UVM-ML Quick Start Guide

For version 1.3

14 October, 2013

Notices

Questions or suggestions relating to this document or product can be sent to

support_uvm_ml@cadence.com

Contents

Introduction to UVM-ML

UVM-ML enables co-simulation of multi-language verification environments. It provides methodology and tools to integrate verification components into a larger verification environment and enables them to communicate in a coordinated co-simulation environment. It is a natural extension of the UVM-SV standard in the multi-language domain.

The central component is a ***backplane*** that enables the control and data communication between the various components using a well defined API. The various components of the verification environment are implemented in different ***frameworks*** which communicate with the backplane through ***adapters***.

This architecture can support many use cases involving integration of verification components that were developed in different languages and different methodologies. Just to mention a few:

- Incorporating a VIP developed in one language into an environment written in a different language.
- Integrating an SoC verification environment built out of legacy verification components.
- Connecting a high level stimulus generator to multiple drivers in different languages.

UVM-ML provides an open architecture that can be extended by new features. This release already supports **TLM communication** between the frameworks, **unified hierarchy** where verification components from different frameworks can be structured in a logical hierarchy, thus providing central control of the multi-language environment using **hierarchical configuration** and **hierarchical phase control**.

The UVM-ML release package contains the code and documentation needed to apply the methodology.

- Start with installing the package as described in the **README.txt** file at the top directory of the package.
- Once installed, use this **QuickStart.pdf** for first experience with the methodology using the various demos.
- Further reading is available in the white paper that provides in depth discussion of the requirements, use cases, leading concepts and terminology in **ml/docs/uvm-ml-whitepaper.pdf**.
- For the testbench integrator there is a User Guide explaining in detail how to integrate a UVM-ML verification environment in **ml/docs/IntegratorUserGuide.pdf**.
- For developers of adapters there is an Adapter Developer Guide that describes how to develop an adapter for a new framework in **ml/docs/AdapterGuide.pdf**.

The UVM-ML Package

The full UVM-ML solution is distributed in the UVM-ML *tar* file. The main components are:

- Source code for the ***backplane*** library
- UVM-SV ***framework*** and the corresponding ML adapter source code
- UVM-SC ***framework*** and the corresponding ML adapter source code
- UVM-***e*** ***framework*** and the corresponding ML adapter source code
- Setup script and `make` files
- Examples
- Documentation

If you are using the OSCI SystemC simulator you must download and install it from the Accellera downloads site.

Package Installation

To install the UVM-ML package, *untar* it in a clean directory and follow the instructions in the README.txt file in the top directory.
In case of issues with the installation consult the trouble shooting section at the end of this document.

Demos

This *Quick Start* document provides you with the minimal amount of information you need to run, and explore the results of the demos. Different demos are provided for different use cases. There is considerable repetition of explanation so select and explore the most appropriate demo for your needs.

The demos provide sample code demonstrating the use of the main capabilities provided with the library, which are the UVM-ML ***Backplane*** that connects several sample ***Frameworks*** written in different languages.

There are three demos in the `demos/prod_cons` directory.

- The demo in sc_sv demonstrates the UVM-ML SV-SC multi-language testbench
- The demo in sv_e demonstrates the UVM-ML SV-e multi-language testbench
- The demo in sc_sv_e demonstrates UVM-ML SC-SV-e multi-language testbench

Three additional demos are provided in the demos/unified_hierarchy directory.

- The demo in sc_sv demonstrates the unified hierarchy with SC IP in SV environment
- The demo in sv_e demonstrates the unified hierarchy with ***e*** IP in SV environment
- The demo in e_sv demonstrates the unified hierarchy with SV IP in ***e*** environment

The demos can be run in several modes:

- IES —When you are using the Cadence tools

- IES_OSCI—When you are using the Cadence tools with the OSCI SystemC simulator
- VCS —When you are using the Synopsys tools with the OSCI SystemC simulator
- QUESTA — when you are using the Mentor tools with the OSCI SystemC simulator

Demo 1: SC-SV

The SC-SV example is a simple environment that demonstrates the basic features of UVM-ML. In the demo, two cooperating environments, one in SystemC (SC) and one in SystemVerilog (SV) are co-simulated, and exchange data using TLM1 and TLM2 transactions, both blocking and non-blocking, in both directions.

The following figure illustrates the SC-SV architecture.

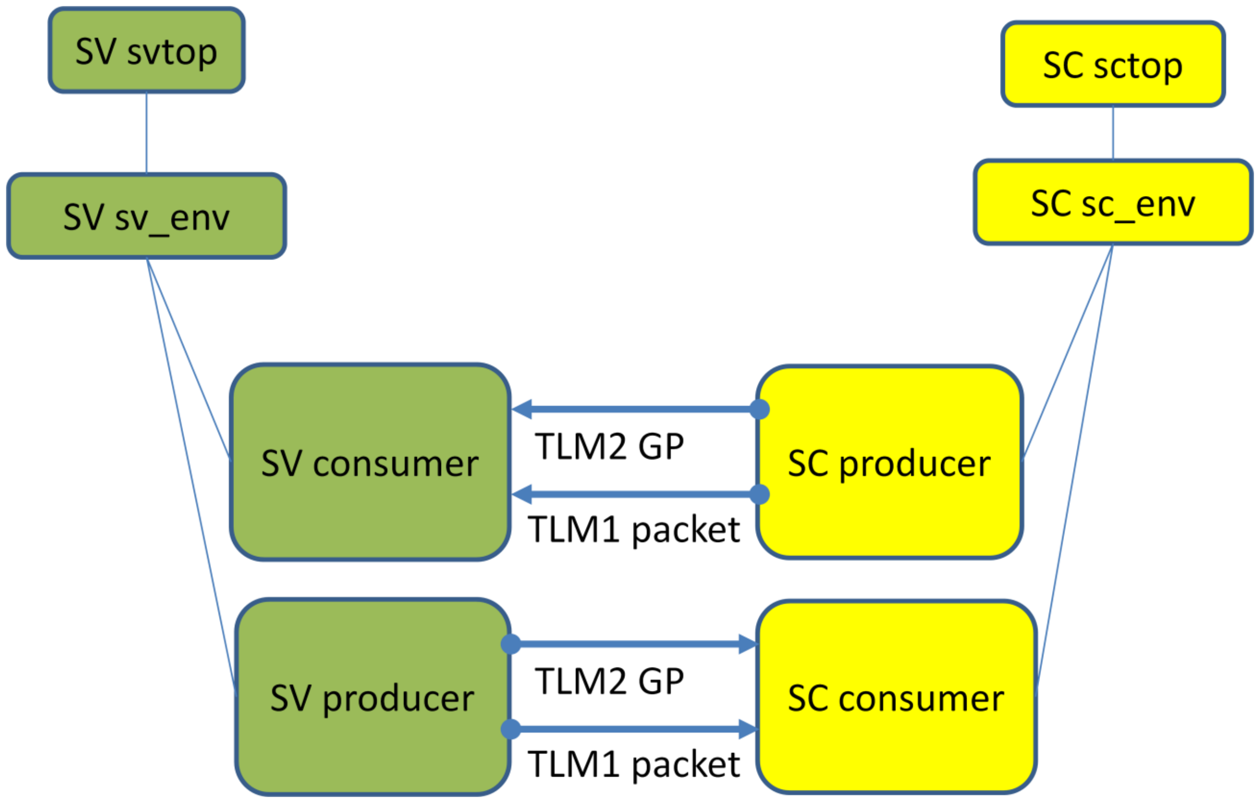

Figure 1: Demo Architecture (SV+SC)

As shown in Figure 1, each environment has an `env` component. The `env` components contain *producer* and *consumer* components. The producers generate data and send it through the various ports and sockets, while the consumers react to the data using both the blocking and non-blocking interfaces.

