Virtual Sequencers & Virtual Sequences

Clifford E. Cummings
Sunburst Design, Inc.
cliffc@sunburst-design.com

Janick Bergeron
Synopsys Inc
Janick.Bergeron@synopsys.com

World-class SystemVerilog & UVM Verification Training

Life is too short for bad or boring training!

UVM Virtual Sequencer Components

UVM Reference

top

Virtual sequences will execute AHB & ETH packets on their respective sequencers using the handles in the virtual sequencer

test1

Test runs virtual sequences on the virtual sequencer

env

e

v_sqr

vsequencer
tb_ahb_sequencer ahb_sqr
tb_eth_sequencer eth_sqr

uvm_config_db
"ahb_sqr" ahb_agnt.sqr
"eth_sqr" eth_agnt.sqr

ahb_agnt
tb_ahb_agent
vif
sqr
tb_ahb_sequencer
tb_ahb_driver
drv
vif

eth_agnt
tb_eth_agent
vif
sqr
tb_eth_sequencer
tb_eth_driver
drv
vif

AHB driver sends AHB packets

ETH driver sends ETH packets

ahb_if
eth_if
DUT

```
ahb_pkt:
  m_addr (32-bits)
  m_data (64-bits)
```

```
eth_pkt:
  m_src (48-bits)
  m_dst (48-bits)
```

Agenda

- Verification often requires the coordination of multiple complex threads of stimulus generation & sampling

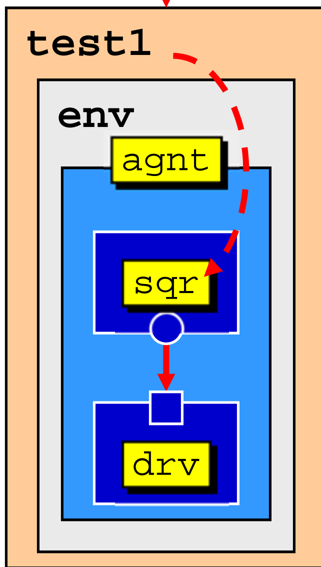
---

- Agenda:
  - UVM virtual sequencers
  - `m_sequencer`, `p_sequencer` & `` `uvm_declare_p_sequencer ``
  - UVM virtual sequences

