PARADIGM® WORKS

Stacking UVCs Methodology

Revision 1.2

Table of Contents

1 Stacking UVCs Overview

This document describes a methodology that allows for organizations to create UVM Verification Components or UVCs that have the flexibility to optionally be connected to other UVCs. This document is accompanied with an example to help aid users with understanding the solution.

2 References

- UVM User Guide Universal Verification Methodology (UVM) 1.1 User's Guide

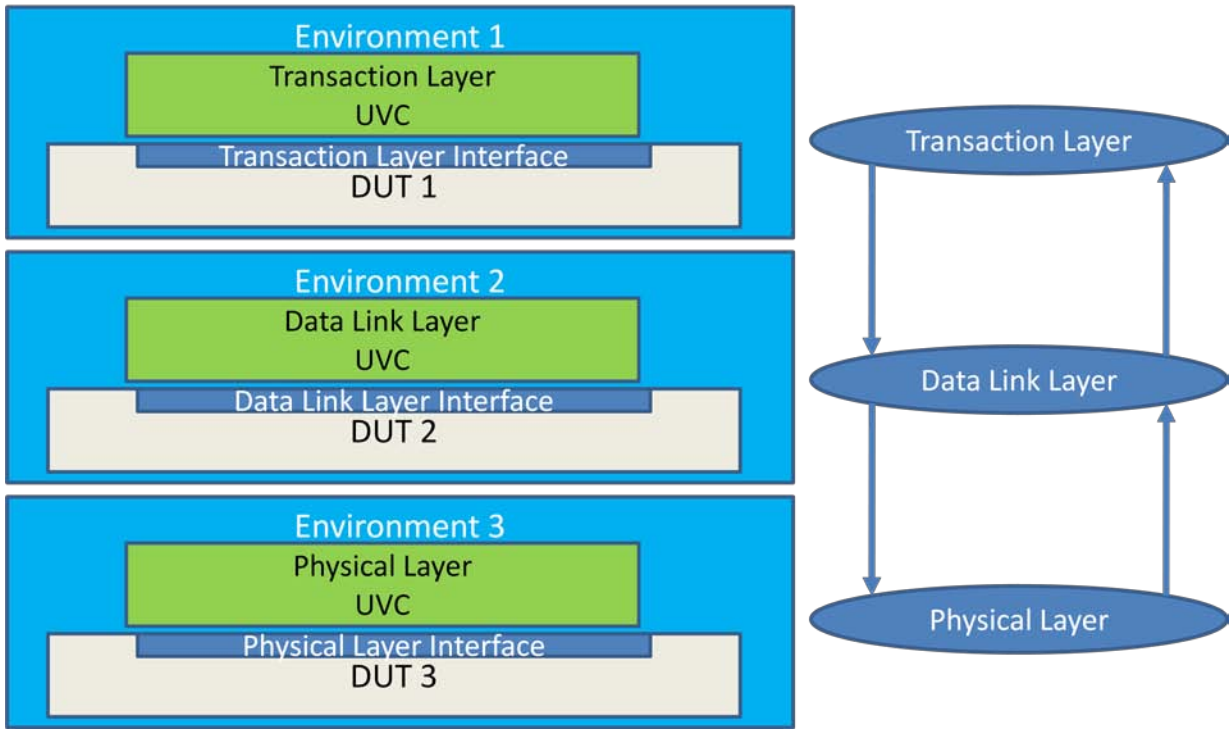
3 Terms, Definitions, and Abbreviations

- UVM - Universal Verification Methodology
- UVC - UVM Verification Component

4 Stacking UVCs Motivation

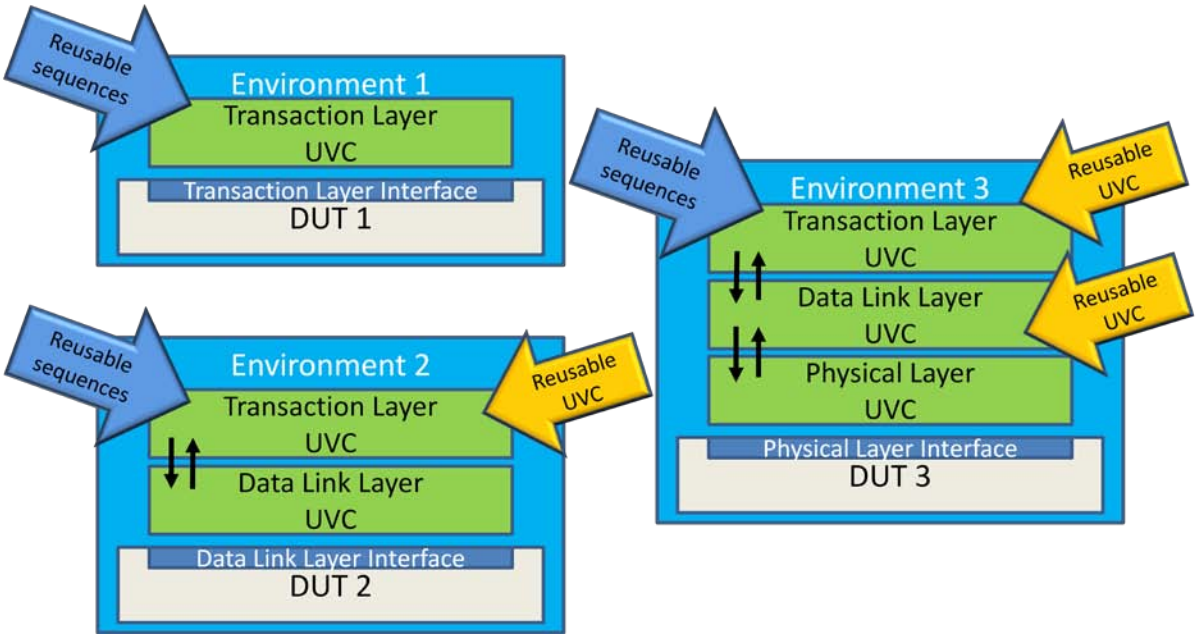
Often verification teams are performing verification tasks against dealing with designs that include multiple protocols that contain multiple layers. An example of a 3 layer protocol is PCIE which contains a Transaction, Data Link, and Physical Layer. Additionally, the teams will often create multiple unit level testbenches to test each RTL layer as well as a system level testbench to test all the layers connected up in RTL. This is just one example of many layered protocols.

A verification team may decide to architect three separate unit testbenches for each protocol layer as shown below. Inside each testbench, the unit level teams may independently develop UVCs for the protocol stack layer – a Transaction UVC, Data Link UVC, and Physical Layer UVC. The teams may develop their entire stimulus at each layer without taking into account data coming in from the higher layers. This results in stimulus that cannot easily be reused amongst the testbenches. Even worse the stimulus will likely not be able to easily be reused in a system level testbench.



4-1 UVCs without Reuse Taken Into Account

An alternative architect approach is shown below. Using this approach each of the upper layer's UVC and their associated stimulus libraries may be reused in subsequent testbenches. For example, the Transaction's UVC's sequences developed in Environment 1 can be reused in Environment 2 and Environment 3. In addition, the Transaction UVC developed in Environment 1 is reused in Environment 2 and Environment 3.



