Simple testbench integration

Creating Meaningful tests

- Randomization for test patterns
- User controls test creation by:-
  - Adding constraints to control individual data items (basic functionality)
  - Using UVM sequences to control the order of multiple data items (more flexibility & control)

Constraining Data Items

a) Identify the data items and their generated fields in the verification component.
b) Create a derivation of the data item class that adds or overrides default constraints.
c) In a test, adjust the environment to use the newly-defined data items.
d) Run the simulation using a command line option to specify the test name.

Example

```
typedef enum bit {BAD_PARITY, GOOD_PARITY} parity_e;
class uart_frame extends uvm_sequence_item;
  rand int unsigned transmit_delay;
  rand bit start_bit;
  rand bit [7:0] payload;
  rand bit [1:0] stop_bits;
  rand bit [3:0] error_bits;
  bit parity;
  // Control fields
  rand parity_e parity_type;

  function new(input string name);
    super.new(name);
  endfunction

// Optional field declarations and automation flags
`uvm_object_utils_begin(uart_frame)
`uvm_field_int(start_bit, UVM_ALL_ON)
`uvm_field_int(payload, UVM_ALL_ON)
`uvm_field_int(parity, UVM_ALL_ON)
`uvm_field_enum(parity_e, parity_type, UVM_ALL_ON + UVM_NOCOMPARE)
`uvm_field_int(xmit_delay, UVM_ALL_ON + UVM_DEC + UVM_NOCOMPARE)
`uvm_object_utils_end

// Specification section 1.2: the error bits value should be
// different than zero.
constraint error_bits_c {error_bits != 4'h0;}

// Default distribution constraints
constraint default_parity_type {parity_type dist {
GOOD_PARITY:=90, BAD_PARITY:=10};}
// Utility functions
extern function bit calc_parity ( );
...
endfunction
endclass: uart_frame
```

Creating a Test-Specific Frame

```
class short_delay_frame extends uart_frame;

constraint test1_txmit_delay {transmit_delay < 10;}
`uvm_object_utils(short_delay_frame)

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

endclass: short_delay_frame
```

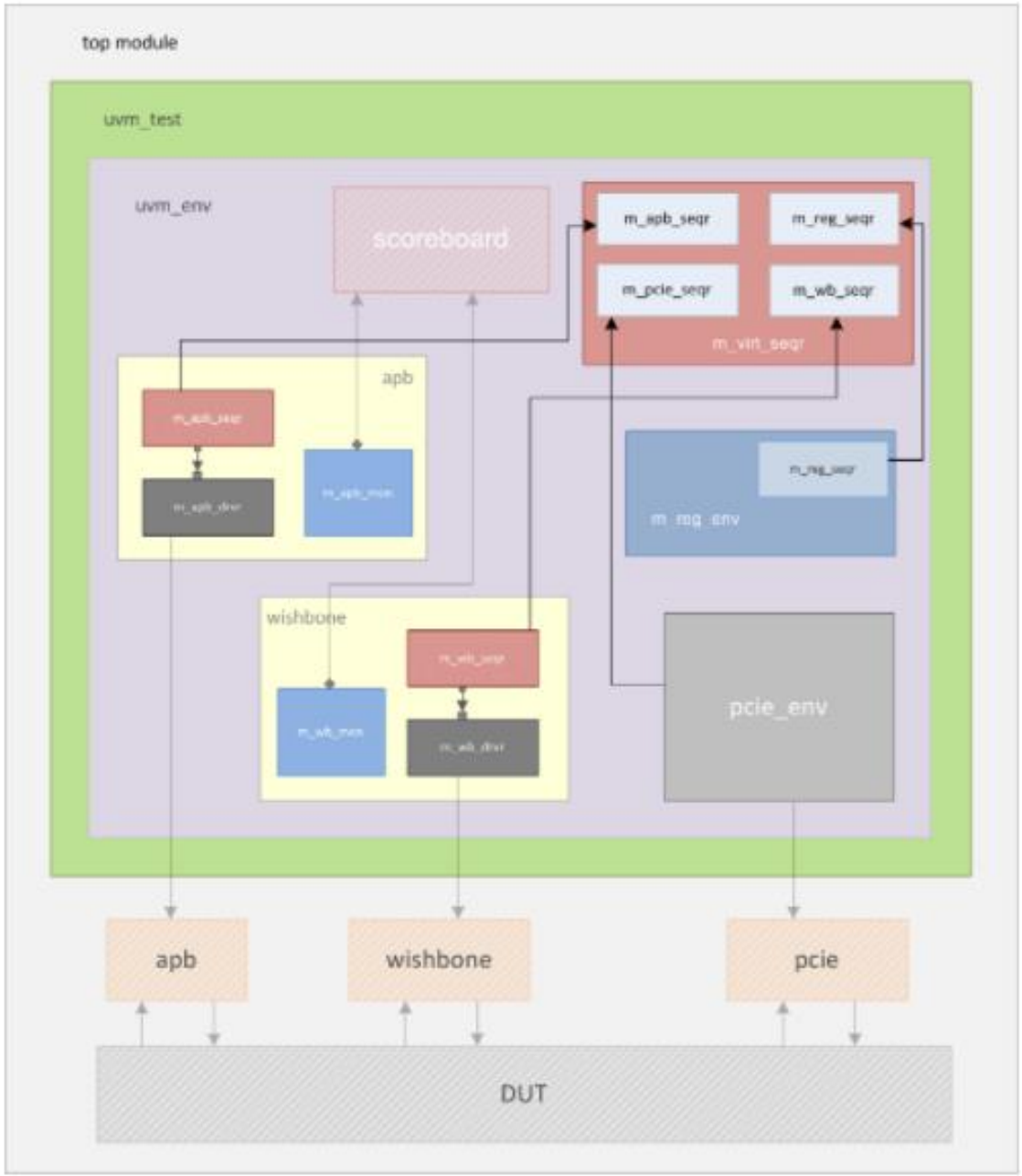
```
class short_delay_test extends uvm_test;
`uvm_component_utils(short_delay_test)
uart_tb uart_tb0;

  function new (string name = "short_delay_test",uvm_component parent = null);
    super.new(name, parent);
  endfunction

  virtual function build_phase(uvm_phase phase);
    super.build_phase(phase);
    // Use short_delay_frame throughout the environment.
    factory.set_type_override_by_type(uart_frame::get_type(),
    short_delay_frame::get_type());
    uart_tb0 = uart_tb::type_id::create("uart_tb0", this);
  endfunction

  task run_phase(uvm_phase phase);
    uvm_top.print_topology();
  endtask

endclass
```