SC-SV Demo Location

Before you run the SC-SV demo, ensure that the installation and setup steps listed in the *UVM Package Installation* section were successful.

To run the demo, first go to the demo directory using the following command line, and then follow the instructions in the section, Running the SC-SV Demo:

```
% cd $UVM_ML_HOME/ml/examples/demos/prod_cons/sc_sv
```

The following files can be found in the demo directory:

- README.txt—Instructions for running the demo
- demo.sh—A shell script to invoke the demo
- packet.h—An SC definition of the data passed on TLM1 interfaces
- producer.h—An SC definition of the initiator of transactions
- consumer.h—An SC definition of the target for the transactions
- sc.cpp—The SC top level
- packet.sv—An SV definition of the data passed on TLM1 interface
- producer.sv—An SV definition of the initiator of transactions
- consumer.sv—An SV definition of the target for the transactions
- test.sv—The SV top level
- Makefile—A *makefile to build and run the demo*
- Makefile.ius —A *makefile* used for IES simulation
- Makefile.vcs—A *makefile* used for VCS simulation
- Makefile.questa – A *makefile* used for Questa simulation

Running the SC-SV Demo

To run the demo using Cadence tools, use the following command line:

```
% demo.sh IES_NCSC
```

To run the demo using Cadence tools with OSCI SystemC, use the following command line:

```
% demo.sh IES_OSCI
```

To run the demo with Synopsys tools and OSCI SystemC, use the following command line:

```
% demo.sh VCS
```

To run the demo with Mentor tools and OSCI SystemC, use the following command line:

```
% demo.sh QUESTA
```

SC-SV Demo Results

The demo executes in batch mode and the output shows eight sections of transactions passing from producer to consumer, which represent all combinations of blocking/non-blocking, TLM1/TLM2, and SC/SV as either initiator or target, including:

- Non-blocking TLM2 transactions from SC
- Blocking TLM2 transactions from SC
- Non-blocking TLM1 transactions from SC
- Blocking TLM1 transactions from SC

- Non-blocking TLM1 transactions from SV
- Blocking TLM1 transactions from SV
- Non-blocking TLM2 transactions from SV
- Blocking TLM2 transactions from SV

The following listing is a small example from the log file that is generated by the demo.

```
SC env::env
SC sctop::sctop
UVM_INFO @ 0: svtop [svtop] SV svtop::new
SC producer::build
SC consumer::build
UVM_INFO @ 0: svtop [svtop] SV svtop::build
UVM_INFO @ 0: svtop.sv_env [env] SV env::new
UVM_INFO @ 0: svtop.sv_env [env] SV env::build
UVM_INFO @ 0: svtop.sv_env.consumer [cons_type] SV consumer::new
UVM_INFO @ 0: svtop.sv_env.producer [prod_type] SV producer::new
UVM_INFO @ 0: svtop.sv_env.consumer [cons_type] SV consumer::build
UVM_INFO @ 0: svtop.sv_env.producer [prod_type] SV producer::build
UVM_INFO @ 0: svtop [svtop] SV svtop::build phase ended
SC sctop::before_end_of_elaboration
SC env::before_end_of_elaboration
UVM_INFO @ 0: svtop.sv_env.consumer [cons_type] SV consumer::connect
UVM_INFO @ 0: svtop.sv_env.producer [prod_type] SV producer::connect
UVM_INFO @ 0: svtop.sv_env [env] SV env::connect
UVM_INFO @ 0: svtop [svtop] SV svtop::connect
UVM_INFO @ 0: svtop [svtop] SV svtop:run_phase

TLM2 non-blocking transactions from SC
[SC 0 s] creating transaction with base = 1
[SC 5 ns] producer::nb_transport_fw  address = 457 data_length = 2 m_data = 0 1  status= 0
UVM_INFO @ 5: svtop.sv_env.consumer [cons_type] SV nb_transport_fw  UVM_TLM_WRITE_COMMAND
[0x0000000000000457] = 00 01 (status=INCOMPLETE)
[SC 5 ns] nb_transport_bw
[SC 5 ns] creating transaction with base = 1
UVM_INFO @ 10: svtop.sv_env.consumer [cons_type] SV nb_transport_fw  UVM_TLM_READ_COMMAND
[0x0000000000000457] = 00 00 (status=INCOMPLETE)
[SC 10 ns] nb_transport_bw
[SC 10 ns] producer::nb_transport_fw  address = 457 data_length = 2 m_data = 0 1  status= 0

TLM2 blocking transactions from SC
…
** UVM TEST PASSED **
```

At the beginning of the demo output, you can see the trace of the coordinated construction of the ML testbench, in the two language frameworks, SC and SV.

In each one of the runtime sections, you can see messages from both frameworks, including the time stamp indicating coordinated simulation. The first section in the listing above shows non-blocking TLM2 transactions initiated in the SystemC framework and received in the SystemVerilog framework. The address is randomized and the data is a variable list of integers. The log shows that the transaction received in the SystemVerilog framework is identical to that generated in the SystemC framework.

Demo 2: SV-e

The SV-e demo is similar to the SC-SV demo described above, with some minor differences.

Note: This demo requires the Cadence Incisive release: IES12.20-s012 or higher

SV-e Demo Location

Before you run the SV-e demo, ensure that the installation and setup steps listed in the *UVM Package Installation* section were successful.

To run the demo, first go to the demo directory using the following command line, and then follow the instructions in the section Running the SV-e Demo:

```
    % cd $UVM_ML_HOME/ml/examples/demos/prod_cons/sv_e
```

The following files can be found in the demo directory:

- README.txt—Instructions for running the demo
- consumer.e – ***e*** code for the consumer
- consumer.sv – SV code for the consumer
- demo.sh—A shell script to invoke the demo
- packet.e – ***e*** code defining the TLM1 packet
- packet.sv – SV code defining the TLM1 packet
- producer.e – ***e*** code for the producer
- producer.sv –SV code for the producer
- test.sv—SV code for the top components
- top.e — ***e*** code for the top components
- Makefile—A *makefile* to *build and run the demo*
- Makefile.ies—A *makefile* used for IES simulation
- Makefile.vcs—A *makefile* used for VCS simulation
- Makefile.questa – A *makefile* used for Questa simulation

Running the SV-e Demo

Make sure the installation and setup steps listed in the **Error! Reference source not found.** section were successful before you run the demo.