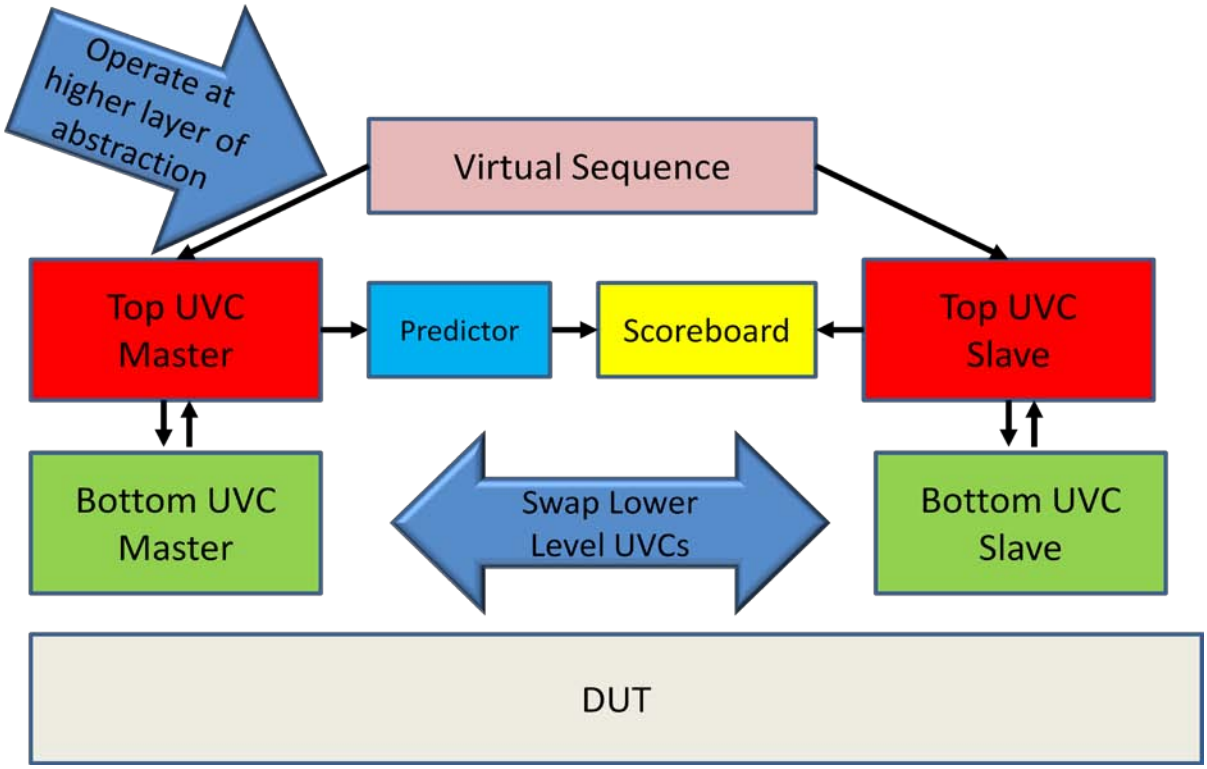
4-2 Vertical Reuse Example

Another example of a common verification scenario where stacking UVCs comes in handy is shown below. In this example, we generate stimulus at a higher level of abstraction in the TOP Master & Slave UVCs rather than Bottom UVCs.

The Top UVC is an abstract UVC. Ethernet is an example of an abstract Top UVC.

The Bottom UVC is an interface UVC which communicates directly to the RTL. Two examples of interface UVCs are SPI and PCIE.

Using this example, we generate stimulus, predict, and scoreboard data at the Ethernet level of abstraction. The Ethernet transactions are converted to either PCIE or SPI. This methodology allows us to easily swap the PCIE and SPI UVCs while reusing the same stimulus, prediction, and scoreboarding. So if a design operates with different low level interface protocols but a high level protocol is used by the design or designs then this methodology is very beneficial.



4-3 Horizontal Reuse Example

5 What is a Stacked UVC

A stacked UVC is a standard UVC that includes the flexibility to optionally be connected to other UVCs. A SV "adapter package" is used to setup an interconnection strategy for UVCs. With this ability we can easily meet the reuse concepts described above.