Creating a Test-Specific Frame

```
set_inst_override_by_type("uart_env0.master.sequencer.*",
                          uart_frame::get_type(), short_delay_frame::get_type());
```

```
set_inst_override_by_type("uart_env*.master.sequencer.*",
                          uart_frame::get_type(), short_delay_frame::get_type());
```

UVM Virtual Sequencer



Example

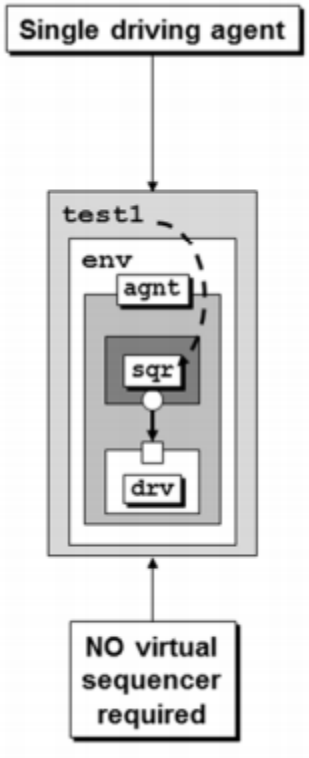
```
class my_virtual_sequencer extends uvm_sequencer;
    `uvm_component_utils (my_virtual_sequencer)

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

    // Declare handles to other sequencers here
    apb_sequencer       m_apb_seqr;
    reg_sequencer       m_reg_seqr;
    wb_sequencer        m_wb_seqr;
    pcie_sequencer      m_pcie_seqr;
endclass
```