The paper includes all the code for a full working example

When to Use a Virtual Sequencer

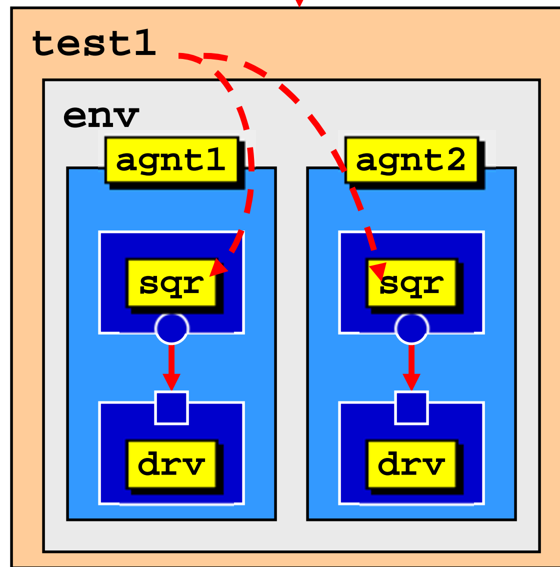
Single driving agent



NO virtual sequencer required

Multiple driving agents

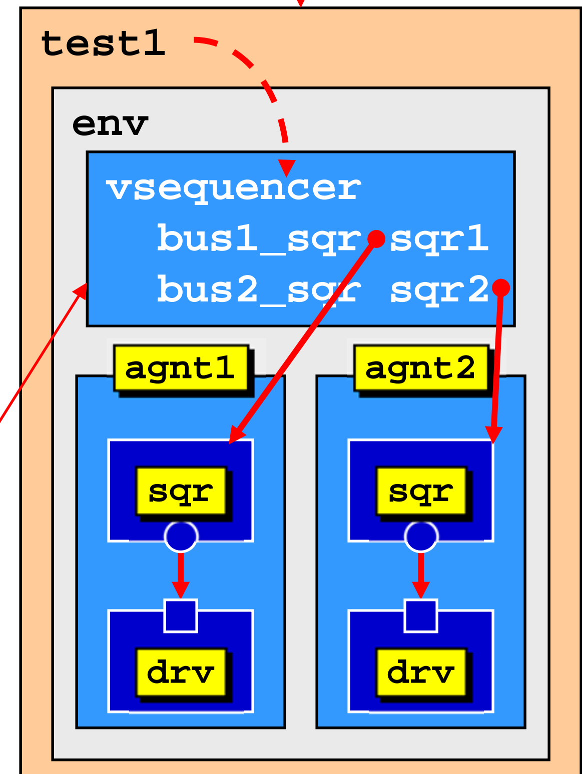
No stimulus coordination required



NO virtual sequencer required

Multiple driving agents

Stimulus coordination is required

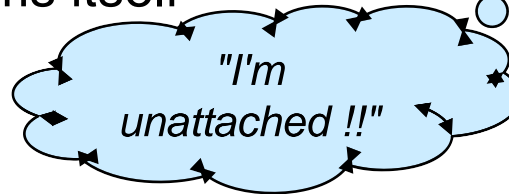


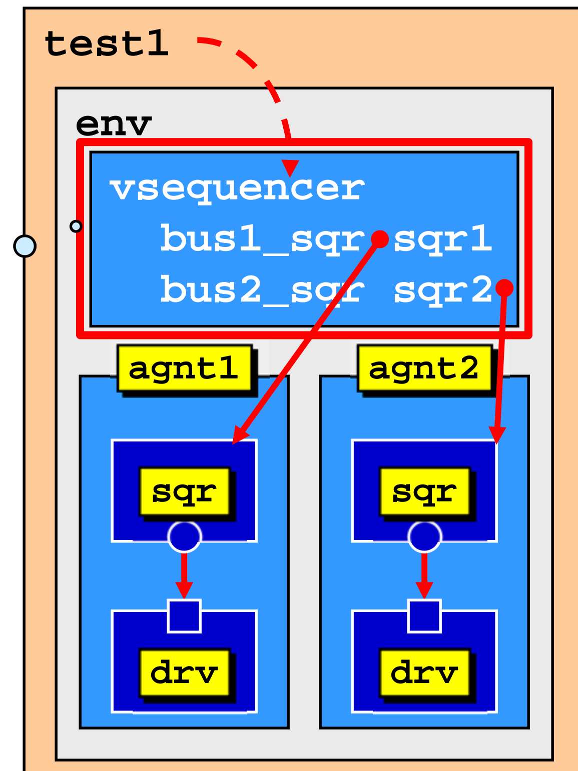
Virtual sequencer *REQUIRED!*

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Virtual Sequencer

- A virtual sequencer:
  - Controls other sequencers
  - Is not attached to a driver
  - Does not process items itself





- 3 virtual sequencer modes to interact with sub-sequencers

Most common mode covered in the paper / presentation

  - Parallel traffic generation

Also called "business as usual"

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION 2016 UNITED STATES

Virtual Sequencers & Sequences

Requirements Overview

Copies real sub-sequencer handles into `uvm_config_db`

- Env `sets` sub-sequencer handles into `uvm_config_db`
- Create a virtual sequencer
  - Declare and `get` sub-sequencer handles from `uvm_config_db`

Testbench structure

- Create a `vseq_base` class ← Virtual sequence base class
  - Use `` `uvm_declare_p_sequencer() `` macro to set `vsequencer` handle
  - Use the `body()` task to set the sub-sequencer handles
- Create virtual sequences by extending `vseq_base`
  - Start sub-sequences on sub-sequencers
  - Coordinate execution order of sub-sequences
- Test starts virtual sequences on the virtual sequencer

Sequences & tests

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

UVM Virtual Sequencer Construction

top

build_phase() is top-down

The agents will declare ahb_if & eth_if handles called vif

Build 1 → test1

e

Build 2 → env

v_sqr

Build 3a → vsequencer
tb_ahb_sequencer ahb_sqr
tb_eth_sequencer eth_sqr

Build 3b → ahb_agnt tb_ahb_agent

Build 3c → eth_agnt tb_eth_agent

Build 4c → vif

Build 4f → vif

Build 4a → sqr tb_ahb_sequencer

Build 4d → sqr tb_eth_sequencer

Build 4b → drv tb_ahb_driver vif

Build 4e → drv tb_eth_driver vif

get the ahb_if handle from uvm_config_db

get the eth_if handle from uvm_config_db

uvm_config_db
"ahb_vif" ahb_if
"eth_vif" eth_if

ahb_if

eth_if

DUT

Pre- run_test()
Set ahb_if & eth_if handles in uvm_config_db

UVM Virtual Sequencer Construction

UVM Virtual Sequencer Construction

UVM Virtual Sequence[...]

Source Code

Code shown in the presentation:

- env
- vsequencer
- vseq_base
- v_seq1 & v_seq2
- test_base
- test1 & test2
- ahb_pkt
- eth_pkt
- ahb_seq
- eth_seq