To run the demo using Cadence tools, use the following command line:

```
    % demo.sh IES
```

To run the demo with Synopsys tools and OSCI SystemC, use the following command line:

```
    % demo.sh VCS
```

To run the demo with Mentor tools and OSCI SystemC, use the following command line:

```
    % demo.sh QUESTA
```

SV-e Demo Results

The demo executes in batch mode, and the output shows eight sections of transactions passing from producer to consumer, which represent all combinations of blocking/non-blocking, TLM1/TLM2, and SV/***e*** as either initiator or target, including:

- Non-blocking TLM2 transactions from SV

- Blocking TLM2 transactions from SV
- Non-blocking TLM1 transactions from SV
- Blocking TLM1 transactions from SV
- Non-blocking TLM2 transactions from ***e***
- Blocking TLM2 transactions from ***e***
- Non-blocking TLM1 transactions from ***e***
- Blocking TLM1 transactions from ***e***

The following listing is a small example from the log file that is generated by the demo.

```
Running the test ...

*** Starting non-blocking TLM2 transactions from SV
UVM_INFO @ 0: svtop.sv_env.producer [prod_type] SV producer sends                   55:  28  45
251  45
[0] consumer-@7: Received nb_transport_fw 55: 28 45 251 45
UVM_INFO @ 5: svtop.sv_env.producer [prod_type] SV producer sends                  198:  40 128
244  34
[5] consumer-@7: Received nb_transport_fw 198: 40 128 244 34
UVM_INFO @ 10: svtop.sv_env.producer [prod_type] SV producer sends                  68: 106 214
172  82
[10] consumer-@7: Received nb_transport_fw 68: 106 214 172 82
UVM_INFO @ 15: svtop.sv_env.producer [prod_type]

*** Starting blocking TLM2 transactions from SV
…
** UVM TEST PASSED **
```

In each section you can see messages from the two language frameworks, SV and ***e***, including the time stamps indicating coordinated simulation.

The first section in the listing above, shows non-blocking TLM2 transactions generated in the SystemVerilog framework. The transactions containing random address and data are sent to the UVM-***e*** framework. The log shows that the received transaction is identical to the original transaction.

Demo 3: SC-SV-e

The SC-SV-e demo connects all three environments using blocking and non-blocking TLM1 and TLM2 interfaces.

Note: This demo requires the Cadence Incisive release: IES12.20-s007

The following figure illustrates the SC-SV-e architecture.

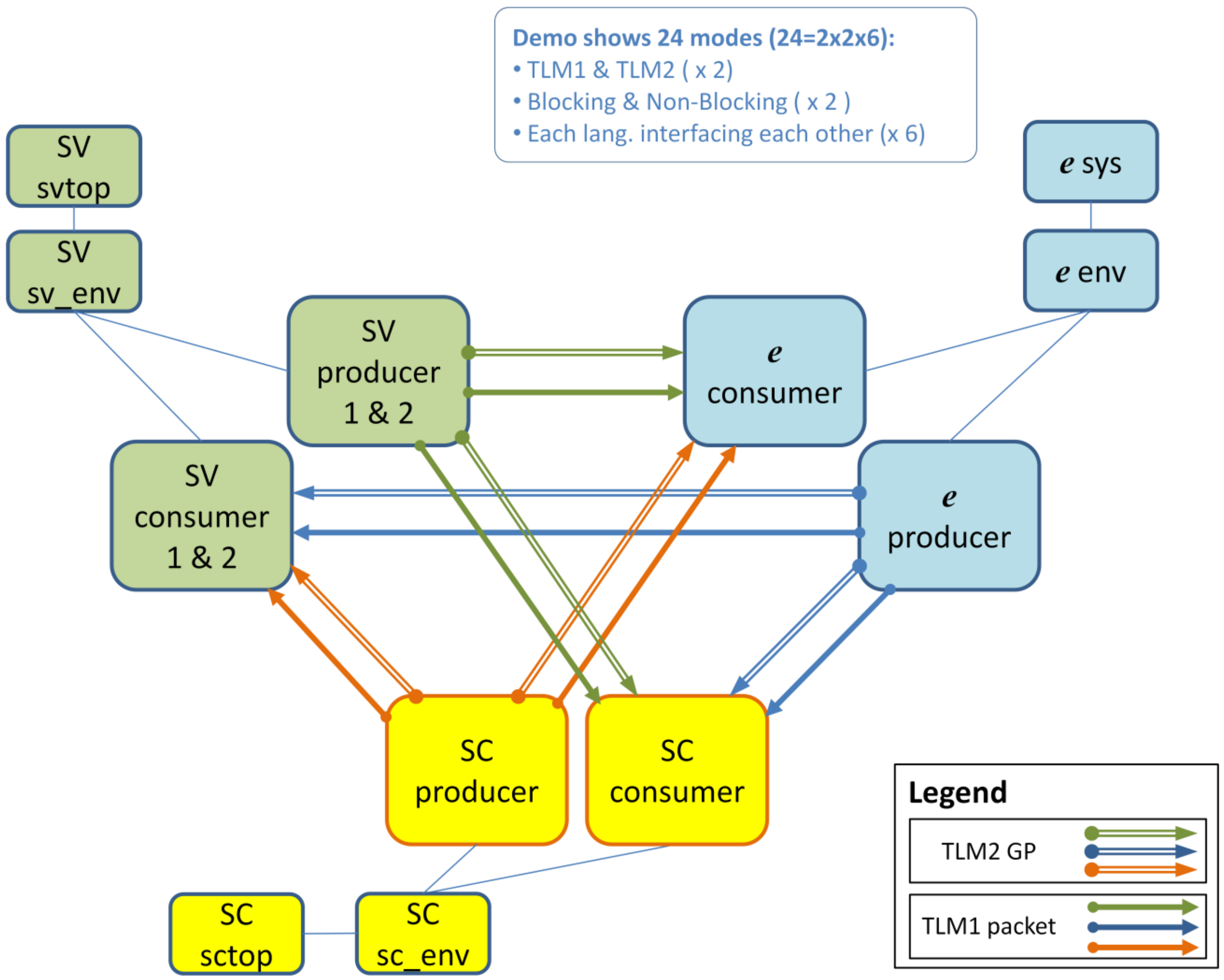

*Figure 2: Demo Architecture (SV+SC+**e**)*

SC-SV-e Demo Location

Before you run the SC-SV-e demo, ensure that the installation and setup steps listed in the *UVM Package Installation* section were successful.