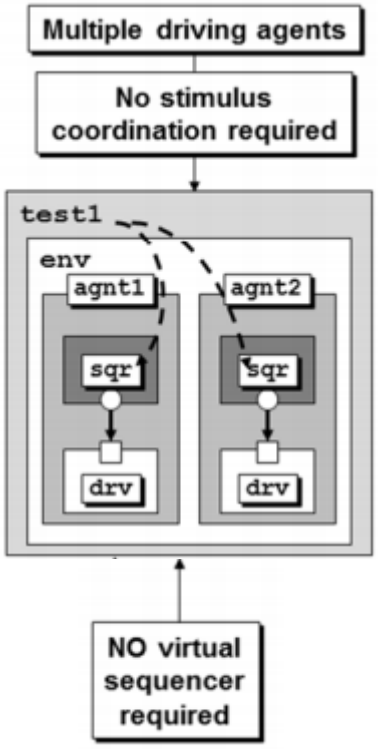
```
class top_env extends uvm_env;
    ...

    my_virtual_sequencer    m_virt_seqr;

    virtual function void build_phase (uvm_phase phase);
        ...
        m_virt_seqr = my_virtual_sequencer::type_id::create ("m_virt_seqr", this);
        ...
    endfunction

    // Connect virtual sequencer handles to actual sequencers
    virtual function void connect_phase (uvm_phase phase);
        ...
        m_virt_seqr.m_apb_seqr   = m_apb_agent.m_apb_seqr;
        m_virt_seqr.m_reg_seqr   = m_reg_env.m_reg_seqr;
        m_virt_seqr.m_pcie_seqr  = m_pcie_env.m_pcie_agent.m_pcie_seqr;
        ...
    endfunction
endclass
```

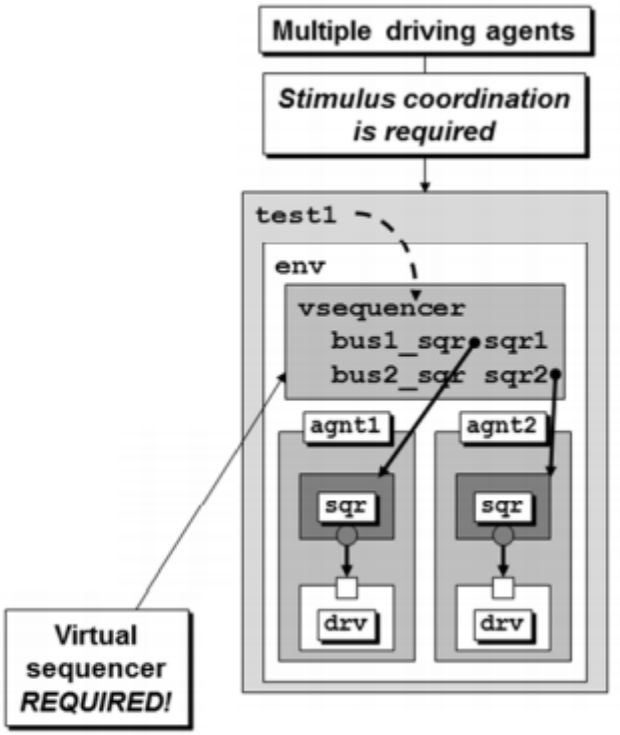
```
UVM_INFO @ 0: reporter [RNTST] Running test base_test...
UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
--------------------------------------------------------------
Name                          Type                        Size  Value
--------------------------------------------------------------
uvm_test_top                  ba
  m_top_env                   to
    m_apb_agent               ap
      m_apb_drv               ap
        rsp_port              uv
        seq_item_port         uv
      m_apb_mon               ap
      m_apb_seqr              uv
        rsp_export            uv
        seq_item_export       uv
        arbitration_queue     ar
        lock_queue            ar
        num_last_reqs         in
        num_last_rsps         in
    m_spi_agent               sp
      m_spi_drv               spi_driver                  -      @4897
        rsp_port              uvm_analysis_port           -      @5043
        seq_item_port         uvm_seq_item_pull_port      -      @4995
      m_spi_mon               spi_monitor                 -      @5024
      m_spi_seqr              uvm_sequencer               -      @4321
        rsp_export            uvm_analysis_export         -      @4377
        seq_item_export       uvm_seq_item_pull_imp       -      @4917
        arbitration_queue     array                       0      -
        lock_queue            array                       0      -
        num_last_reqs         integral                    32     'd1
        num_last_rsps         integral                    32     'd1
    m_virt_seq                virtual_sequencer           -      @2870
      rsp_export              uvm_analysis_export         -      @2928
      seq_item_export         uvm_seq_item_pull_imp       -      @3478
      arbitration_queue       array                       0      -
      lock_queue              array                       0      -
      num_last_reqs           integral                    32     'd1
      num_last_rsps           integral                    32     'd1
    m_wb_agent                wb_agent                    -      @2810
      m_wb_drv                wb_driver                   -      @5712
        rsp_port              uvm_analysis_port           -      @5858
        seq_item_port         uvm_seq_item_pull_port      -      @5810
      m_wb_mon                wb_monitor                  -      @5839
      m_wb_seqr               uvm_sequencer               -      @5136
        rsp_export            uvm_analysis_export         -      @5192
        seq_item_export       uvm_seq_item_pull_imp       -      @5732
        arbitration_queue     array                       0      -
        lock_queue            array                       0      -
        num_last_reqs         integral                    32     'd1
        num_last_rsps         integral                    32     'd1
--------------------------------------------------------------
```

```
UVM_INFO ./tb/my_pkg.sv(68) @ 0: uvm_test_top.m_top_env.m_virt_seq@@m_virt_seq [VSEQ] Start of virtu
al sequence
UVM_INFO ./tb/wb_agent.sv(63) @ 0: uvm_test_top.m_top_env.m_wb_agent.m_wb_seqr@@m_wb_reset_seq [RESE
T_SEQ] Starting wb_reset_seq
UVM_INFO ./tb/apb_agent.sv(52) @ 20000: uvm_test_top.m_top_env.m_apb_agent.m_apb_seqr@@m_apb_rw_seq
[RW_SEQ] Starting apb_rw_seq
UVM_INFO ./tb/spi_agent.sv(74) @ 30000: uvm_test_top.m_top_env.m_spi_agent.m_spi_seqr@@m_spi_tx_seq
[tx_SEQ] Starting spi_tx_seq
UVM_INFO ./tb/my_pkg.sv(75) @ 30000: uvm_test_top.m_top_env.m_virt_seq@@m_virt_seq [VSEQ] End of vir
tual sequence
UVM_INFO ./tb/my_pkg.sv(108) @ 30000: uvm_test_top [SHUT] Shutting down test ...

--- UVM Report catcher Summary ---
```

