DAY 85 - 100 DAYS VERIFICATION CHALLENGE

Topic: UVM Driver & UVM Monitor

DAY 85 CHALLENGE:

1. Write a uvm driver template and explain each line.
2. Explain below methods in uvm_driver:
   i. get_next_item
   ii. try_next_item
   iii. item_done
   iv. put
3. Explain the protocol handshake between a sequencer and driver?
4. What is the difference between a pipelined and non-pipelined driver?
5. Write a uvm monitor template and explain each line.
6. What does a monitor do?
7. What is the difference between a monitor and a scoreboard in UVM methodology?
8. How do you connect a monitor with a scoreboard?

classmate
Date
Page

antityagi uvm2024

DAY : 85

Topic : Uvm Driver & Uvm monitor :

Sol^n① Write a uvm_driver template and explain each line.
- A driver is written by extending the uvm_drive.
- uvm_driver is inherited from uvm_component, methods and TLM port are defined for communication between sequences and driver.
- The uvm_driver is a parameterized with the type of the request sequence_item and the type of the response sequnce_item.

◎ DRIVER template :

```
Class driver extends uvm_driver #(tx_class);
virtual intf_class vif;

 `uvm_component_utills (extends_class_name)
//constructor
    function new (string name, uvm_component parent);
        super.new (name, parent);
    endfunction
// Build phase
    function void build_phase (uvm_phase phase);
        super. build_phase (phase);
        if (!uvm_resouce_db # (virtual intf_class ):: read_by_name (
                        "GLOBAl", "INTF", vif, this)
     endfunction
// Run phase
   task run_phase (uvm_phase phase);
        forever begin
            seq_item_port. get_next_item (req);
            req.print();
            // drive (req);
            seq_item_port .item_clone ();
//drive     endtask
        task drive (tx_class tx)
        endtask     // drive signal DUT.
```

- Run Continuously with a forever loop.
- obtains the next transaction item afer one Complets.
- Bi-dirctional Communication with the sequencer, item-done Can send a response to the sequencer.

Sol^n ② Explain below methods in uvm driver:

i.) get_next_item:
This method blocks until a REQ Sequence_item is available in the Sequencer

ii.) try_next_item:
- This is a non-Blocking variant of the get_next_item() method.
- or its similar to get_next_item, but with the subtle difference
- it attempts to retrive the next transaction item without actually remaing it from the Sequencer's queue.
- or it will return null pointer if there is no REQ Sequence item available in the sequencer.

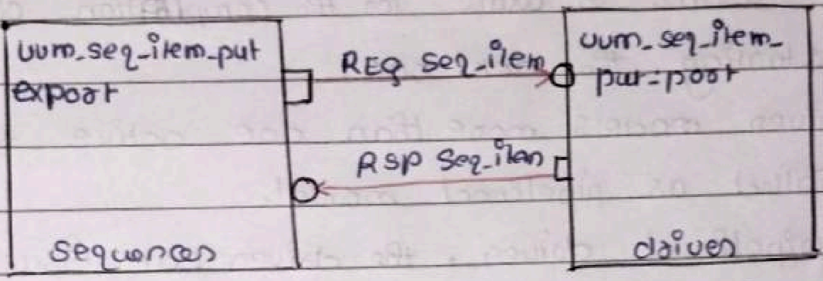
iii.) item_done:
- This is non_blocking item_done method Completes the driver sequencer handshake and it should be called after a get_next_item() or successful try_next_item() call.
- or This method is typically called by the driver to inform the sequencer that it has finished processing a transaction item.

iv.) put:
- The put() method is a non-Blocking methods.
- it is used to place an RSP Sequence_item in the sequencer or this method is used to send a

classmate
Date
Page

Sol③ Explain the protocol handshake between a sequencer and driver?

- The connect Between a driver and Sequencer is a one to one Connection.
- multiple driver are not Connected to a Sequencer nor are multiple sequencer Connected to single driver.



- If we want to send the transaction from the ~~driver~~ Sequence to Driver in order to provide stimulus to the DUT.
- The transfer of request and response sequence items Btw Sequencer and their target driver is facilated by a TLM Communication implemented in the sequencer.

start_item : This request the sequencer to have access to the driver for the Sequence item & returns when the sequencer grant access.

finish_item : This method results in the driver receiving the sequence item and its Blocking method which returns only after driver calls the item_done() method.

get_next_item (req): This is Blocking method in driver that blocks untill a sequence item is received on the port connected the sequencer.

item_done (req) : The driver uses this non Blocking call to signal to the Sequencer that it can unblock the sequences finish_item() method, either when the driver accepts the sequences request or it has executed it.

Soln④ what is the difference Btween a pipelined and non-piplined driver?

- if the driver models only once active transaction at a time then its called a non-pipelined model.
- or in a non-pipelined driver, the driver send transactions one at a time & waits for the completion of each transaction before starting the next one.
- if the driver model's more than one active transaction at a time then it called as pipelined model.
- or in a pipelined driver, the driver can issue multiple transaction in parallel without waiting for the completion of each transaction.

