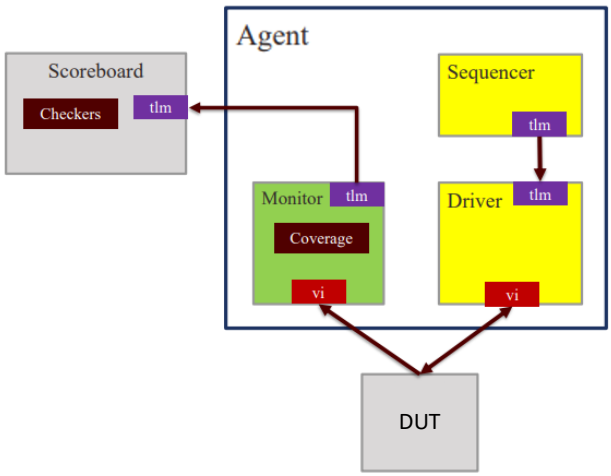
UVM Agent & Scoreboard

Prof Michael Quinn
From Slides by Dr. Niels
Burkhardt & Sheena Goel

CSCE 616 Fall 2018

Agent (Driver/Monitor) and Scoreboard

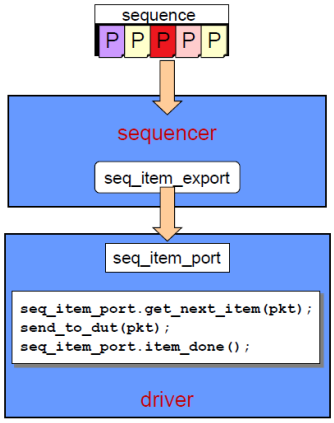


2

From Week 6: Sequencer Driver Operation

Sequencer is linked to a sequence.
- ❑Sequence is a pattern of individual data items

Sequencer Driver Operation:
- ❑Driver calls get_next_item()
- ❑Sequencer generates next data item from sequence and sends to driver as output of get_next_item()
- ❑Driver sends data item to DUT by driving interface signals
- ❑Driver indicates item is finished by calling item_done() from port.
- ❑Operation continues until all data items are sent.



3

From Week 6: Driver

```
class yapp_driver extends uvm_driver #(yapp_packet);

  // yapp_packet req;

  `uvm_component_utils(yapp_driver)

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

  virtual task run_phase(uvm_phase phase);
    forever begin
      seq_item_port.get_next_item( req );
      send_to_dut( req );
      seq_item_port.item_done();
    end
  endtask

  virtual task send_to_dut(yapp_packet packet);
    ... // protocol-specific syntax to drive DUT

endclass : yapp_driver
```

Extends uvm_driver with type parameter

Parameterization defines built-in yapp_packet handle req

uvm_component_utils macro

component constructor

Executed during UVM run phase

get_next_item returns yapp_packet instance

4

From Week 6: Sequencer

Extend uvm_sequencer

Parameterized driver needs a parameterized sequencer

```
class yapp_sequencer extends uvm_sequencer #(yapp_packet);

 `uvm_component_utils(yapp_sequencer)

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

Component macro

Component constructor

5

From Week 6: Agent

```
class yapp_agent extends uvm_agent;

  uvm_active_passive_enum is_active = UVM_ACTIVE;

  yapp_driver     driver;     // driver handle
  yapp_sequencer  sequencer;  // sequence handle
  yapp_monitor    monitor;    // monitor handle

  `uvm_component_utils_begin(yapp_agent)
   `uvm_field_enum(uvm_active_passive_enum, is_active, UVM_ALL_ON)
  `uvm_component_utils_end

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

  // build_phase():
  //   create instances

  // connect_phase():
  //   make connections

endclass
```

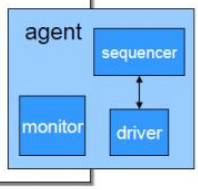
Extends uvm_agent no type parameter

Agent active flag using UVM type

Component macro

Automation for flag (important)

Component constructor



6

From Week 6: Build & Connect

```
class yapp_agent extends uvm_agent;
  uvm_active_passive_enum is_active = UVM_ACTIVE; //agent active flag

  // HANDLE, MACRO AND CONSTRUCTOR DECLARATIONS

 virtual function void build_phase (uvm_phase phase);
   super.build_phase(phase);
   monitor = new("monitor", this);
   if (is_active == UVM_ACTIVE) begin
     sequencer = new("sequencer", this);
     driver    = new("driver", this);
   end
 endfunction

 virtual function void connect_phase(uvm_phase phase);
   if (is_active == UVM_ACTIVE)
     driver.seq_item_port.connect(sequencer.seq_item_export);
 endfunction

