Verification Continuum™

VC Verification IP DisplayPort UVM Getting Started Guide

Version Q-2020.06, June 2020

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://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 DP (referred to as VIP) into testbenches that are compliant with the SystemVerilog Universal Verification Methodology (UVM). You are assumed to be familiar with the DP SVT 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:

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

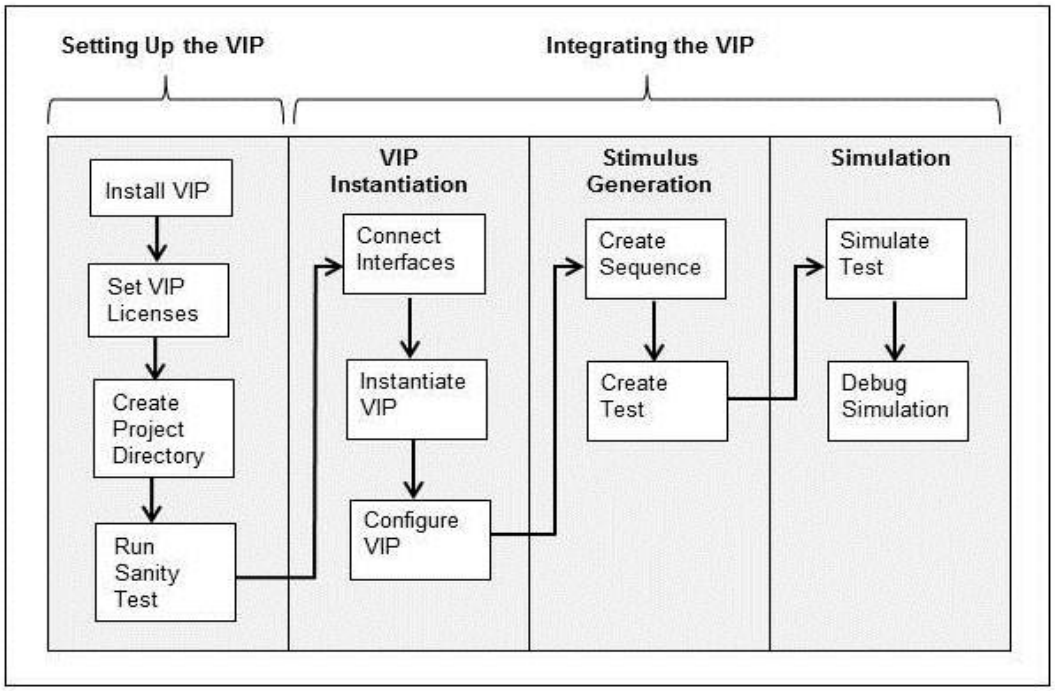
1 Overview of the Getting Started Guide

This Getting Started Guide presents information about integrating the VC VIP DP (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 SolvNet Download Center (click here -> VC VIP Library -> Q-2020.03 -> 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 DP SVT protocol and UVM. For more information on the VIP, refer to the *VC Verification IP DP SVT UVM User Guide* on SolvNet (click here) or in the VIP installation at the following location:

$DESIGNWARE_HOME/vip/svt/dp_svt/latest/doc/dp_svt_uvm_user_guide.pdf

2 Integrating the VIP into a User Testbench

The VC VIP for DP 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, refer to the main page of the VC VIP for DP *Class Reference* (only in HTML format) at the following location:

$DESIGNWARE_HOME/vip/svt/dp_svt/latest/doc/dp_svt_uvm_class_reference/html/index.html

2.1 DP VIP Testbench Integration Flow

The DP environment (svt_dp_env) is the top-level component provided by the VIP for modeling DP source and sink devices. This generic environment encapsulates the following components:

- Main Link Agent that contains the following components:
  - Sequencer
  - Monitor
  - Driver
- AUX Agent that contains the following components:
  - Sequencer
  - Monitor
  - Driver

You can instantiate and construct the DP environment in the top-level environment of your UVM testbench.