top

test1

e

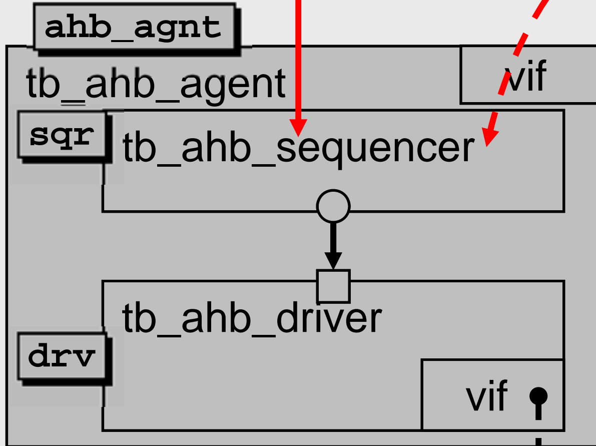
env

v_sqr

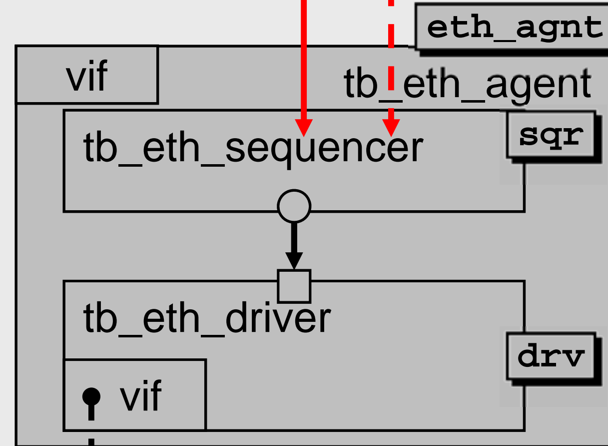
vsequencer
tb_ahb_sequencer ahb_sqr
tb_eth_sequencer eth_sqr

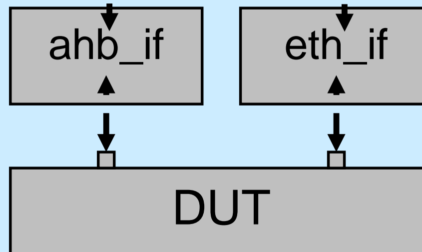
uvm_config_db
"ahb_sqr" ahb_agnt.sqr
"eth_sqr" eth_agnt.sqr

ahb_agnt



eth_agnt





Code shown in the paper:

- ahb_agent
- eth_agent
- ahb_if
- eth_if
- DUT

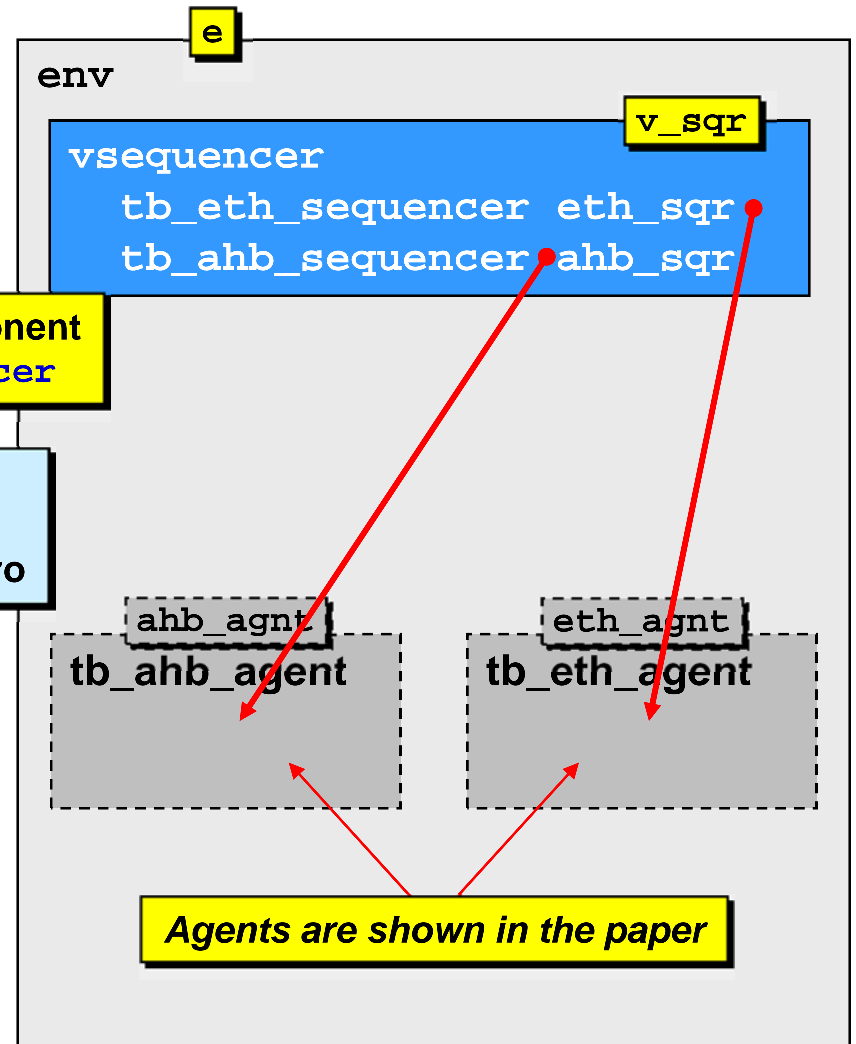
UVM Virtual Sequencer & Sequences

- On the next slides, we will build:
  - env
  - vsequencer
  - **

env for the vsequencer and agents

Virtual sequencer component built from uvm_sequencer

Info about the built-in m_sequencer & p_sequencer handles and the `uvm_declare_p_sequencer() macro