To run the demo, first go to the demo directory using the following command line, and then follow the instructions in the section Running the SV-e DemoRunning the SC-SV-e Demo:

```
% cd $UVM_ML_HOME/ml/examples/demos/prod_cons/sc_sv_e
```

The following files can be found in the demo directory:

- README.txt—Instructions for running the demo
- consumer.e – ***e*** code for the consumer
- consumer.h – SC code for the consumer
- consumer.sv - SV code for the consumer
- demo.sh—A shell script to invoke the demo
- packet.e – ***e*** code for the TLM1 packet

- packet.h - SC code for the TLM1 packet
- packet.sv – SV code for the TLM1 packet
- producer.e – ***e*** code for the producer
- producer.h – SC code for the producer
- producer.sv – SV code for the producer
- sc.cpp – SC code for the top components
- test.sv—SV code for the top components
- top.e — ***e*** code for the top components
- Makefile—The *makefile* to build and run the demo
- Makefile.ies—A *makefile* used for IES simulation
- Makefile.vcs—A *makefile* used for VCS simulation
- Makefile.questa – A *makefile* used for Questa simulation

Running the SC-SV-e Demo

Make sure the installation and setup steps listed in the **Error! Reference source not found.** section were successful before you run the demo.

To run the demo using Cadence tools, use the following command line:

```
% demo.sh IES
```

To run the demo with Synopsys tools and OSCI SystemC, use the following command line:

```
% demo.sh VCS
```

To run the demo with Mentor tools and OSCI SystemC, use the following command line:

```
% demo.sh QUESTA
```

SC-SV-e Demo Results

The demo executes in batch mode, and the output shows eight sections of transactions passing from producer to consumer, which represent all combinations of blocking/non-blocking, TLM1/TLM2, and SV/SC and SV/***e*** as either initiator or target, including:

- non-blocking TLM2 transactions from SV to ***e***
- blocking TLM2 transactions from SV to ***e***
- non-blocking TLM1 transactions from SV to ***e***
- blocking TLM1 transactions from SV to ***e***
- non-blocking TLM2 transactions from SV to SC
- blocking TLM2 transactions from SV to SC
- non-blocking TLM1 transactions from SV to SC
- blocking TLM1 transactions from SV to SC
- non-blocking TLM2 transactions from ***e*** to SV
- blocking TLM2 transactions from ***e*** to SV
- non-blocking TLM1 transactions from ***e*** to SV

- blocking TLM1 transactions from ***e*** to SV
- blocking TLM1 transactions from ***e*** to SC
- non-blocking TLM1 transactions from ***e*** to SC
- blocking TLM2 transactions from ***e*** to SC
- non-blocking TLM2 transactions from ***e*** to SC
- non-blocking TLM2 transactions from SC to SV
- blocking TLM2 transactions from SC to SV
- non-blocking TLM1 transactions from SC to SV
- blocking TLM1 transactions from SC to SV
- blocking TLM2 transactions from SC to ***e***
- non-blocking TLM2 transactions from SC to ***e***
- blocking TLM1 transactions from SC to ***e***
- non-blocking TLM1 transactions from SC to ***e***

The following listing is a small example from the log file that is generated by the demo.

```
Running the test ...
SC sctop::start_of_simulation

*** Starting non-blocking TLM2 transactions from SV to e
UVM_INFO @ 0: uvm_test_top.sv_env.producer_1 [producer_1] SV producer sends                     64:
33 cb 67 08
[0] consumer-@7: Received nb_transport_fw WRITE 0x33 0xcb 0x67 0x08
UVM_INFO @ 5: uvm_test_top.sv_env.producer_1 [producer_1] SV producer sends                     64:
00 00 00 00
[5] consumer-@7: Received nb_transport_fw READ 0x33 0xcb 0x67 0x08
UVM_INFO @ 5: uvm_test_top.sv_env.producer_1 [producer_1] SV producer received
64: 33 cb 67 08
UVM_INFO @ 10: uvm_test_top.sv_env.producer_1 [producer_1]

…
*** Starting Blocking TLM1 transactions from SC to SV
[390 ns] SC producer::sending T packet: 17
UVM_INFO @ 390: uvm_test_top.sv_env.consumer_2 [consumer_2] SV consumer::put          17
UVM_INFO @ 391: uvm_test_top.sv_env.consumer_2 [consumer_2] SV consumer::put returns          17
[391 ns] SC producer::sent T packet: 17
[395 ns] SC producer::sending T packet: 18
UVM_INFO @ 395: uvm_test_top.sv_env.consumer_2 [consumer_2] SV consumer::put          18
UVM_INFO @ 396: uvm_test_top.sv_env.consumer_2 [consumer_2] SV consumer::put returns          18
[396 ns] SC producer::sent T packet: 18
[400 ns] SC producer::sending T packet: 19
UVM_INFO @ 400: uvm_test_top.sv_env.consumer_2 [consumer_2] SV consumer::put          19
UVM_INFO @ 401: uvm_test_top.sv_env.consumer_2 [consumer_2] SV consumer::put returns          19
[401 ns] SC producer::sent T packet: 19
...

** UVM TEST PASSED **
```

In each section you can see messages from two language frameworks, including the time stamps indicating coordinated simulation.

The first section in the listing above, shows non-blocking TLM2 transactions generated in the SystemVerilog framework, targeted to the ***e*** consumer. The transactions containing random address and data are written to the ***e*** framework and then read back, comparing the result to the original transaction.

Demo 4: Unified hierarchy SC-SV

The SC-SV unified hierarchy demo shows how a UVM-SC verification IP can be incorporated in a UVM-SV environment.

The following figure illustrates the architecture of this demo.