Figure 2-1 Top-level Architecture of a DP VIP Testbench (DUT as MASTER, VIP as SLAVE)

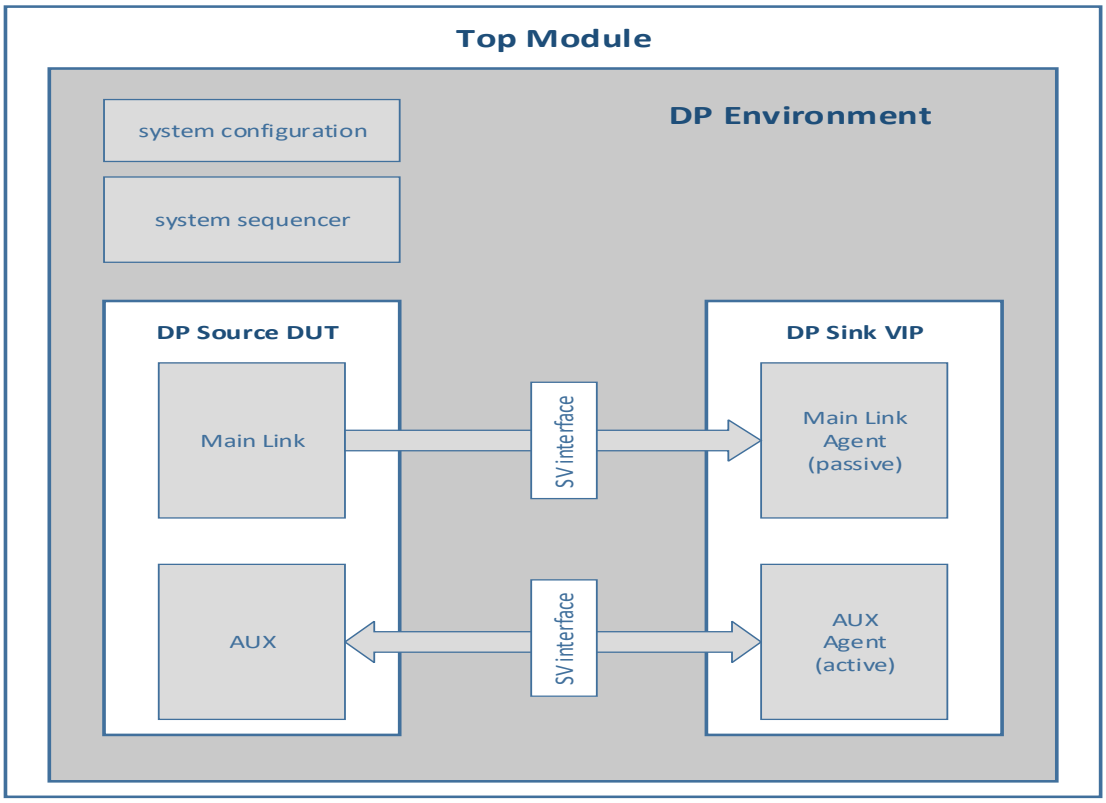


Figure 2-1 shows top-level architecture of a simple VC VIP for DP testbench. The steps for integrating the VIP into a UVM testbench are described in the following sections:

- "Connecting the VIP to the DUT"
- "Creating a Test Sequence"
- "Creating a Test"

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/dp_svt/latest/examples/sverilog/tb_dp_svt_uvm_basic_sy
```

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 following VIP directories in your env.

```
+<design_dir>/include
+<design_dir>/lib
+<design_dir>/src
```

`<design_dir>` is a base directory where example is installed.

- Include the standard UVM and VIP files and packages.

```
`include "uvm_pkg.sv"
`include "svt_dp.uvm.pkg"
`include "svt_hdcp.uvm.pkg"
`include "svt_dp_if.svi" //top-level DP interface
`include "uvm_macros.svh"

import uvm_pkg::*;
import svt_uvm_pkg::*;
import svt_hdcp_uvm_pkg::*
import svt_dp_uvm_pkg::*
```

2.1.1.1 VIP as Source (Tx) and Sink (Rx) as DUT

2.1.1.1.1 Connecting DUT Signals to the VIP Interface

If Source is VIP, then perform the following steps:

In testbench top, instantiate the DP interface.