Virtual Sequencer Environment

Ethernet & AHB Agents + vsequencer

```
class env extends uvm_env;
  `uvm_component_utils(env)
  tb_eth_agent eth_agnt;
  tb_ahb_agent ahb_agnt;
  vsequencer   v_sqr;

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    eth_agnt = tb_eth_agent::type_id::create("eth_agnt", this);
    ahb_agnt = tb_ahb_agent::type_id::create("ahb_agnt", this);
    v_sqr    =   vsequencer::type_id::create("v_sqr"   , this);
  endfunction

  function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    uvm_config_db#(tb_ahb_sequencer)::set(this,"*","ahb_sqr",ahb_agnt.sqr);
    uvm_config_db#(tb_eth_sequencer)::set(this,"*","eth_sqr",eth_agnt.sqr);
  endfunction
endclass
```

Declare the two agents

Declare a virtual sequencer

Build the two agents and the virtual sequencer

Store the `ahb_agnt.sqr` handle in the `uvm_config_db`

Store the `eth_agnt.sqr` handle in the `uvm_config_db`

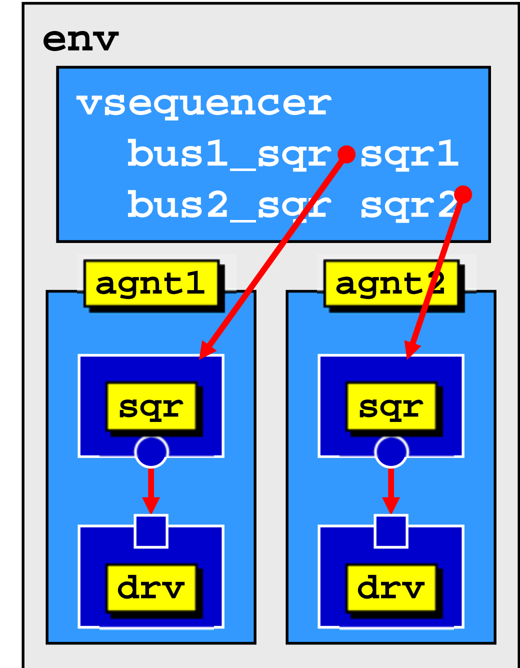
The `vseq_base` will `get` these subsequencer handles

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

Creating a Virtual Sequencer

- To create a virtual sequencer:
  - Derive a virtual sequencer from the `uvm_sequencer` class
  - Declare handles to the sub-sequencers
  - `get` the subsequencer handles from the `uvm_config_db`

The virtual sequences will coordinate the activity



Virtual Sequencer

vseqeuncer.sv

Not parameterized because the virtual sequencer executes different transaction types on different parameterized sub-sequencers

Virtual sequencer is *NOT* parameterized

```
class vsequencer extends uvm_sequencer;
  `uvm_component_utils(vsequencer)


  tb_ahb_sequencer ahb_sqr;
  tb_eth_sequencer eth_sqr;

  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  function void end_of_elaboration_phase(uvm_phase phase);
    super.end_of_elaboration_phase(phase);

    if (!uvm_config_db#(tb_ahb_sequencer)::get(this, "ahb_sqr", "", ahb_sqr))
        `uvm_fatal("VSQR/CFG/NOAHB", "No ahb_sqr specified for this instance");

    if (!uvm_config_db#(tb_eth_sequencer)::get(this, "eth_sqr", "", eth_sqr))
        `uvm_fatal("VSQR/CFG/NOETH", "No eth_sqr specified for this instance");

  endfunction
endclass
```

Declare handles of two real sequencer types

end_of_elaboration_phase()

Get and check the ahb_sqr and eth_sqr handles from the uvm_config_db

m_sequencer, p_sequencer & `uvm_declare_p_sequencer macro

Lots of confusion about these

m_sequencer & p_sequencer

`uvm_declare_p_sequencer macro

- m_sequencer
  - A handle inside each sequence that points to its controlling sequencer
  - Set automatically

Users should never directly access m_variables

Set by calling sequence.start(*path_to_sequencer*) from the test

- p_sequencer
  - A handle set by calling the `uvm_declare_p_sequencer macro
  - User-settable handle to access local sequencer variables from the test

Setting the p_sequencer handle can help set sub-sequencer handles

- `uvm_declare_p_sequencer macro:
  - Used to set the p_sequencer handle

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

`uvm_declare_p_sequencer()

Defined in macros/uvm_sequence_defines.svh File

```
class vseq_base extends uvm_sequence;
  `uvm_object_utils(vseq_base)
  `uvm_declare_p_sequencer(vsequencer)
  ...
```

This macro sets the p_sequencer handle to point to the vsequencer

```
`define uvm_declare_p_sequencer(SEQUENCER)   \
  SEQUENCER p_sequencer;                      \

  virtual function void m_set_p_sequencer();\
    super.m_set_p_sequencer();               \
    if( !$cast(p_sequencer, m_sequencer))    \
        `uvm_fatal("DCLPSQ",                 \
        $sformatf("%m %s Error casting p_sequencer, ... ", \
        get_full_name()))                    \
  endfunction
```

The macro also creates the m_set_p_sequencer() function

The user *never* calls this function

Cast and check m_sequencer to the p_sequencer handle

This function is executed by the seq.start() method

(The full string)

"%m %s Error casting p_sequencer, please verify that this sequence/sequence item is intended to execute on this type of sequencer"

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Virtual Sequences -vs- Regular Sequences

- A virtual sequence:
  - Is *not parameterized* to any transaction type
  - Executes sequences that have *different transaction types*
  - Executes sequences on the *sub-sequencer handles stored in the virtual sequencer*
  - Does not require any modifications to existing sequences
  - Can use `start_item()` / `randomize()` / `finish_item()`
  - Can use the `` `uvm_do `` macros:

    `` `uvm_do_on ``

    `` `uvm_do_on_with ``

A regular sequence

Is parameterized

Only executes sequences of one transaction type

Executes sequences on a specified sequencer

This is good !!