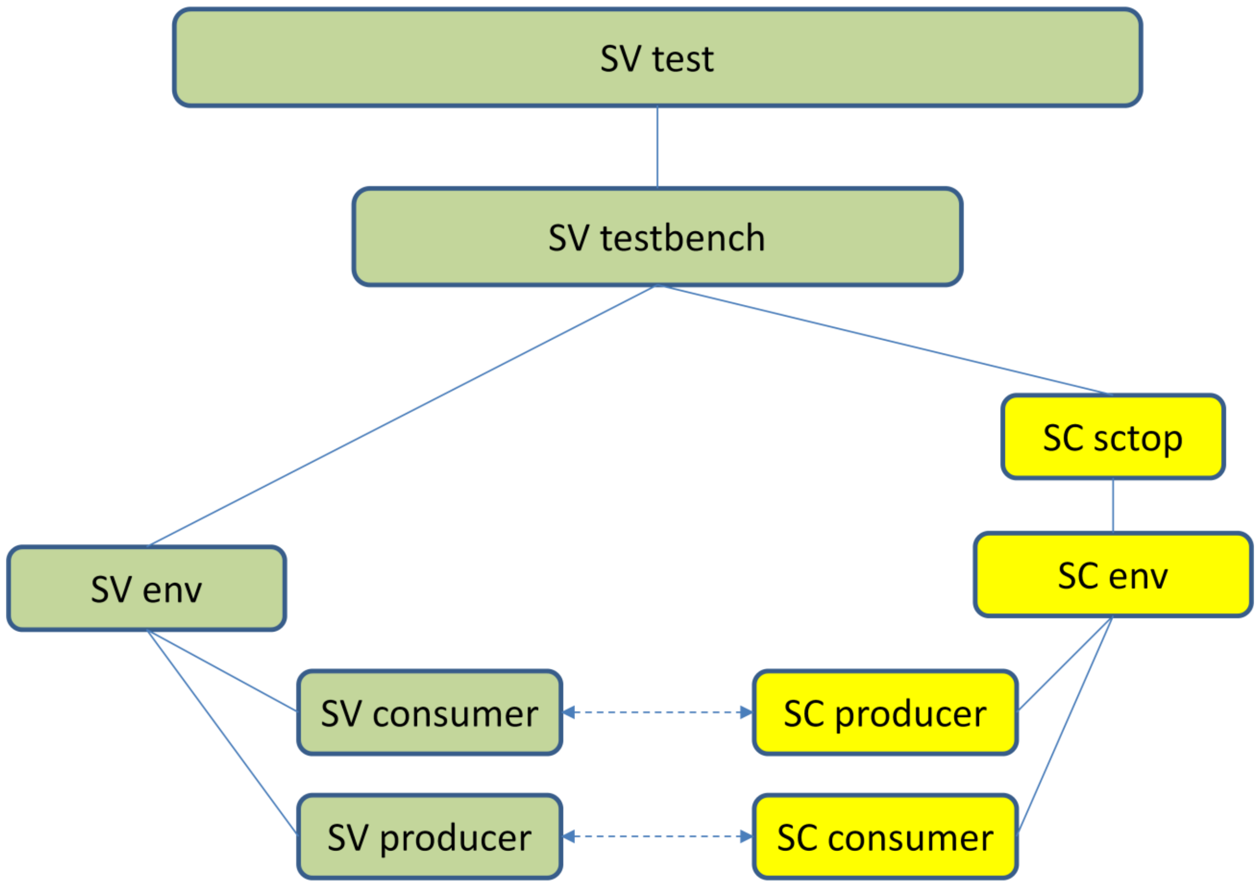

Figure 3: Demo Architecture for SC+SV unified hierarchy

Unified Hierarchy Demo Location

Before you run the demo, ensure that the installation and setup steps listed in the *UVM Package Installation* section were successful.

To run the demo, first go to the demo directory using the following command line, and then follow the instructions in the section Running the Unified Hierarchy SC-SV Demo:

```
% cd $UVM_ML_HOME/ml/examples/demos/unified_hierarchy/sc_sv
```

The following files can be found in the demo directory:

- README.txt—Instructions for running the demo

- consumer.h – SC code for the consumer
- consumer.sv - SV code for the consumer
- demo.sh—A shell script to invoke the demo
- producer.h – SC code for the producer
- producer.sv – SV code for the producer
- consumer.h – SC code for the consumer
- consumer.sv – SV code for the consumer
- sctop.cpp – SC code for the top components
- test.sv—SV code for the top components
- Makefile—The *makefile* to build and run the demo
- Makefile.ies—A *makefile* used for IES simulation
- Makefile.vcs—A *makefile* used for VCS simulation
- Makefile.questa – A *makefile* used for Questa simulation

Running the Unified Hierarchy SC-SV Demo

Make sure the installation and setup steps listed in the **Error! Reference source not found.** section were successful before you run the demo.

To run the demo using Cadence tools, use the following command line:

```
% demo.sh IES
```

To run the demo with Synopsys tools and OSCI SystemC, use the following command line:

```
% demo.sh VCS
```

To run the demo with Mentor tools and OSCI SystemC, use the following command line:

```
% demo.sh QUESTA
```

Unified Hierarchy Demo Results

The demo executes in batch mode, and the output shows the order of construction of the ML environment and the propagation of phases from the UVM-SV environment into the UVM-SC VIP.

The following listing is a small example from the log file that is generated by the demo.

```
UVM_INFO @ 0: uvm_test_top [test] SV test::new uvm_test_top
UVM_INFO @ 0: uvm_test_top [test] SV test::build uvm_test_top
UVM_INFO @ 0: uvm_test_top.sv_env [env] SV env::new sv_env
UVM_INFO @ 0: uvm_test_top.testbench [testbench] SV env::new testbench
UVM_INFO @ 0: uvm_test_top.sv_env [env] SV env::build
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::new
UVM_INFO @ 0: uvm_test_top.sv_env.producer [prod_type] SV producer::new
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::build
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::b_target_socket.get_full_name
= uvm_test_top.sv_env.consumer.b_target_socket
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV
consumer::nb_target_socket.get_full_name = uvm_test_top.sv_env.consumer.nb_target_socket
UVM_INFO @ 0: uvm_test_top.sv_env.producer [prod_type] SV producer::build
UVM_INFO @ 0: uvm_test_top.testbench [testbench] SV testbench::build
SC producer::producer name= uvm_test_top.testbench.sctop.sc_env.prod
SC env::env name= uvm_test_top.testbench.sctop.sc_env
SC sctop::sctop name= uvm_test_top.testbench.sctop type= sctop
SC sctop::build uvm_test_top.testbench.sctop
SC env::build uvm_test_top.testbench.sctop.sc_env
SC producer::build uvm_test_top.testbench.sctop.sc_env.prod
SC consumer::build
UVM_INFO @ 0: uvm_test_top [test] SV test registered uvm_test_top.sv_env.consumer.b_target_socket
UVM_INFO @ 0: uvm_test_top [test] SV test registered
uvm_test_top.sv_env.consumer.nb_target_socket
UVM_INFO @ 0: uvm_test_top [test] SV test registered
uvm_test_top.sv_env.producer.b_initiator_socket
UVM_INFO @ 0: uvm_test_top [test] SV test registered
uvm_test_top.sv_env.producer.nb_initiator_socket
SC sctop::before_end_of_elaboration uvm_test_top.testbench.sctop
SC env::before_end_of_elaboration uvm_test_top.testbench.sctop.sc_env
SC env registered uvm_test_top.testbench.sctop.sc_env.prod.b_isocket
SC env registered uvm_test_top.testbench.sctop.sc_env.prod.nb_isocket
SC env registered uvm_test_top.testbench.sctop.sc_env.cons.tsocket
SC producer::before_end_of_elaboration uvm_test_top.testbench.sctop.sc_env.prod
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::connect
uvm_test_top.sv_env.consumer
UVM_INFO @ 0: uvm_test_top.sv_env.producer [prod_type] SV producer::connect
UVM_INFO @ 0: uvm_test_top.sv_env [env] SV env::connect uvm_test_top.sv_env
SC producer::connect uvm_test_top.testbench.sctop.sc_env.prod
SC env::connect uvm_test_top.testbench.sctop.sc_env
SC sctop::connect uvm_test_top.testbench.sctop
UVM_INFO @ 0: uvm_test_top.testbench [testbench] SV testbench::connect uvm_test_top.testbench
UVM_INFO @ 0: uvm_test_top [test] SV test::connect uvm_test_top
...
UVM_INFO @ 10: uvm_test_top.sv_env.producer [prod_type] SV producer::b_transport
UVM_TLM_WRITE_COMMAND [0x00000000000000bb] = 7a e3 f2 4f (status=INCOMPLETE)
[SC 12 ns] consumer::b_transport TLM_WRITE_COMMAND address = 187 data_length = 4 m_data = 7a e3
f2 4f  status= 1
UVM_INFO @ 12: uvm_test_top.sv_env.producer [prod_type] SV producer::b_transport done
UVM_TLM_WRITE_COMMAND [0x00000000000000bb] = 7a e3 f2 4f (status=OK)
UVM_INFO @ 20: uvm_test_top.sv_env.producer [prod_type] SV producer::b_transport
UVM_TLM_READ_COMMAND [0x00000000000000bb] = 00 00 00 00 (status=INCOMPLETE)
[SC 22 ns] consumer::b_transport TLM_READ_COMMAND address = bb data_length = 4 m_data = 7a e3 f2
4f  status= 1
UVM_INFO @ 22: uvm_test_top.sv_env.producer [prod_type] SV producer::b_transport done
UVM_TLM_READ_COMMAND [0x00000000000000bb] = 7a e3 f2 4f (status=OK)
[SC 30 ns] producer::b_transport TLM_WRITE_COMMAND address = 7f data_length = 4 m_data = e6 ea 53
fe  status= 0
UVM_INFO @ 30: uvm_test_top.sv_env.consumer [cons_type] SV consumer::b_transport
UVM_TLM_WRITE_COMMAND [0x000000000000007f] = e6 ea 53 fe (status=INCOMPLETE)
[SC 32 ns] producer::b_transport done TLM_WRITE_COMMAND address = 7f data_length = 4 m_data = e6
ea 53 fe  status= 1
[SC 40 ns] producer::b_transport TLM_READ_COMMAND address = 7f data_length = 4 m_data = 0 0 0 0
status= 0
UVM_INFO @ 40: uvm_test_top.sv_env.consumer [cons_type] SV consumer::b_transport
UVM_TLM_READ_COMMAND [0x000000000000007f] = 00 00 00 00 (status=INCOMPLETE)
[SC 42 ns] producer::b_transport done TLM_READ_COMMAND address = 7f data_length = 4 m_data = e6
ea 53 fe  status= 1
UVM_INFO @ 42: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
...
```