```
svt_dp_if    src_dp_if();

Connect top level dp interface as mentioned below:
assign dut.lane0 = src_dp_if.main_link_if.lane_0;
assign dut.lane1 = src_dp_if.main_link_if.lane_1;
assign dut.lane2 = src_dp_if.main_link_if.lane_2;
assign dut.lane3 = src_dp_if.main_link_if.lane_3;
assign dut.pix_clk = src_dp_if.main_link_if.pix_clk;

assign src_dp_if.aux_if.hpd = dut.hpd;

Connect top level dp_if to virtual interface of DP Sink environment
uvm_config_db#(virtual svt_dp_if)::set(uvm_root::get(), "uvm_test_top.env",
"source_vif", src_dp_if);
```

Create UVM Environment

a. Extend `uvm_env` to create DP user environment.

```
class dp_test_env extends uvm_env;
```

b. Create configuration data object reference.

```
dp_shared_cfg shared_cfg;
```

c. Instantiate Source DP environment.

```
svt_dp_env source_device;
```

d. In build phase of environment, create both configuration and `source_env` and set source configuration through `uvm_config_db` as shown in the following snippet:

```
shared_cfg = svt_dp_env_configuration::type_id::create("shared_cfg");
uvm_config_db#(svt_dp_env_configuration)::set(this, "source_device", "cfg",
this.shared_cfg.source_cfg);
source_device = svt_dp_env::type_id::create("source_device",this);
```

Create UVM Test Case and Configure VIP

a. Create a test case by extending the class `dp_base_test` (if you extend from `uvm_test`, then `dp_base_test` should be replicated by using all functions implemented).

```
class user_testcase extends dp_base_test
```

b. Instantiate DP test environment.

```
dp_test_env env;
```

c. Declare configuration class.

```
dp_shared_cfg shared_cfg;
```

d. In build phase, construct environment and shared configuration class and pass the configuration to the environment.

```
env = dp_test_env::type_id::create("env", this);
            shared_cfg = dp_shared_cfg::type_id::create("shared_cfg");
            uvm_config_object_db#(dp_shared_cfg)::set(this,"env",
"shared_cfg", shared_cfg);
```

Base test includes configurations in which the variables will remain almost same across all the tests. Some of the configurations are shown in the following example:

```
shared_cfg.source_cfg.device_type = svt_dp_types::SOURCE;
shared_cfg.source_cfg.dp_protocol_version = svt_dp_types::DP_VERSION_1_3;
```

If you want to provide your own constraints, then it can be done in the `user_testcase`:

```
rand_status = shared_cfg.randomize() with{
      shared_cfg.source_cfg.framing_mode == shared_cfg.sink_cfg.framing_mode;
      shared_cfg.source_cfg.stream_mode      == svt_dp_types::SST_MODE;
      shared_cfg.source_cfg.tu_size                  == 64;
      shared_cfg.source_cfg.dp_stream_cfg[0].vic       == 1;
```

2.1.1.2 DUT as Source (Tx) and VIP as Sink (Rx)

2.1.1.2.1 Connecting DUT Signals to the VIP Interface

If Sink is VIP, then perform the following steps:

In testbench top, instantiate the DP interface.

```
svt_dp_if   sink_dp_if();

Connect top level dp interface as mentioned below:
assign sink_dp_if. main_link_if.lane_0 =   dut.lane0;
assign sink_dp_if.main_link_if.lane_1  =  dut.lane1;
assign sink_dp_if.main_link_if.lane_2  =  dut.lane2;
assign sink_dp_if.main_link_if.lane_3 = dut.lane3;
assign sink_dp_if.main_link_if.pix_clk = dut.pix_clk;

assign dut.hpd = sink_dp_if.aux_if.hpd;
```

Connect top level `dp_if` to virtual interface of DP Sink environment

```
uvm_config_db#(virtual svt_dp_if)::set(uvm_root::get(),
"uvm_test_top.env", "sink_vif", sink_dp_if);
```

Create UVM Environment

a. Extend `uvm_env` to create DP user environment.

```
class dp_test_env extends uvm_env;
```

b. Create configuration data object reference.

```
dp_shared_cfg shared_cfg;
```

c. Instantiate Sink DP environment.