DO a sequence or sequence_item
ON the specified sequencer
WITH these inline constraints

UVM Basic Transaction Objects

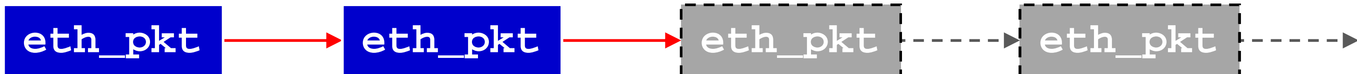
- On the next slides, we will build:
  - eth_pkt
  - ahb_pkt
  - eth_seq1
  - ahb_seq1

eth_pkt
src (48-bits)
dst (48-bits)

ahb_pkt
addr (32-bits)
data (64-bits)

Basic transaction types extended from uvm_sequence_item

Randomly generate sequence of 2-4 eth_pkt's



Randomly generate sequence of 2-5 ahb_pkt's

Sequences extended from uvm_sequence

Ethernet & AHB Packet Transaction Types

Extended from uvm_sequence_item

```
class eth_pkt extends uvm_sequence_item;
  `uvm_object_utils(eth_pkt)
  rand bit [47:0] src;
  rand bit [47:0] dst;
  ...
endclass
```

Declare two random fields

```
class ahb_pkt extends uvm_sequence_item;
  `uvm_object_utils(ahb_pkt)
  rand bit [31:0] addr;
  rand bit [63:0] data;
  ...
endclass
```

Declare two random fields

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION 2016 UNITED STATES

AHB Sequence

Dummy Sequence of AHB Items

```
class ahb_seq1 extends uvm_sequence #(ahb_pkt);
  `uvm_object_utils(ahb_seq1);


  rand int cnt;
  constraint c1 {cnt inside {[2:5]};}

  function new(string name = "ahb_seq1");
    super.new(name);
  endfunction

  virtual task body();
    ahb_pkt ahb_pkt1;
    `uvm_info("AHBcnt", $sformatf("** Loop cnt=%0d **", cnt), UVM_MEDIUM)
    repeat(cnt) `uvm_do(ahb_pkt1)
  endtask


endclass
```

Sequence of `ahb_pkt` sequence items

Declare and constrain a random loop `cnt`

Declare an `ahb_pkt` sequence item type

Print a `Loop cnt` message

Execute the `ahb_pkt1` item 2:5 times

`` `uvm_do `` command

DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION 2016 UNITED STATES

Ethernet Sequence

Dummy Sequence of Ethernet Items

```
class eth_seq1 extends uvm_sequence #(eth_pkt);
  `uvm_object_utils(eth_seq1);


  rand int cnt;
  constraint c1 {cnt inside {[2:4]};}

  function new(string name = "eth_seq1");
    super.new(name);
  endfunction

  virtual task body();
    eth_pkt eth_pkt1 = eth_pkt::type_id::create("eth_pkt1");
    `uvm_info("ETHcnt", $sformatf("** Loop cnt=%0d **", cnt), UVM_MEDIUM)
    repeat(cnt) begin
      start_item(eth_pkt1);
      if(!eth_pkt1.randomize()) `uvm_error("RAND", "FAILED")
      finish_item(eth_pkt1);
    end
  endtask

endclass
```