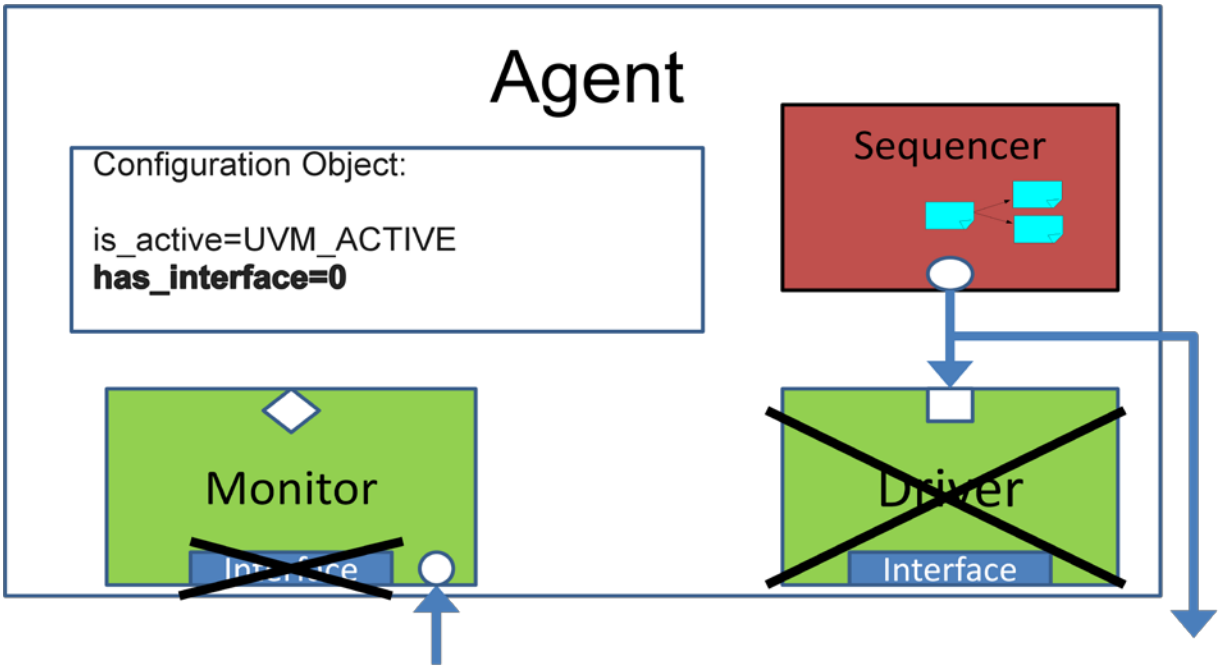
5.1 Stacking UVC Methodology Requirements

This is a list of stacking requirements that we agreed upon

- UVCs do not have handles or references of other UVCs
- UVCs have a configuration field which allow it to exclude its SV interface
- A SV adapter package that is separate from the UVC connects to other UVCs
- The SV adapter package are reusable from testbench to testbench
- The SV adapter package includes configuration

5.2 Stacked UVC

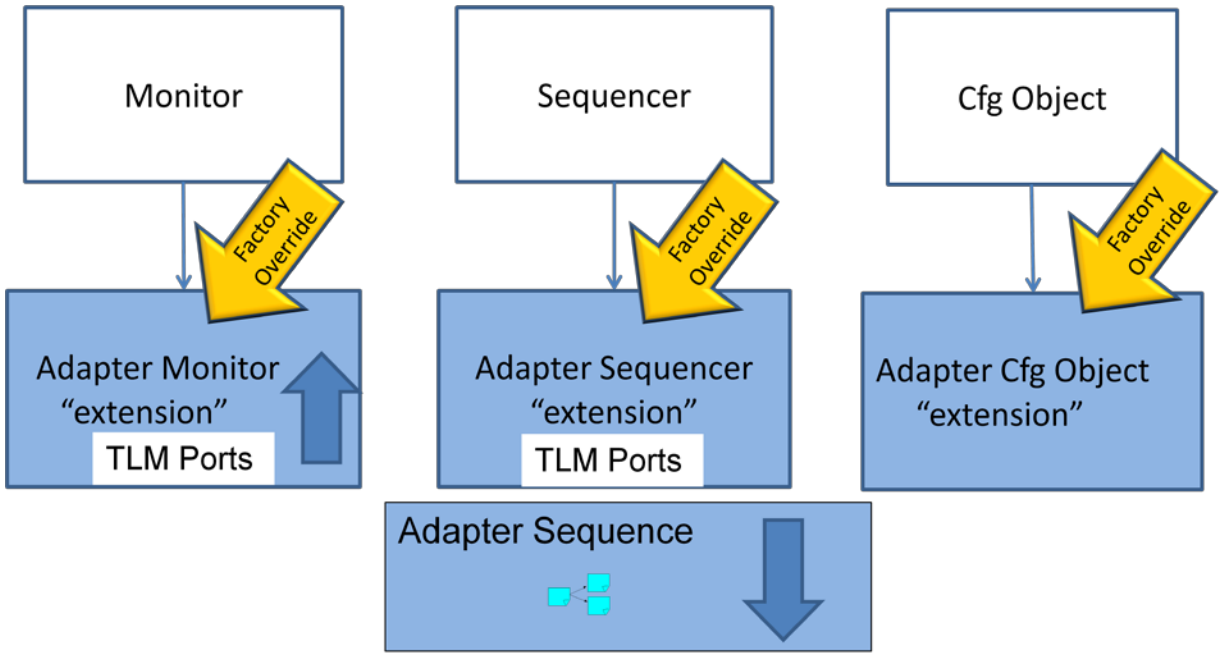
An example of an UVC Agent is shown below. For stacking capabilities, the **has_interface** configuration switch is added. When **has_interface=1** the UVC operates in the same exact manner as that described in the UVM User Guide. When **has_interface=0** the UVC's Agent excludes its Driver and the SV interface is not setup and used by the Agent's Monitor. In this mode, we setup the Agent to communicate to other UVCs.