The first section in the listing above, shows the pre-run phases as they alternate between the SV and SC frameworks as the environment is constructed and then the ports bound across the frameworks.

Demo 5: Unified hierarchy SV-*e*

This demo shows how a UVM-***e*** verification IP can be instantiated under a UVM-SV environment and how to use configuration to control the construction of the UVM-***e*** IP.

The following figure illustrates the architecture of this demo.

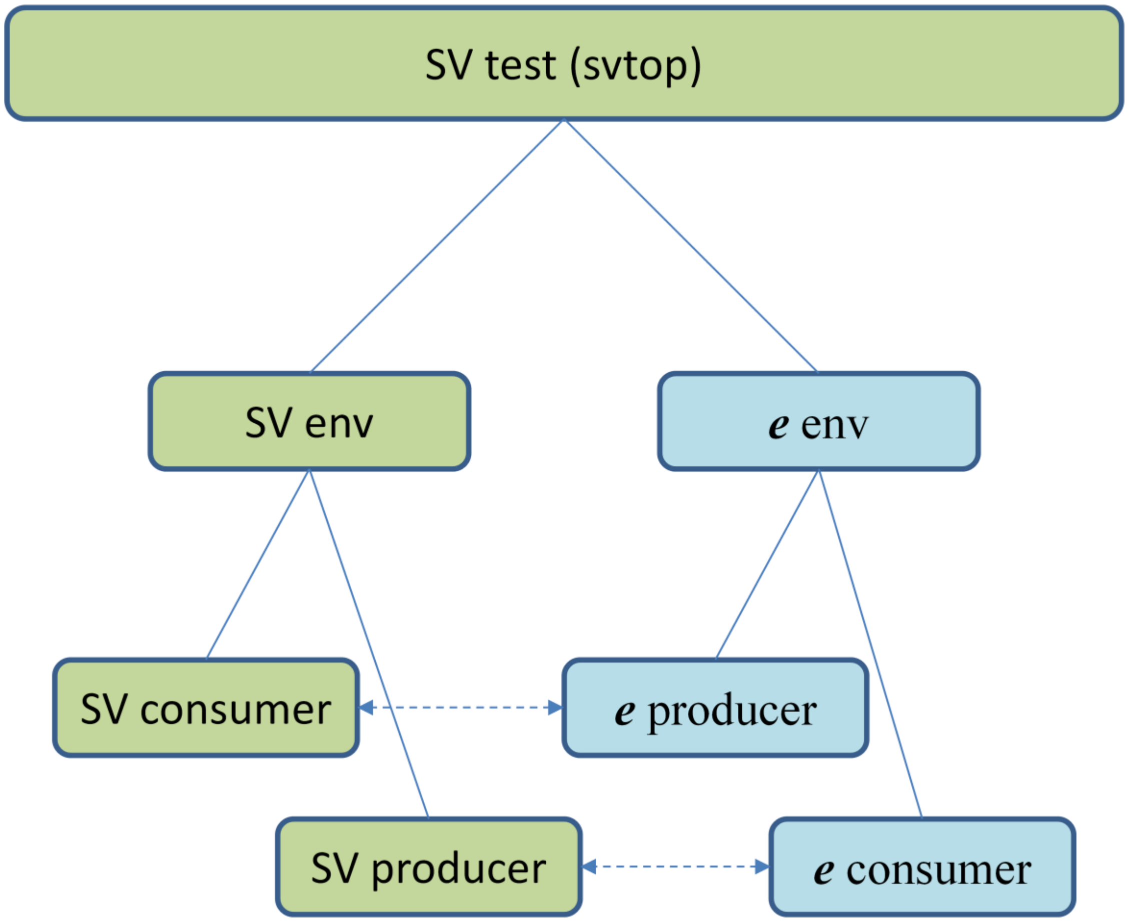

*Figure 4: Demo Architecture for SV+**e** unified hierarchy*

Unified Hierarchy Demo Location

Before you run the demo, ensure that the installation and setup steps listed in the *UVM Package Installation* section were successful.

To run the demo, first go to the demo directory using the following command line, and then follow the instructions in the section Running the Unified Hierarchy SV-e Demo:

```
% cd $UVM_ML_HOME/ml/examples/demos/unified_hierarchy/sv_e
```

The following files can be found in the demo directory:

- README.txt—Instructions for running the demo
- demo.sh—A shell script to invoke the demo
- consumer.e – ***e*** code for the consumer
- consumer.sv - SV code for the consumer
- env.e – ***e*** IP environment code
- env.sv – SV IP environment code
- producer.e – ***e*** code for the producer
- producer.sv – SV code for the producer
- svtop.sv—SV code for the top component
- Makefile—The *makefile* to build and run the demo
- Makefile.ies—A *makefile* used for IES simulation
- Makefile.vcs—A *makefile* used for VCS simulation
- Makefile.questa – A *makefile* used for Questa simulation

Running the Unified Hierarchy SV-e Demo

Make sure the installation and setup steps listed in the section were successful before you run the demo.

To run the demo using Cadence tools, use the following command line:

```
% demo.sh IES
```

To run the demo with Synopsys tools and OSCI SystemC, use the following command line:

```
% demo.sh VCS
```

To run the demo with Mentor tools and OSCI SystemC, use the following command line:

```
% demo.sh QUESTA
```

Unified Hierarchy Demo Results

The demo executes in batch mode, and the output shows the order of construction of the ML environment and the propagation of phases from the UVM-SV environment into the UVM-SV and UVM-***e*** verification IP.