Sequence of eth_pkt sequence items

Declare and constrain a random loop cnt

Declare and create an eth_pkt sequence item type

Print a Loop cnt message

Execute the eth_pkt1 item 2-4 times

start_item() / randomize() / finish_item() commands

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

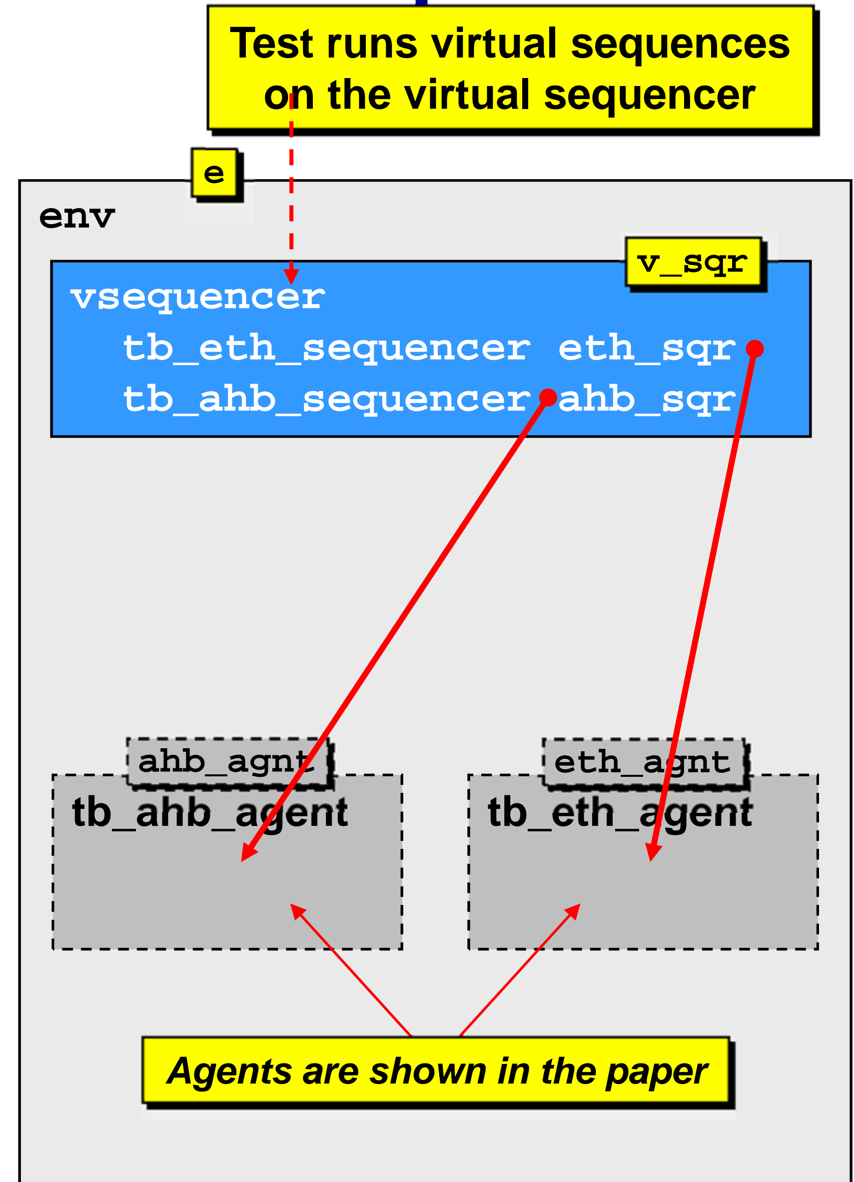
UVM Virtual Sequencer & Sequences

- On the next slides, we will build:
  - `vseq_base`

    Virtual sequence base built from `uvm_sequence`

  - `v_seq1`, `v_seq2`

    Virtual sequences extended from the virtual sequence base (`vseq_base`)



Virtual Sequence Base

- Recommendations:
  - Use a virtual-sequence-base class to copy the real sequencer handles to the virtual sequencer handles
  - Other virtual sequences should extend the virtual-sequence-base
  - Other virtual sequences will call the `super.body()` to get the connections

The virtual-sequence-base does the hard work!

Virtual Sequence Base

Re-Usable - Recommended

```
class vseq_base extends uvm_sequence;
  `uvm_object_utils(vseq_base)
  `uvm_declare_p_sequencer(vsequencer)

  function new(string name="vseq_base");
    super.new(name);
  endfunction



  tb_ahb_sequencer ahb_sqr;
  tb_eth_sequencer eth_sqr;


  virtual task body();
    ahb_sqr = p_sequencer.ahb_sqr;
    eth_sqr = p_sequencer.eth_sqr;
  endtask
endclass
```

This macro sets the `p_sequencer` handle to point to the `vsequencer`

Declare the two sub-sequencer handles

This sequence will assign these handles to point to the handles from the virtual sequencer *(see the body() task)*

Make the sub-sequencer handles in this base sequence point to the two sequencer handles in the `p_sequencer` (`vsequencer`)

The sub-sequencer handles are declared and assigned in this class *(re-usable!)*

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

Virtual Sequence #1

Sequentially Executes Sequences on Different Sequencers

```
class v_seq1 extends vseq_base;
  `uvm_object_utils(v_seq1)

  function new(string name="v_seq1");
    super.new(name);
  endfunction


  virtual task body();
    ahb_seq1 ahb_seq;
    eth_seq1 eth_seq;
    //-----------------------------------
    super.body();

    `uvm_info("v_seq1", "Executing sequence", UVM_HIGH)

    `uvm_do_on(ahb_seq, ahb_sqr)
    `uvm_do_on(eth_seq, eth_sqr)
    `uvm_do_on(eth_seq, eth_sqr)
    `uvm_do_on(ahb_seq, ahb_sqr)

    `uvm_info("v_seq1", "Sequence complete", UVM_HIGH)
  endtask
endclass
```

v_seq1 extends from vseq_base

vseq_base extends from uvm_sequence

Inherit the handles:
tb_ahb_sequencer ahb_sqr;
tb_eth_sequencer eth_sqr

Declare the two sequence types

body() task in the vseq_base did most of the hard work and is called by this virtual sequence

Sequentially execute the sequences on the two sequencers (ahb_sqr & eth_sqr) that were declared in the vseq_base class