5-1 UVC Agent with has_interface

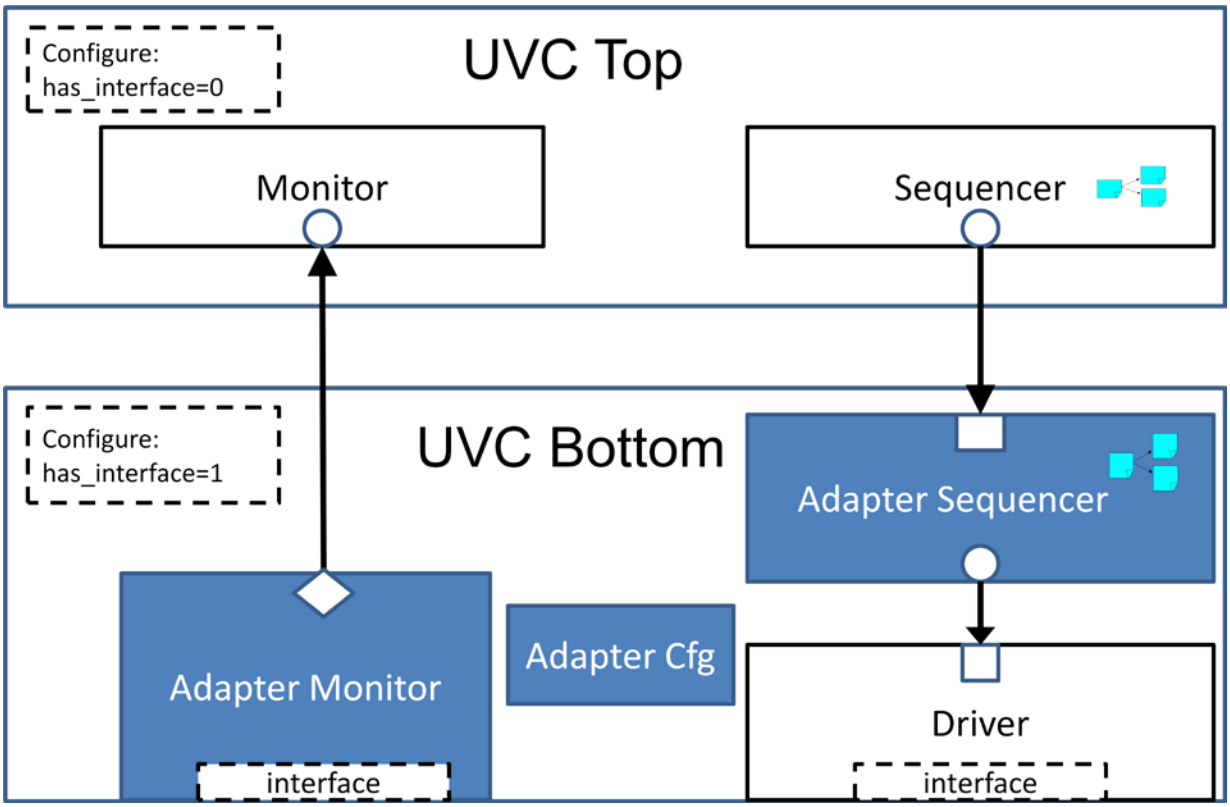
5.3 Adapter Package

The adapter is a SV package that allows a UVC to connect to one or more additional UVCs. It consists of an SV extension of the UVM's monitor, sequencer, and configuration class. TLM ports are added inside the adapter monitor and sequencer. Additionally, an adapter sequence is included inside the adapter package. Finally, transfer functions that convert the UVC transaction to/from other UVCs are included inside the adapter packages. The UVM factory mechanism is used to override the UVM monitor, sequencer, and configuration with the adapter extensions.