Soln⑤ what does a monitor do?

- uvm monitor is a passive component used to capture DUT signal using a virtual interface and translate them into a sequence item format.
- These sequence item or transactions are broadcasted to other components like the scoreboard, coverage etc.
- its use a TLM analysis port to broadcast transaction.

Soln⑥ what is the difference Btween a monitor and a scoreboard in uvm?

monitor	Scoreboard
• a monitor is a component that observes pin level activity and converts its observations in to transactions or sequence items. • it also sends those tx to analysis components through an analysis port.	• A scoreboard is an analysis components that checks if the DUT is behaving correctly. uvm • Scoreboard use analysis tx from the monitor implemented inside agents.

solⁿ ⑦ write a uvm_monitor template and explain each line.

```
Class monitor extends uvm_monitor;
  virtual intf_class vif;          //declare virtual interface

  uvm_analysis_port #(tx_class) ap_port;  //declare analysis
                                              port
   tx_class tx;
   `uvm_component_utils (monitor)
 //Constructor
      function void build_phase (uvm_phase phase);
           Super.build_phase (phase);
      if (!uvm_resource_db #(virtual intf_class) :: read_by_name(
 //Buildphase                        "GLOBAL", "*", "VIF" this);
         ap_port = new ("ap_port", this);
      endfunction
 //run_phase
      task run_phase (uvm_phase phase);
          // capture the signal to be monitored
          ap_port. write (tx);
       endtask
   endclass
```

- The user defined monitor is extended from uvm_monitor. uvm_monitor is inherited By uvm_component
- monitor Samples DUT Signal but does not drive them
- declare virtual interface and connect interface to virtual interface by using get method.
- decare anylis port. and also decdare transaction class instance
- add Sampling logic to in run_phase.
- After sampling, by using the write method send the Sampled transaction packet to the scoreboard.
- uses a one-to-many connection (Broadcast model).

DAY 86 - 100 DAYS VERIFICATION CHALLENGE

Topic: UVM Agent, config_db

DAY 86 CHALLENGE:

1. Write a uvm agent template and explain each line.
2. What are Active and Passive modes in an agent?
3. What is get_is_active() method in uvm_agent? Why do we need it?
4. What is uvm_config_db? Why do we need it?
5. What is uvm_resource_db?
6. Explain the difference between uvm_config_db & uvm_resource_db.
7. How set_config_* works?
8. What is the difference between set_config_* and uvm_config_db?
9. Can we use set_config and get_config in sequence?

Aditiyagi-uvm2024

DAY:86

Topic : Uvm agent , Config db

Sol'1) Write a uvm agent template and explain each line.

- Agent Act as intermediates Btw the testbench and the DUT. They encompass drivers, monitors and sequencers and helps in organizing and managing the verification process.

```
class     m_agent extends  uvm_agent;

      driver   drv;
      monitor  mon;
      sequencer sqr;
      Coverage  cov ;

       'uvm_component_utils ( m_agent)

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

   function void   build_phase (uvm_phase phase);
     drv = driver :: type_id :: create ( "drv" , this);
     mon   = monitor :: type_id :: create ("mon", this);
     sqr   = sequencer :: type_id :: create ("sqr", this);
     cov   = coverage :: type_id :: create ("cov", this);
   endfunction

   function void Connect_phase (uvm_phase phase);
            Super. connect_phase (phase);
     drv.seq_item_port . connect ( sqr . seq_item_export);
     mon . ap_port . Connect (cov. analysis_export );
   endfunction
endclass
```

Date
Page

Sol^n② what are Active and passive modes in an agent?

- in uvm active mode & passive modes have specific meaning related to the way a Component responds to events & transaction in the Testbench envioroment.
- Active mode: in uvm active mode refers to a mode of operation where a Component initiates transactions or events, such as sending request for driving signals.
- An active Component typically has a task or function that actively generates stimulus. Such as driver or Sequencer. or Active agents generate stimulus and drive to DUT.
    - An active agent shall consists of all three Component driver, sequencer & monitor.
- passive mode: Passive agents sample DUT signals but do not drive them.
    - A passive agent consist of only the monitor or Scoreboard.

Sol^n③ what is get_is_active () method in uvm agent? why do we need it?

- The get_is_active() function is used to find out the type of agent.
- get_is_active() Returns uvm_active if the agent is acting as an active agent & uvm_passive if the agent acting as a passive agent.
- The driver, Sequencer instance are created if its an active agent and monitor instance can be created by default irrespective of agent type.

Solⁿ ④ what is uvm_config_dB ? why do we need it ?