endclass
```

super.build_phase is essential!

Agent sub-components created in built (sub-components then implicitly built)

Connect sequencer and driver

Connect method is called *from* the port and takes the export as an *argument*

- Create the components & objects using create().
  - my_driver=driver::type_id::create("my_driver",this);
  - my_data=data_item::type_id::create("my_data");

7

NEW: Monitor

- Observe DUT input & output interfaces
- Capture observations in packets
- Transfer[input]/Compare[output] packets to Scoreboard

- Which fields must we populate in the packets?
  - Depends on the DUT
  - *In our case just use HTAX packets (TX and RX)*
  - We'll use YAPP packets for these slides

8

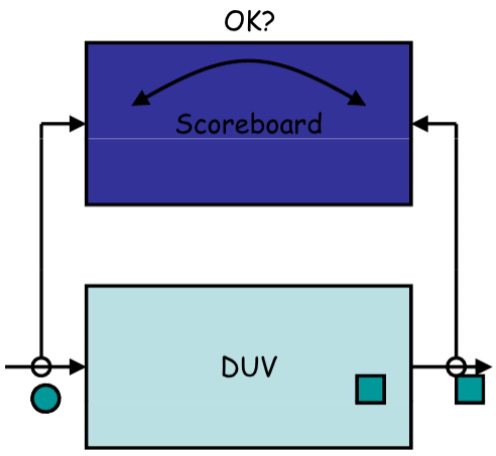
NEW: Scoreboard

- General principle of what went into the design and predicting what should come out of the design
- Input from Driver goes to both DUT and input Monitor. *In our case TX Monitor sends HTAX packets to Scoreboard (stored as Expected Data).*
- DUT output goes to output Monitor. *In our case RX Monitor compares HTAX packets to Expected Data that was stored in Scoreboard.*

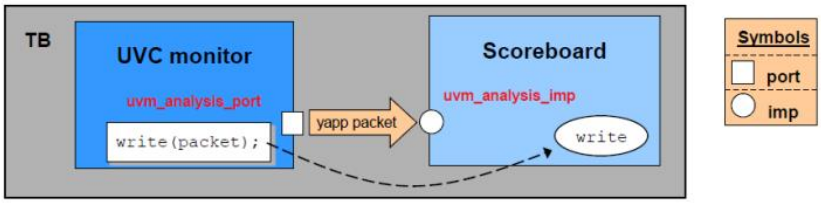
SCOREBOARD CONTAINS THE LIST OF THINGS WE EXPECT TO SEE COME OUT OF DUT.

9

Scoreboard Operation



Implementation



Scoreboard (consumer):

- Defines implementation of transaction using a `write` method.
- Declares an `uvm_analysis_imp` object, parameterized for data type.

UVC monitor (producer):

- Declares an `uvm_analysis_port` object, parameterized for data type.
- Initiates transfer by calling write method off port object.
  - Mapped to module implementation.

`imp` and `port` are connected together in higher hierarchical level (testbench).

11

Analysis Interface Example

```
class yapp_monitor extends uvm_monitor;
  ...
  uvm_analysis_port #(yapp_packet) yapp_out;

 function new (string name,
               uvm_component parent);
   super.new(name, parent);
   yapp_out = new("yapp_out", this);
 endfunction : new

  yapp_packet collected_packet;
...
  task collect_packets();
    // Collect Header {Length, Addr}
    // Collect the Payload
    // Collect Parity and Compute Parity Type
    // Send packet to scoreboard
    yapp_out.write(collected_packet);
    ...
  endtask
...
```

YAPP UVC Monitor

```
class router_sb extends uvm_scoreboard;
  ...
  uvm_analysis_imp
    #(yapp_packet, router_sb) yapp_in;

 function new (string name,
               uvm_component parent);
   super.new(name, parent);
   yapp_in = new("yapp_in", this);
 endfunction : new

...
  function void write
    (input yapp_packet packet);
    case (packet.addr)
       2'b00:q0.push_back(packet);
       ...
  endfunction
...
```

Module UVC Scoreboard

Analysis port object:

- Type parameterized for transaction type
- Call `write()` from port with transaction argument

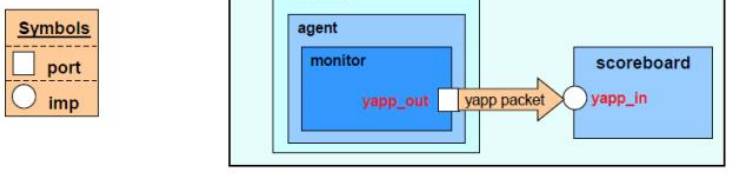
Analysis imp object:

- Type parameterized for transaction and component type
- Scope of object must have implementation of `write()`

12

Analysis Interface Connection

```
class router_testbench extends uvm_component;
  yapp_env    yapp;
  router_sb   scoreboard;
  ...
  virtual function void build_phase(uvm_phase phase);
    // create UVC instances
    ...
  virtual function void connect_phase(uvm_phase phase);
    yapp.agent.monitor.yapp_out.connect(scoreboard.yapp_in);
    ...
```



13

Scoreboarding for HTAX

- Recording a packet in the Scoreboard:
  - The TX Monitor(s) has an analysis port
  - A TLM channel forwards the packet to the analysis implementation in the Scoreboard.
  - Current packet is copied to Scoreboard.

- Checking for a packet in the Scoreboard:
  - The RX Monitor(s) has an analysis port
  - A TLM channel forwards the packet to the analysis implementation in the Scoreboard.
  - Try to find the matching packet in the Scoreboard.

Assume: DUV does not change order of packets.
- Hence, first packet in scoreboard has to match received packet.

SUMMARY: Scoreboards

- Scoreboards are smart data structures that keep track of events in the DUT during simulation
- Usually, scoreboards are global
  - One scoreboard per verification environment
- Scoreboard are not checking mechanisms, but
- The main purpose of using scoreboards is for checking
  - In practice, many checkers are implemented inside
  - There are many typical checks that are done with scoreboards

SUMMARY: Scoreboards

- Sources of information to the scoreboard
  - Primarily, the inputs and outputs of the DUT
  - Internal events can also be used
- Types of checks done with scoreboard
  - Matching between inputs and outputs
    - Nothing is lost
      - Input with no matching output
    - Nothing is born
      - Output with no matching inputs
    - Data matching
  - Timing rules
    - Delay from input to output is within limits
  - Ordering rules
- Scoreboards are very useful in data flow designs
  - Routers and Fifos