When do we need a virtual sequencer?







Why “virtual” sequencer/sequence

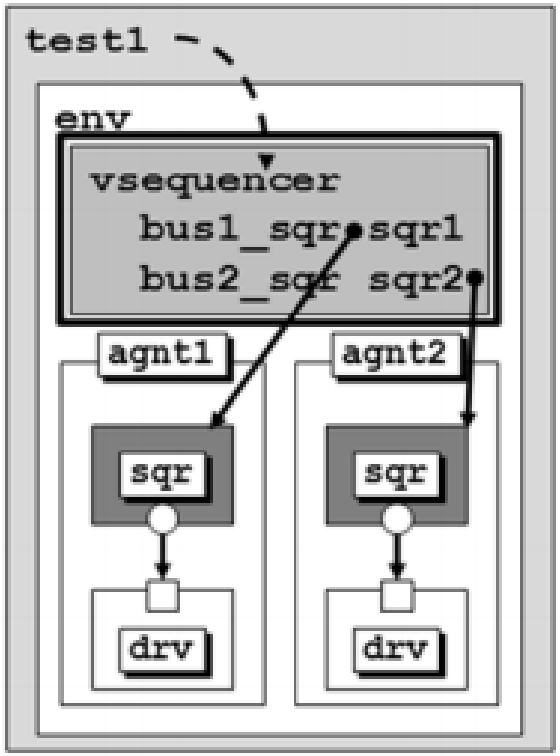
- Unlike SV virtual classes, virtual methods, and virtual interfaces, “virtual” keyword not required for virtual sequencers or virtual sequences.

- There is no uvm_virtual_sequencer or uvm_virtual_sequence base class.

So, why “virtual”?