This virtual sequence uses the `uvm_do_on() macros

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Virtual Sequence #2

Sequentially Executes Sequences on Different Sequencers

```
class v_seq2 extends vseq_base;
  `uvm_object_utils(v_seq2)

  function new(string name="v_seq2");
    super.new(name);
  endfunction

  virtual task body();
    ahb_seq1 ahb_pkts = ahb_seq1::type_id::create("ahb_pkts");
    eth_seq1 eth_pkts = eth_seq1::type_id::create("eth_pkts");
    //----------------------------------
    super.body();
    `uvm_info("v_seq2", "Executing sequenc
    if(!ahb_pkts.randomize()) `uvm_error("
    ahb_pkts.start(ahb_sqr);
    if(!eth_pkts.randomize()) `uvm_error("RAND","FAILED");
    eth_pkts.start(eth_sqr);
    if(!eth_pkts.randomize()) `uvm_error("RAND","FAILED");
    eth_pkts.start(eth_sqr);
    if(!ahb_pkts.randomize()) `uvm_error("RAND","FAILED");
    ahb_pkts.start(ahb_sqr);
    `uvm_info("v_seq2", "Sequence complete", UVM_MEDIUM)
  endtask
endclass
```

v_seq2 extends from vseq_base

vseq_base extends from uvm_sequence

Inherit the handles:
tb_ahb_sequencer ahb_sqr;
tb_eth_sequencer eth_sqr

Declare and create two sequence types

body() task in the vseq_base did most of the hard work and is called by this virtual sequence

This virtual sequence uses the randomize()/start() methods

Sequentially start() sequences on ahb_sqr & eth_sqr

UVM Tests

Using Virtual Sequencer & Sequences

- On the next slides, we will build:
  - `test_base`

    Common base test code

  - `test1`, `test2`

    Two tests that execute virtual sequences on a virtual sequencer - built from a common base-test

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Common Test Base

```
class test_base extends uvm_test;
  `uvm_component_utils(test_base)

  env e;

  function new (string name, uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info("test_base build", "Starting test build", UVM_FULL)
    e = env::type_id::create("e", this);
  endfunction

  function void start_of_simulation_phase(uvm_phase phase);
    super.start_of_simulation_phase(phase);
    if (uvm_report_enabled(UVM_HIGH)) begin
      this.print();
      factory.print();
    end
  endfunction
endclass
```

Create a common base-test class

All tests will use this common env environment

Create the env

Pre run() phase

If sim is run with +UVM_VERBOSITY=UVM_HIGH (or higher) print the test-components and factory contents

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Test1 Running Virtual Sequence

test1 extends the base-test

```
class test1 extends test_base;
  `uvm_component_utils(test1)


  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction


  task run_phase(uvm_phase phase);
    v_seq1 vseq = v_seq1::type_id::create("vseq");
    phase.raise_objection(this);
    `uvm_info("test1 run", "Starting test", UVM_MEDIUM)

    vseq.start(e.v_sqr);

    `uvm_info("test1 run", "Ending test", UVM_MEDIUM)
    phase.drop_objection(this);
  endtask
endclass
```

Inherit the environment and env-build phase

Declare & create a v_seq1 virtual sequence type (vseq)

Start the virtual v_seq1 (vseq) on the virtual sequencer (e.v_sqr)

Starting the sequence on a sequencer will set m_sequencer to point to the selected sequencer

Test1 Simulation Output

The virtual sequencer executed:
2-5 `ahb_pkt` / 2-4 `eth_pkt` / 2-4 `eth_pkt` / 2-5 `ahb_pkt`

```
test1.sv(24) @ 0: uvm_test_top [test1 run] Starting test
ahb_seq1.sv(23) @ 0: ...    [AHBcnt] ** Loop cnt=3 **
dut.sv(21) @ 134: [DUT AHB] ahb_addr=06361db6       ahb_data=3ddd4f4ffbf06a7e
dut.sv(21) @ 268: [DUT AHB] ahb_addr=ec0b0d27       ahb_data=82f67462a4354abe
dut.sv(21) @ 402: [DUT AHB] ahb_addr=20235580       ahb_data=340d4cf2f17d2bdf
eth_seq1.sv(23) @ 402: ...  [ETHcnt] ** Loop cnt=2 **
dut.sv(25) @ 492: [DUT ETH] eth_src =84df747d2537   eth_dst =5654ddb2c570
dut.sv(25) @ 582: [DUT ETH] eth_src =a3c5e0fd1a73   eth_dst =e0c6db5e392f
eth_seq1.sv(23) @ 582: ...  [ETHcnt] ** Loop cnt=2 **
dut.sv(25) @ 672: [DUT ETH] eth_src =1ffe6da7050e   eth_dst =d2d6de397e3c
dut.sv(25) @ 762: [DUT ETH] eth_src =92141cfb4593   eth_dst =bface63b02da
ahb_seq1.sv(23) @ 762: ...  [AHBcnt] ** Loop cnt=4 **
dut.sv(21) @ 896: [DUT AHB] ahb_addr=6fb4982b       ahb_data=25016b70482d259f
dut.sv(21) @ 1030: [DUT AHB] ahb_addr=e766b6bd       ahb_data=14171be78470dba9
dut.sv(21) @ 1164: [DUT AHB] ahb_addr=9975f3a6       ahb_data=93c8dda3f8b9de1e
dut.sv(21) @ 1298: [DUT AHB] ahb_addr=eaaa2743       ahb_data=d887a80c21e85c81
test1.sv(26) @ 1298: uvm_test_top [test1 run] Ending test
verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 1298: [TEST_DONE] 'run'
                              phase is ready to proceed to the 'extract' phase
```