```
svt_dp_env  sink_device;
```

d. In build phase of environment, create both configuration and sink environment and set sink the configuration through `uvm_config_db` as shown in the following snippet:

```
shared_cfg = svt_dp_env_configuration::type_id::create("shared_cfg");
uvm_config_db#(svt_dp_env_configuration)::set(this, "sink_device", "cfg",
this.shared_cfg.sink_cfg);
sink_device = svt_dp_env::type_id::create("sink_device",this);
```

Create UVM Test Case and Configure VIP

a. Create a test case by extending the class `dp_base_test` (if you extend from `uvm_test`, then `dp_base_test` should be replicated by using all functions implemented).

b. Class `user_testcase` extends `dp_base_test`.

c. Instantiate DP test environment.

```
dp_test_env env;
```

d. Declare configuration class.

```
dp_shared_cfg shared_cfg;
```

e. In build phase, construct environment and shared configuration class and pass the configuration to the environment.

```
env = dp_test_env::type_id::create("env", this);
                  shared_cfg = dp_shared_cfg::type_id::create("shared_cfg");
                  uvm_config_object_db#(dp_shared_cfg)::set(this,"env",
"shared_cfg", shared_cfg);
```

Base test includes configurations in which the variables will remain almost same across all the tests. Some of the configurations are shown in the following example:

```
shared_cfg.sink_cfg.device_type = svt_dp_types::SINK;
shared_cfg.sink_cfg.dpcd_version = svt_dp_types::DPCD_REV_1_4;
```

If you want to provide your own constraints, then it can be done in the user_test case as follows:

```
rand_status = shared_cfg.randomize() with{
      shared_cfg.sink_cfg.framing_mode == shared_cfg.sink_cfg.framing_mode;
      shared_cfg.sink_cfg.stream_mode      == svt_dp_types::SST_MODE;
      shared_cfg.sink_cfg.tu_size                  == 64;
      shared_cfg.sink_cfg.dp_stream_cfg[0].vic       == 1;
```

2.1.2 Creating a Test Sequence

The VIP provides a base sequence class for the DP AUX agent (svt_dp_aux_transaction_base_sequence) that executes on AUX agent sequencer, DP Main Link agent (svt_dp_frame_line_transaction_sequence) that executes on Main Link agent sequencer and DP virtual base sequence that executes on virtual sequencer. You can extend this virtual base sequence to create test sequences for the DP source environment.

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

$DESIGNWARE_HOME/vip/svt/dp_svt/latest/doc/dp_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/dp_svt/latest/examples/sverilog
```

2.1.3 Creating a Test

You can create a VIP test by extending the dp_base_test class.

Class dp_base_test creates env, calls randomize_cfg() and sets randomized configuration trough configuration db.

In the build phase of the extended class, you construct the testbench environment and set the respective DP sequences.

```
class random_test extends dp_base_test;

  uvm_config_db #(uvm_object_wrapper)::set(this,
       "env.source_device.sequencer.main_phase",
       "default_sequence",
       cust_svt_dp_source_random_sequence::type_id::get());
```

2.1.4 Running Example Testbench

The test cases are available under *tests* directory:

<design_dir>/examples/sverilog/dp_svt/tb_dp_svt_uvm_basic_sys/tests/

Format of test case name: *ts.<test_case_name>.sv*

Sample example test cases: *ts.aux_native_rd_wr_test.sv, ts.video_basic_test.sv*

Perform the following steps to run the example:

Step 1: Set `DESIGNWARE_HOME` variable to the directory where VIP is installed.

```
unix%  setenv   DESIGNWARE_HOME    <directory_path>
```

Step 2: List all installed components in *DESIGNWARE_HOME*.