3 attributes of a virtual sequencer:-

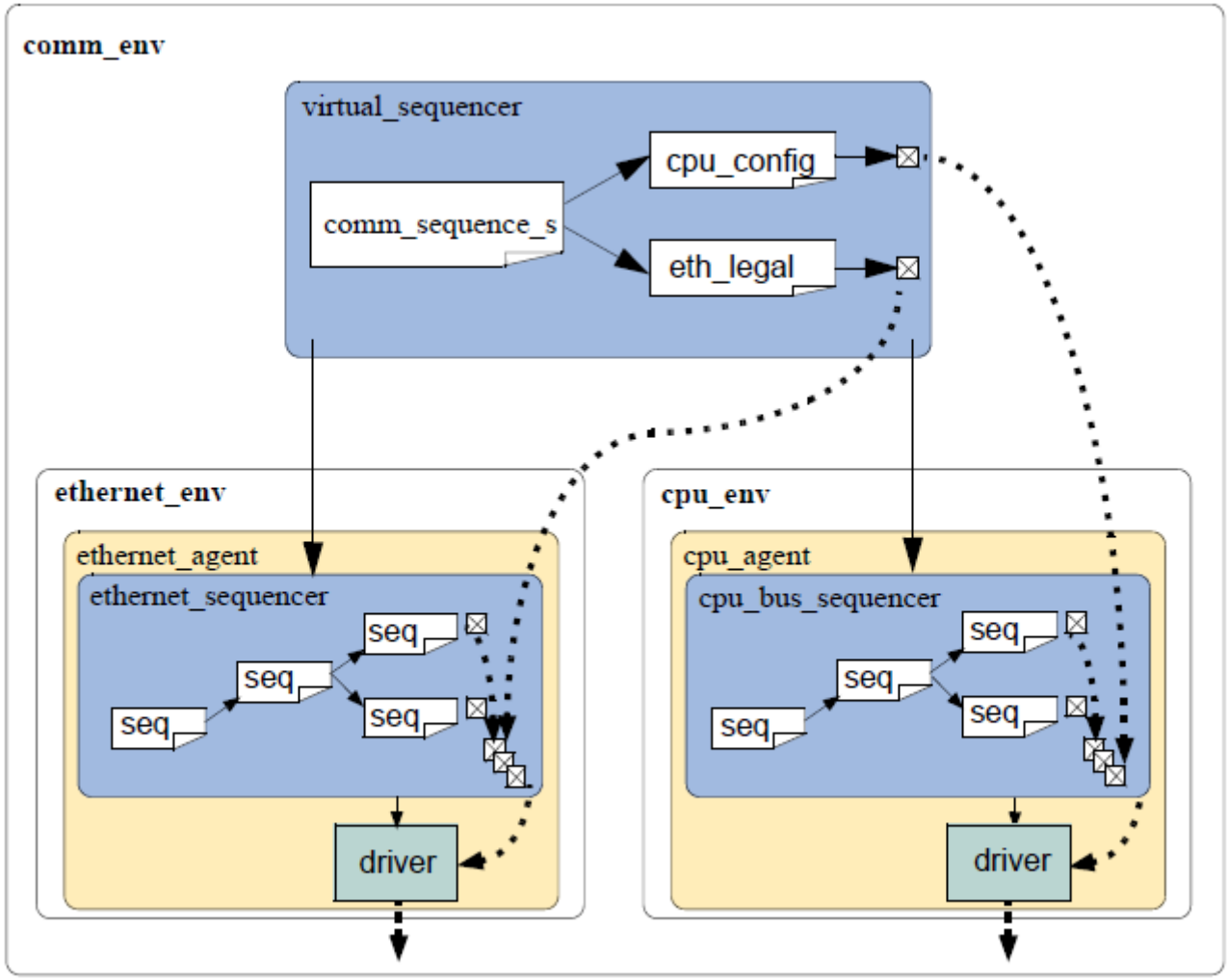
- Controls other sequencers
- Not attached to a driver
- Does not process items itself
  - Uses subsequencers



Three virtual sequencer modes

3 ways in which virtual sequences interact with sub-sequencers:-

- “Business as usual” – Virtual sub-sequencers and sub-sequencers send transactions simultaneously
- Disable sub-sequencers – Only virtual sequencer is driving
- Using grab() and ungrab() – Virtual sequencer takes control of the underlying driver(s) for a limited time


comm_env
virtual_sequencer
cpu_config
comm_sequence_s
eth_legal
ethernet_env
ethernet_agent
ethernet_sequencer
seq
seq
seq
seq
driver
cpu_env
cpu_agent
cpu_bus_sequencer
seq
seq
seq
seq
driver