5-2 Adapter Package

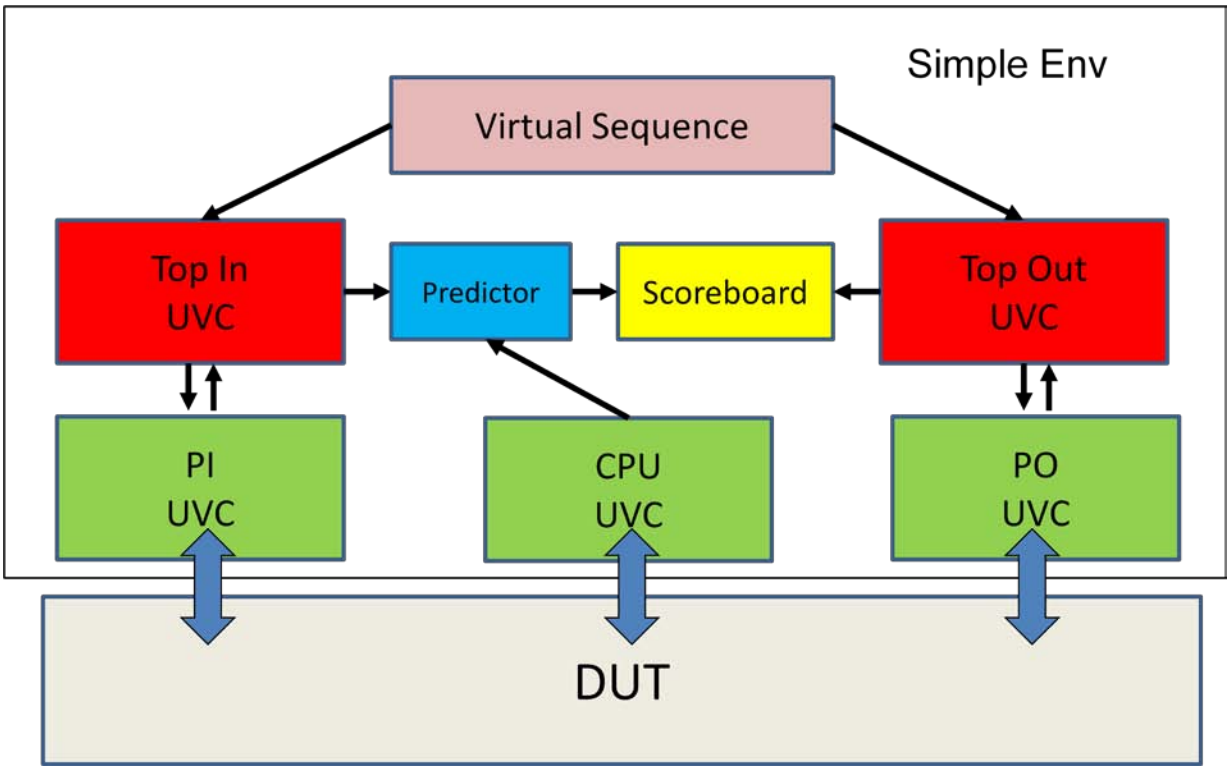
The adapters are used in the lower layer UVCs as shown below – in this example the Bottom UVC. The adapter monitor, sequencer, and configuration object are connected to the upper UVC's monitor, sequencer, and configuration object.



5-3 Stacking UVC with Adapter

6 UVC Stack Example

Below is the testbench architecture of the Paradigm Work's UVC Stack Example.