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2016

Test2 Running Virtual Sequence

```
class test2 extends test_base;
  `uvm_component_utils(test2)


  function new(string name, uvm_component parent);
    super.new(name, parent);
  endfunction



  task run_phase(uvm_phase phase);
    v_seq2 vseq = v_seq2::type_id::create("vseq");
    phase.raise_objection(this);
    `uvm_info("test2 run", "Starting test", UVM_MEDIUM)

    vseq.start(e.v_sqr);

    `uvm_info("test2 run", "Ending test", UVM_MEDIUM)
    phase.drop_objection(this);
  endtask
endclass
```

test1 extends the base-test

Inherit the environment and env-build phase

Declare & create a v_seq2 virtual sequence type (vseq)

Start the virtual v_seq2 (vseq) on the virtual sequencer (e.v_sqr)

Starting the sequence on a sequencer will set m_sequencer to point to the selected sequencer

Test2 Simulation Output

The virtual sequencer executed:
2-5 ahb_pkt / 2-4 eth_pkt / 2-4 eth_pkt / 2-5 ahb_pkt

```
test2.sv(24) @ 0: uvm_test_top [test2 run] Starting test
ahb_seq1.sv(23) @ 0: ...    [AHBcnt] ** Loop cnt=2 **
dut.sv(21) @ 134: [DUT AHB] ahb_addr=ca06d8be       ahb_data=ee09b806a0c605dc
dut.sv(21) @ 268: [DUT AHB] ahb_addr=5a8ef50b       ahb_data=737b1e8b4a149b05
eth_seq1.sv(23) @ 268: ...  [ETHcnt] ** Loop cnt=4 **
dut.sv(25) @ 358: [DUT ETH] eth_src =ca2d05640a29   eth_dst =000aa70c3542
dut.sv(25) @ 448: [DUT ETH] eth_src =a361d437ec28   eth_dst =b30354a9da32
dut.sv(25) @ 538: [DUT ETH] eth_src =fc75b9433c74   eth_dst =b50709212801
dut.sv(25) @ 628: [DUT ETH] eth_src =f29e0b9fa957   eth_dst =b5a8d97f54a0
eth_seq1.sv(23) @ 628: ...  [ETHcnt] ** Loop cnt=4 **
dut.sv(25) @ 718: [DUT ETH] eth_src =72451ba1363d   eth_dst =1f566cad9f2f
dut.sv(25) @ 808: [DUT ETH] eth_src =d2680002084f   eth_dst =305749f95e99
dut.sv(25) @ 898: [DUT ETH] eth_src =ccb01d16a602   eth_dst =40bbf305dfb2
dut.sv(25) @ 988: [DUT ETH] eth_src =324aeddab7c0   eth_dst =d434ed8b4764
ahb_seq1.sv(23) @ 988: ...   [AHBcnt] ** Loop cnt=3 **
dut.sv(21) @ 1122: [DUT AHB] ahb_addr=d1f6c5b0       ahb_data=f09749776c1aea52
dut.sv(21) @ 1256: [DUT AHB] ahb_addr=29ae06db       ahb_data=3f52b23047269047
dut.sv(21) @ 1390: [DUT AHB] ahb_addr=fe3b2df1       ahb_data=7beb4517215e573f
test2.sv(26) @ 1390: uvm_test_top [test2 run] Ending test
verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1268) @ 1390: [TEST_DONE] 'run'
                               phase is ready to proceed to the 'extract' phase
```

2016 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Summary

- Virtual sequencers enable coordinated stimulus control

Have handles to underlying sequencers for use by virtual sequences

Macro **Handle**

- `` `uvm_declare_p_sequencer `` & `p_sequencer`

  - **Declare a common `p_sequencer` handle of the sequencer type**
  - **Casts the `m_sequencer` handle to the `p_sequencer`**
  - **Checks that the cast was successful**

- Virtual sequences coordinate execution of other sequences across multiple interfaces

Can use existing libraries of sequences without modification

Acknowledgements

- JL Gray — Helped put together the first version of the Virtual Sequence / Sequencer example shown in this paper
- Mark Litterick — Helped refine the use of the `p_sequencer` handle and `` `uvm_declare_p_sequencer `` macro
- Heath Chambers — Helped correct these materials for use in UVM training
- Multiple Verilab engineers — Responded to survey on preferred virtual sequencer usage modes
- Logie Ramachandran — Provided valuable feedback and suggestions for this paper

Virtual Sequencers & Virtual Sequences

Clifford E. Cummings
Sunburst Design, Inc.
cliffc@sunburst-design.com

Janick Bergeron
Synopsys Inc
Janick.Bergeron@synopsys.com

World-class SystemVerilog & UVM Verification Training

Life is too short for bad or boring training!