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

For example,

```
# LIBRARIES:
# common L-2016.06
# dp_svt L-2016.06
# hdcp_svt L-2016.06
# svt L-2016.06

# MODELS:
# dp_aux_agent_svt L-2016.06
# dp_aux_group_svt L-2016.06
# dp_aux_svt L-2016.06
# dp_env_svt L-2016.06
# dp_hdcp_svt L-2016.06
# dp_main_link_agent_svt L-2016.06
# dp_main_link_group_svt L-2016.06
# dp_main_link_svt L-2016.06

# EXAMPLES:

# dp_svt/tb_dp_svt_uvm_basic_sys
```

Step 3: Install an example in the directory of your choice or <*design_dir*>.

```
unix%  $DESIGNWARE_HOME/bin/dw_vip_setup -path <design_dir> -
example dp_svt/tb_dp_svt_uvm_basic_sys -svtb
```

Note Installing example ensures all required VIP files are installed.

Step 4: Run the test case.

You can run the test cases (either one at a time or all together) using one of the following two options:

Option 1.

Use the run script that is generated by dw_vip_setup, and available in the current working directory:

Run script:

```
unix%  cd <design_dir>/examples/sverilog/dp_svt/tb_dp_svt_uvm_basic_sys/
unix%  run_dp_svt_uvm_basic_sys <test_case_name>     <simulator_type>
```

Format:

```
run_dp_svt_uvm_basic_sys    <test_case_name>     <simulator_type>
```

Samples:

```
run_dp_svt_uvm_basic_sys    aux_native_rd_wr_test vcsvlog
run_dp_svt_uvm_basic_sys    video_basic_test vcsvlog
run_dp_svt_uvm_basic_sys    all vcsvlog
```

For more command-line options, invoke the following command:

```
run_dp_svt_uvm_basic_sys   -help
```

Option 2

Use the provided make file.

```
unix%  cd <design_dir>/examples/sverilog/dp_svt/tb_dp_svt_uvm_basic_sys/
```

Format:

```
gmake   USE_SIMULATOR=<simulator_type>      <test_case_name>
```

Samples:

```
gmake  USE_SIMULATOR=vcsvlog   aux_native_rd_wr_test
gmake  USE_SIMULATOR=vcsvlog   video_basic_test
gmake  USE_SIMULATOR=vcsvlog   all
```

Add `WAVES=1` at the end of the command to generate waves.

Add `WAVES=fsdb` at the end of the command to generate FSDB file.

Add `WAVES=vdb` at the end of the command to generate vdb file.

The waveform display groups the signals in a meaningful way and sets up the display radices.

Outputs generated after running the examples are as follows:

Simulation transcript: *./output/<test_case_name>.transcript*

Simulation run file: *./<test_case_name>.vcsvlog.svlog* (when vcsvlog simulator is used)

VIP trace files: **_trace* (for more details, see *VC Verification IP DisplayPort UVM User Guide*.)

WAVE files: *./*.vpd, ./*.fsdb*

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 DP:

```
+define+SVT_UVM_TECHNOLOGY
+define+UVM_PACKER_MAX_BYTES=1500000
+define+UVM_DISABLE_AUTO_ITEM_RECORDING
+define+SYNOPSYS_SV
+define+SVT_DP_DATA_WIDTH=8
```

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.

Macro	Description
SVT_UVM_TECHNOLOGY	Specifies SystemVerilog based VIPs that are compliant with UVM
UVM_PACKER_MAX_BYTES	Sets to 1500000 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=random_test
```

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 DP SVT

VC Verification IP UVM Installation and Setup Guide:
$DESIGNWARE_HOME/vip/svt/common/latest/doc/uvm_install.pdf

VC VIP DP SVT UVM User Guide:
$DESIGNWARE_HOME/vip/svt/dp_svt/latest/doc/dp_svt_uvm_user_guide.pdf

VC VIP DP SVT Getting Started Guide:
$DESIGNWARE_HOME/vip/svt/dp_svt/latest/doc/dp_svt_uvm_getting_started.pdf

VC VIP DP QuickStart:

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

tb_dp_svt_uvm_basic_sys/doc/tb_dp_svt_uvm_basic_sys/index_basic.html

VC VIP DP SVT Class Reference:

$DESIGNWARE_HOME/vip/svt/dp_svt/latest/doc/dp_svt_uvm_class_reference/html/index.html

VC VIP DP SVT Verification Plans:

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

A.3 Example Home Directory

Directory that contains a list of VIP example directories:

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

View simulation options for each example:

```
gmake help
```