths in the compilation commands for the compiler to load the VIP files.

```
        +incdir+project_directory_path/include/sverilog
        +incdir+project_directory_path/src/sverilog/simulator
```

Where, *project_directory_path* is your project directory and *simulator* is vcs, ncv or mti.

For example,

```
+incdir+/home/project1/testbench/vip/include/sverilog
+incdir+/home/project1/testbench/vip/src/sverilog/vcs
```

2.2.2 VIP Compile-time Options

The following are the required compile-time options for compiling a testbench with the VC VIP for USB:

```
+define+SVT_UVM_TECHNOLOGY
+define+UVM_PACKER_MAX_BYTES=24000
+define+UVM_DISABLE_AUTO_ITEM_RECORDING
+define+SYNOPSYS_SV
```

Note UVM_PACKER_MAX_BYTES define needs to be set to maximum value as required by each VIP title in your testbench. For example, if VIP title 1 needs UVM_PACKER_MAX_BYTES to be set to 8192, and VIP title 2 needs UVM_PACKER_MAX_BYTES to be set to 500000, you need to set UVM_PACKER_MAX_BYTES to 500000.

Table 2-1 Macro

Macro	Description
SVT_UVM_TECHNOLOGY	Specifies SystemVerilog based VIPs that are compliant with UVM
UVM_PACKER_MAX_BYTES	Sets to 24000 or greater
UVM_DISABLE_AUTO_ITEM_RECORDING	Disables the UVM automatic transaction begin and end event triggering and recording
SYNOPSYS_SV	Specifies SystemVerilog based VIPs that are compliant with UVM

2.2.3 VIP Runtime Option

No VIP specific runtime option is required to run simulations with the VIP. Only relevant UVM runtime options are required.

For example,

```
+UVM_TESTNAME=basic_ss_serial
```

A Summary of Commands, Documents, and Examples

A.1 Commands in This Document

Display VIP models and examples under the VIP installation directory specified by $DESIGNWARE_HOME:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -info home
```

Add VIP models to the project directory:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup  -path project_directory -add VIP_model
-svlog
```

Add VIP examples to the directory where the command is executed:

```
% $DESIGNWARE_HOME/bin/dw_vip_setup -e VIP_example -svlog
```

A.2 Primary Documentation for VC VIP USB

VC Verification IP UVM Installation and Setup Guide:

$DESIGNWARE_HOME/vip/svt/common/latest/doc/uvm_install.pdf

VC VIP USB UVM User Guide:

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

VC VIP USB Getting Started Guide:

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

VC VIP USB QuickStart:

$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 USB Class Reference:

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

VC VIP USB Verification Plans:

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

A.3 Example Home Directory

Directory that contains a list of VIP example directories:

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

View simulation options for each example:

```
gmake help
```