This demo also demonstrates the configuration capability by showing how the UVM-SV test component controls the construction of the UVM-***e*** VIP. In its build phase, the test component sets a random value for the address to be used in both producer components. In addition it controls the generation of both IP using the *_active configuration attribute. When set to FALSE, the IP will not instantiate its producer, and therefore will not generate transactions.

```
set_config_int("*producer","address", ($urandom() & 'hffff));
set_config_int("*env","e_active",e_active);
set_config_int("*env","sv_active",sv_active);
```

The following listing is a small example from the log file that is generated by the demo.

```
UVM_INFO @ 0: uvm_test_top [svtop] SV svtop::new uvm_test_top
Generating the test with IntelliGen using seed 1...
UVM_INFO @ 0: uvm_test_top [svtop] SV svtop::build
UVM_INFO @ 0: uvm_test_top.sv_env [sv_env] SV sv_env::new sv_env
UVM_INFO @ 0: uvm_test_top.sv_env [sv_env] SV sv_env::build
UVM_INFO @ 0: uvm_test_top.sv_env.producer [prod_type] SV producer::new
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::new
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::build
UVM_INFO @ 0: uvm_test_top.sv_env.producer [prod_type] SV producer::build
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::connect
uvm_test_top.sv_env.consumer
UVM_INFO @ 0: uvm_test_top [svtop] SV svtop::connect uvm_test_top
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::resolve_bindings
uvm_test_top.sv_env.consumer
UVM_INFO @ 0: uvm_test_top [svtop] SV env::resolve_bindings uvm_test_top
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::end_of_elaboration
uvm_test_top.sv_env.consumer
UVM_INFO @ 0: uvm_test_top [svtop] SV env::end_of_elaboration uvm_test_top
Starting the test ...
Running the test ...
UVM_INFO @ 0: uvm_test_top.sv_env.consumer [cons_type] SV consumer::start_of_simulation
uvm_test_top.sv_env.consumer
UVM_INFO @ 0: uvm_test_top [svtop] SV env::start_of_simulation uvm_test_top

*** Starting non-blocking TLM2 transactions from e
[10] producer-@6: Calling nb_transport_fw WRITE 0xd0 0xe4 0x59 0x60
UVM_INFO @ 10: uvm_test_top.sv_env.consumer [cons_type] SV consumer::nb_transport_fw
UVM_TLM_WRITE_COMMAND [0x0000000000004123] = d0 e4 59 60 (status=INCOMPLETE)
UVM_INFO @ 15: uvm_test_top.sv_env.consumer [cons_type] SV consumer responds
UVM_TLM_WRITE_COMMAND [0x0000000000004123] = d0 e4 59 60 (status=OK)
[15] producer-@6: Received nb_transport_bw TLM_OK_RESPONSE phase= BEGIN_RESP
[20] producer-@6: Calling nb_transport_fw READ 0x00 0x00 0x00 0x00
UVM_INFO @ 20: uvm_test_top.sv_env.consumer [cons_type] SV consumer::nb_transport_fw
UVM_TLM_READ_COMMAND [0x0000000000004123] = 00 00 00 00 (status=INCOMPLETE)
[20] producer-@6: nb_transport READ returned 0xd0 0xe4 0x59 0x60
UVM_INFO @ 25: uvm_test_top.sv_env.consumer [cons_type] SV consumer responds
UVM_TLM_READ_COMMAND [0x0000000000004123] = d0 e4 59 60 (status=OK)
[25] producer-@6: Received nb_transport_bw TLM_OK_RESPONSE phase= BEGIN_RESP

*** Starting blocking TLM2 transactions from e

...
```

The output shows the construction of the testbench starting with the UVM-SV test component. Both the UVM-***e*** IP and the UVM-SV IP are generated from the testbench during the build phase. The test component controls the construction of both IP using the configuration mechanism.

During the run phase the IP are communicating by passing blocking and non-blocking transactions in both directions.

Demo 6: Unified hierarchy *e*-SV

This demo shows how a UVM-SV verification IP can be instantiated under a UVM-***e*** environment and how to use configuration to control the construction of the UVM-SV IP.

The following figure illustrates the architecture of this demo.

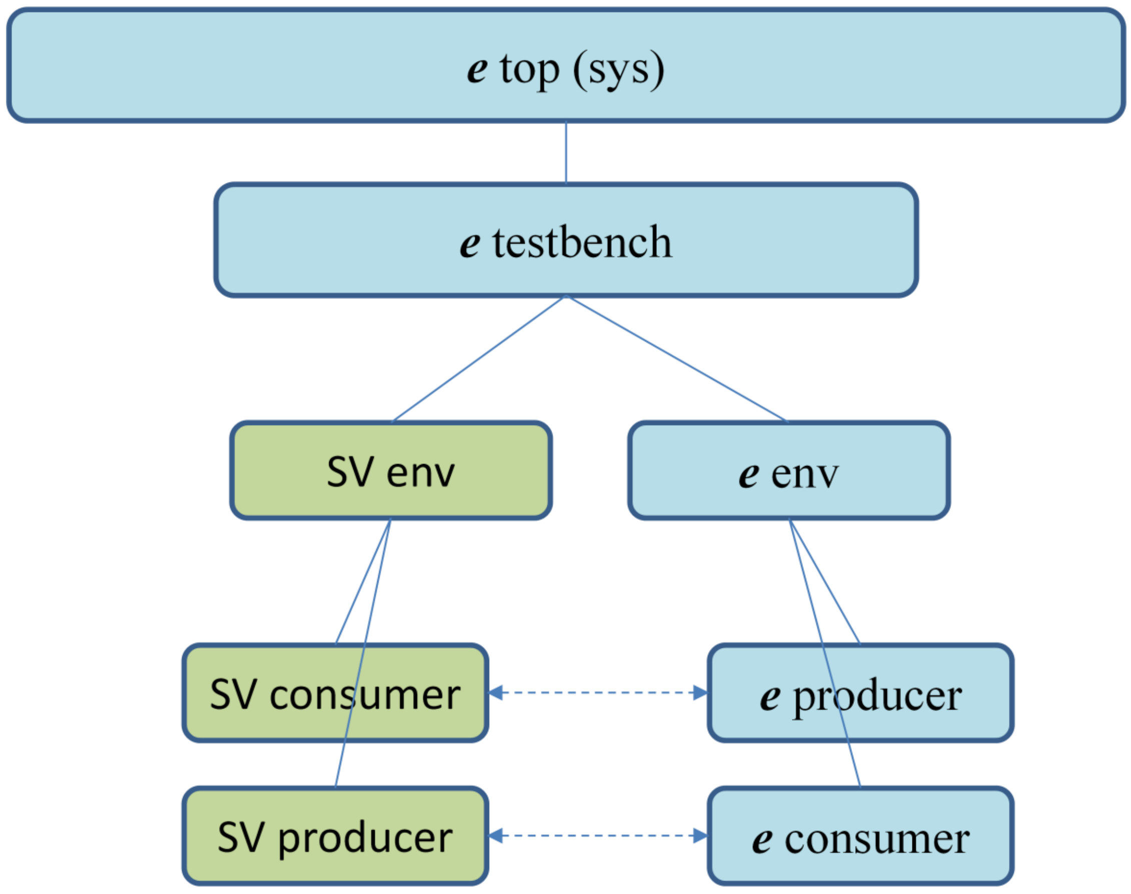

*Figure 5: Demo Architecture for **e**+SV unified hierarchy*

Unified Hierarchy Demo Location

Before you run the demo, ensure that the installation and setup steps listed in the *UVM Package Installation* section were successful.