Example

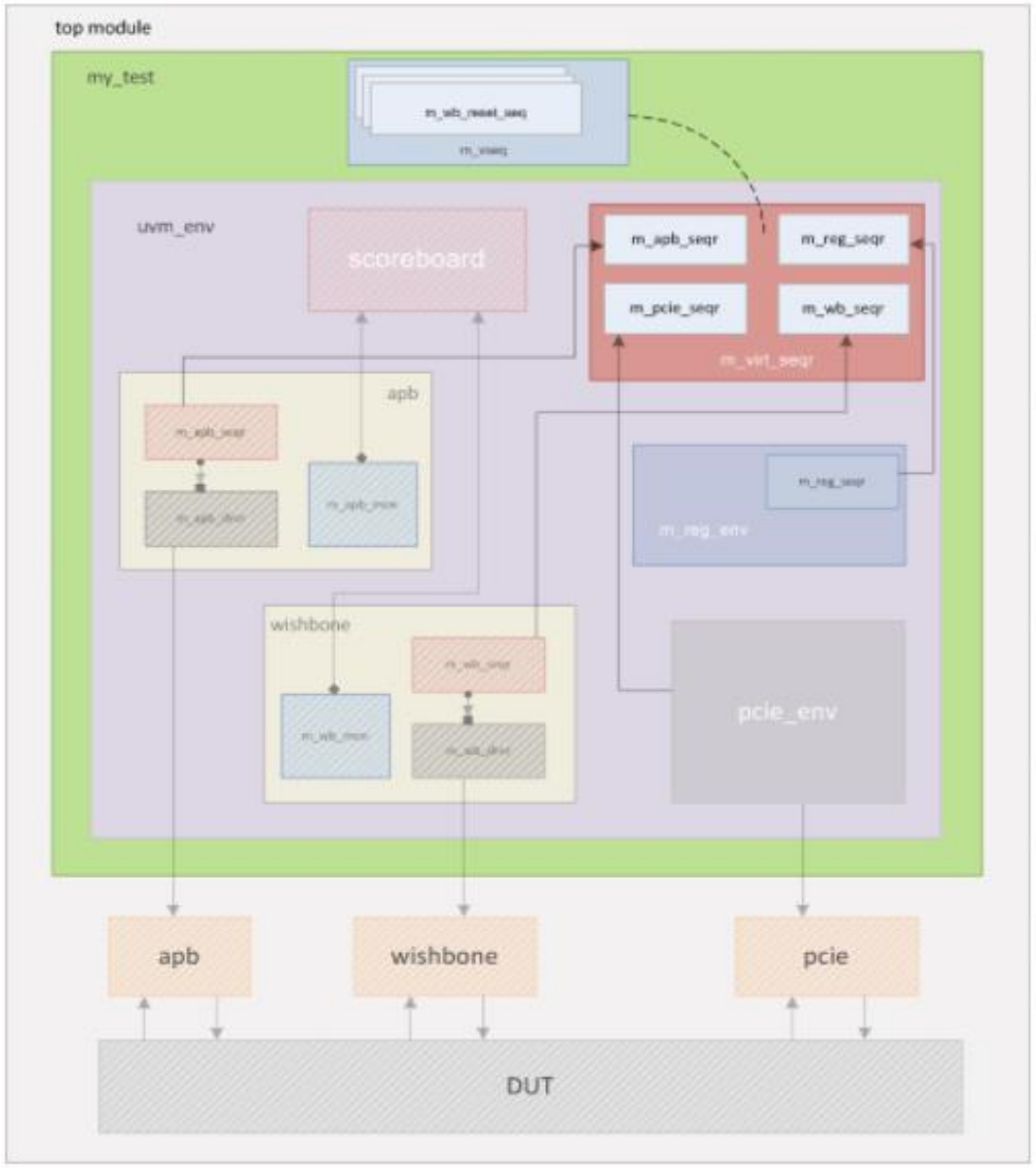
```
class vsequencer extends uvm_sequencer;
  `uvm_component_utils(vsequencer)
  tb_ahb_sequencer ahb_sqr;
  tb_eth_sequencer eth_sqr;

  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
```

Simplified Example

```
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
endclass
```

Virtual Sequences

- A container to start multiple sequences on different sequences in the environment.



2 methods to execute sequences

- `uvm_do - easiest to use, but less simulation efficient

- randomize() with start()

Checking for DUT correctness

- Assertions – Derived from the specification or from the implementation and ensure correct timing behavior.
- Assertions typically focus on signal-level activity.
- Data-checkers – Ensure overall device correctness.

Implementing a coverage model

1. Selecting a coverage method

2. Implementing a Functional Coverage Model

3. Enabling and Disabling Coverage

Selecting a Coverage Model

Explicit Coverage

- User-defined coverage
- User specifies coverage goals, needed values, and collection time
- Eg: Functional coverage
- Disadvantage: missing goals are not taken into account

Selecting a Coverage Model

Implicit Coverage

- Done with automatic metrics that are driven from RTL or other metrics already existing in code
- Eg: code coverage, expression coverage, FSM coverage
- Disadvantage: difficult to map the coverage requirements to the verification goals

Implementing a Functional Coverage Model

- Protocol-specific functional coverage model
- Disable some coverage unimportant aspects or don’t need to be verified
- Extend the functional-coverage model and create associations between verification component coverage and other system attributes