6-1 Example Stack UVC

The testbench includes the following:

- Env layer that may be reused in multiple testbenches. The Env includes a virtual sequences, instantiates the UVCs, predictor, and scoreboard component. The "ENV" code is located inside the "simple/env" directory.
- Virtual Sequence – used to coordinate sending sequences to any one of the UVCs sequencers. The testbench sequences are located inside the "simple/env/seq_lib" directory. There is a sequence for the UVM 1.x reset, configure, main, and shutdown phase is located inside this directory.
- Predictor & Scoreboard – The predictor component takes in datapath transactions from the Abstract Top In UVC and send them into the expected side of the scoreboard based on the state of the DUT. Transactions from the CPU UVC are sent into the predictor to help shadow the state of the DUT. Transactions monitored inside the Abstract Top Out UVC are sent to the actual side of scoreboard. When actual data enters the scoreboard it is compared against the actual data. If there is a mismatch then the testbench will assert an error.
- The testbench utilizes the UVM_REG feature to shadow the DUT registers. The registers are instantiated inside the Env. An initialization sequence inside file "simple/seq_lib/init_sequence.sv" and called out by the "configure" sequence. We also include an UVM_REG adapter "reg2cpu_adapter" which converts UVM REG transactions to CPU UVC transactions.

- UVCs. All the "UVCs" are located inside the "vips" directory.
  - Interface UVCs
    - PI UVC - This is the datapath "master" UVC. It drives "transactions" into the DUT. The PI UVC is located inside the "vips/pi" directory.
    - PO UVC - This is the datapath "slave" UVC. It reacts to "transaction" exiting out of the DUT. It also drives a "ready" backpressure signal. The PO UVC is located inside the "vips/po" directory.
    - CPU UVC – This is the cpu host access "master" UVC. It drives cpu read and cpu write "transactions" into the DUT. The CPU UVC is located inside the "vips/cpu" directory.
    - Interface Control UVC (not shown in the figure above) – This UVC controls asserting/de-asserting the DUT's reset signal.
  - Abstract UVCs
    - Top IN UVCs – This is a "higher level abstraction" that drives transactions into the lower layer stacked UVCs, and the predictor/scoreboard. The Top IN UVC is located inside directory "vips/top_in".
    - Top OUT UVCs – This is a "higher level abstraction" that sends reactive transactions into the lower layer stacked PO UVC and receives "slave" transactions from the lower layer stacked PO UVC. It also passes transactions onto the expect side of the scoreboard. The Top IN UVC is located inside directory "vips/top_out".
  - Adapter Packages
    - PI_STACK – This is the adapter package that extends the PI UVC and adds TLM ports and transfer functions that allows us to connect up with the TOP_IN Abstract UVC. The PI Stack is located inside directory "pi_stack".
    - PO_STACK – This is the adapter package that extends the PO UVC and adds TLM ports and transfer functions that allows us to connect up with the TOP_OUT Abstract UVC. The PO Stack is located inside directory "po_stack".
  - Scoreboard
    - Base scoreboard class. See http://www.uvmworld.org/contributions-details.php?id=100 for more details.

6.1 UVC example tests

The tests are located under the "simple/tests" directory. A base test called "simple_base_test.sv" is responsible to instantiating the ENV and setting up common test parameters. One test called "bringup.sv" is included in the testbench and it is located inside directory "simple/tests/basic". The test does not override any of the default settings in the ENV.

6.2 Running the example

There is a Makefile that allows users to run test(s). You can see all the different options by typing "help" on the command line.

>>make help

At minimum set the following:

STEP 1:

Set the variable UVM_HOME to pint to your UVM distribution.

We tested this code against UVM-1.1.

STEP 2:

Compile the test/testbench and run the simulator

To run Synopsys's VCS type:

>>make test_vcs

We tested against VCS version E-2011.03.

To run Mentor's Questa simulator type:

>>make test_mti

We tested against vsim 10.0a Simulator version 2011.02.

To run Cadence IUS type:

>>make test_ius

We tested against irun version 10.20-s009.

After the test runs you should see a message similar to the one below indicating how many packets were sent in (posted) and then checked by the scoreboard:

UVM_INFO pw_scoreboard(0) @ 21425.000 ns: uvm_test_top.simple_env_inst.packet_sb_inst [packet_sb_inst] Posted to Stream 0->0: 20

UVM_INFO pw_scoreboard(0) @ 21425.000 ns: uvm_test_top.simple_env_inst.packet_sb_inst [packet_sb_inst] Checked in Stream 0->0: 20

6.3 Example details

A detailed listing of all the files included in the testbench can be found inside file "SIMPLE_README.txt".

6.4 Example questions

If you have any questions or would like to learn about other Paradigm Work's UVM Methodologies please do not hesitate to contact us at Paradigm Works.

pw-support@paradigm-works.com