To run the demo, first go to the demo directory using the following command line, and then follow the instructions in the section Running the Unified Hierarchy e-SV Demo:

```
% cd $UVM_ML_HOME/ml/examples/demos/unified_hierarchy/e_sv
```

The following files can be found in the demo directory:

- README.txt—Instructions for running the demo
- demo.sh—A shell script to invoke the demo
- consumer.e – ***e*** code for the consumer
- consumer.sv - SV code for the consumer
- env.e – ***e*** IP environment code
- svtop.sv – SV IP environment code
- producer.e – ***e*** code for the producer
- producer.sv – SV code for the producer
- Makefile—The *makefile* to build and run the demo

- Makefile.ies—A *makefile* used for IES simulation
- Makefile.vcs—A *makefile* used for VCS simulation
- Makefile.questa – A *makefile* used for Questa simulation

Running the Unified Hierarchy e-SV Demo

Make sure the installation and setup steps listed in the section were successful before you run the demo.

To run the demo using Cadence tools, use the following command line:

```
% demo.sh IES
```

To run the demo with Synopsys tools and OSCI SystemC, use the following command line:

```
% demo.sh VCS
```

To run the demo with Mentor tools and OSCI SystemC, use the following command line:

```
% demo.sh QUESTA
```

Unified Hierarchy Demo Results

The demo executes in batch mode, and the output shows the order of construction of the ML environment and the propagation of phases from the UVM-***e*** environment into the UVM-SV and UVM-***e*** verification IP.

This demo also demonstrates the configuration capability by showing how the UVM-***e*** testbench component controls the construction of the both VIP. The testbench component sets the value for the address to be used in both producer components.

```
keep uvm_config_set("*producer", "address", 0x1000);
```

The following listing is a small example from the log file that is generated by the demo.

```
*** Starting non-blocking TLM2 transactions from e
[10] producer-@7: Calling nb_transport_fw WRITE 0xd0 0xe4 0x59 0x60
UVM_INFO @ 10: sys.testbench.sv_child.consumer [cons_type] SV consumer::nb_transport_fw
UVM_TLM_WRITE_COMMAND [0x0000000000001000] = d0 e4 59 60 (status=INCOMPLETE)
[10] producer-@7: return status TLM_ACCEPTED END_REQ
UVM_INFO @ 10: sys.testbench.sv_child.consumer [cons_type] SV consumer::respond
UVM_TLM_WRITE_COMMAND [0x0000000000001000] = d0 e4 59 60 (status=OK)
[10] producer-@7: Received nb_transport_bw TLM_OK_RESPONSE phase= BEGIN_RESP
[20] producer-@7: Calling nb_transport_fw READ 0x00 0x00 0x00 0x00
UVM_INFO @ 20: sys.testbench.sv_child.consumer [cons_type] SV consumer::nb_transport_fw
UVM_TLM_READ_COMMAND [0x0000000000001000] = 00 00 00 00 (status=INCOMPLETE)
[20] producer-@7: return status TLM_ACCEPTED END_REQ
[20] producer-@7: nb_transport READ returned 0xd0 0xe4 0x59 0x60
UVM_INFO @ 20: sys.testbench.sv_child.consumer [cons_type] SV consumer::respond
UVM_TLM_READ_COMMAND [0x0000000000001000] = d0 e4 59 60 (status=OK)
[20] producer-@7: Received nb_transport_bw TLM_OK_RESPONSE phase= BEGIN_RESP
...
```

The output shows the IP communicating by passing blocking and non-blocking transactions in both directions.

Additional Tests

Additional small examples are available under the `$UVM_ML_HOME/ml/tests` directory with *makefiles* to run each test. These tests check for proper behavior of specific features, such as ML TLM communication, and synchronized ML testbench construction.

Conclusion

This *Quick Start* provided you with the basic information you need to install, run, and view the results of the SC-SV demo and the SV-e demo that are supplied with the UVM Multi-Language package. For more information about UVM go to UVM World at: http://www.uvmworld.org

Trouble Shooting

Following are some identified issues users might hit, and the recommended solutions for them:

Test or demo fail with error: ncvlog: *E,NOPBIND

Problem: A test or demo fails with the following error:

```
      import uvm_ml::*;
                   |
      ncvlog: *E,NOPBIND (./test.sv,22|12): Package uvm_ml could not be bound.
```

Explanation: For UVM-ML one must use the ML ready UVM-SV library included in the release. This error message is received when the UVM_HOME environment variable is set to a different path which does not contain the ML ready version of UVM-SV.

Solution: Unset the UVM_HOME environment variable

```
        % unsetenv UVM_HOME
```

setup.sh fails with error message: sc_utils_ids.cpp:110: error: …

Problem: sourcing the script setup.sh (or compiling the SC library in any other way) fails with the following (or similar) errors:

```
     ... /sc_utils_ids.cpp:110: error: 'getenv' is not a member of 'std'
     ... /sc_utils_ids.cpp:111: error: 'strcmp' was not declared in this scope
```

Explanation: This is a known issue with the Accellera SystemC (open source) version systemc-2.2.0 when compiled with GNU C++ version above 4.3. See http://openesl.org/systemc-wiki/Installation for more information.

Solution: The suggested solution is to use the patch in the link above or manually modify sysc/utils/sc_utils_ids.cpp so that it will explicitly include the following files:

```
        #include "string.h"
        #include "cstdlib"
```

Errors during compilation

Problem: some of the necessary packages are not available or the path is not set properly

```
... error: tlm_h/tlm_version.h: No such file or directory
... error: sysc/datatypes/bit/sc_lv.h: No such file or directory
```

Solution: Verify that the environment variables are set properly and the packages are in the place pointed by the environment variable e.g.

```
setenv OSCI_INSTALL ... /2.2/g++-4.4.5-pic
setenv OSCI_SRC ... /systemc-2.2.0/src
setenv TLM2_INSTALL ... /TLM-2009-07-15/include/tlm
```

Elaboration error when running a demo or test

Problem: Elaboration using the wrong library or passing the wrong arguments to the linker

```
ncelab: *F,SCILDD: Could not load SystemC model library...
```

Solution: irun must pass the appropriate arguments to the linker as shown in irun_uvm_ml.64.f

```
irun -f $UVM_ML_HOME/ml/tests/irun_uvm_ml.64.f ...
```