- uvm config db is a static class that provides a database for storing and retrieving configuration values.
- This database can be accessed from anywhere in the verification enviroment & can be used to pass configuration information Btw components that are not directly connected.
- The uvm config dB provides two main function : set() and get()
- The set() function is used to set a configuration value in the database. and takes three arguments : The first argument is the value to be assigned to that field and the third argument is the hierarchical path of the component the own the field.
- The get() function is used to retrive a configuration value from the database & take two arguments : first argument is the name of the configuration field. & the second argument is the hierarchical path of the component that owns the field.
- One of main advantages of using uvm config db is that it allows configuration values to be passed Between components that are not directly connected. for example, a top-level testbench component can set a configuration value using uvm config db:: set(), and this value can be retrived by a lower-level component that does not have a direct connection to the top-level testbench.

Solⁿ ⑤ what is uvm resource_db ?

- its a mechanism used in hardware verification to store & retrive configuration & other data in a standardized way.
- uvm resource_dB is that it allows different components or module within your verification enviorement to easy share & access data withou having to establish direct connection or dependency Btw them.

Sol^n ⑥ Explain the difference Btween uvm_config_db & uvm_resource_db.

- All the methods declared in both them are static in nature so it should be declared with the scope resolution operator as uvm_config_db :: set() and uvm_resource_db :: set().
declaration of each one of them:

uvm_config_dB:-	uvm_resource_dB;
⊙ static function void set(uvm_comp cntxt, string inst_name, string field_name, T value)	⊙ static function void set(input string scop, input string name, T value, input uvm_object accessor = null)
⊙ So uvm_config_db is primarily used in places to acess the resource or parameter where hierarch is important.	⊙ while uvm_resource_db should be used in places to acess the resource in case of non-hierarchical context.
⊙ uvm_config_dB is used when you want to set/get resource within uvm component hierarchy	⊙ uvm_resource_db is used when you want to set/get resauce except uvm component hierarchy.
⊙ The number of uvm_config_db are limited many 4 such as get, set, exits and wait_modified	⊙ while the method for uvm_resourcedb are more like get_type - get_by_name, read_by_name, read_by_type, write_by_name, write_by_type, set_anonymous.

Sol^n ⑦ How set_config_* works?

- when set_config_* method is called, the data is stored w.r.t string in a table. There is also a global configurable table.
- These function provides a standandized way to set configura parameter for uvm components making the envioment more flexible and configurable.

Sol^n 8) what is the difference Between set_config* and uvm_config_db

- uvm_config.dB is a global configoration database provided by uvm for passing configuration information between different components in different hierarches.
- it allows components at different levels of the hierarch to share and retrive configuration information.
- use set_config* for direct & specific configuration setting at the point where the configuration is needed.

Sol^n 9) Can we use set_config and get_config in Sequence?

- yes we can use set_config* and get_config* methods in a sequence to set and retrive configuration values for the sequence.
- when you call set_config* on a sequence, it updated the configurat database with the new value for the specified configuration parameter.
- This value can be retrived using get_config* method. By using set_config* & get_config* methods.

example:

```
Class my_sequence extends uvm_sequence #(my_tx);
        'uvm_object_utils (my_sequence)

    function void body();

        if (! uvm_config_db #(int) :: get(this, " ", "my_param",
                                          my param_value))

            'uvm_error (get_name() , "faild to get my_param_v

    end function
endclass
```

DAY 87 - 100 DAYS VERIFICATION CHALLENGE

Topic: UVM Scoreboard, UVM Testbench – top, test, env

DAY 87 CHALLENGE:

1. What is UVM Scoreboard?
2. What is the use of UVM Scoreboard?
3. Write a uvm_scoreboard template & explain each line.
4. How is scoreboard connected to different components?
5. What is an in-order and out-of-order scoreboard?
6. Explain below components in a UVM testbench with code:
    i. top
    ii. test
    iii. env
7. How can you define custom types for use in your env?

Adityag_uvmx24

DAY : 27

Topic : Uvm Scoreboard, uvm Testbench : Top, Test, env

Solⁿ ① What is Uvm Scoreboard ?

- Scoreboard compare the expected results with actual results from the DUT. They play a crucial role in verifying the Correctness OF the design.
- OR The uvm Scoreboard is a component that check the functionality of the DUT. it recieves transactions from the monitor using the analysis export for checking purpases.

Solⁿ ② What is the use of uvm Scoreboard ?

- Scoreboard is a verification component that contains checkers and verifies the functionality of a design.
- using scoreboard can check the expected and actual output with reference model.
- Scoreboard is a crucial part of the verification process, providing a mechanism to asses and validate the correctness of the DUT 's behavior during simulation.

Solⁿ ③ How is Scoreboard connected to different Components ?

- In other Components in the testbench send data to the Scoreboard via an analysis port by Calling the ports write method. for example, a monitor collects data packet from the bus interface, The packet is Complete when the bus operation has received or sent all the data associated with the transfer.

Solⁿ④ How or write a uvm-scoreboard Template & explane each line

```
class my-scoreboard extends uvm-scoreboard;
// factory registration
    uvm component utils (scoreboard)
//declaration of analysis port
    uvm-analysis-imp-port #(tx-class) ap-port;
//constructor
    function new (string name, uvm-component parent);
        super.new (name, parent);
    endfunction
// build-phase to create object for ap-port imp port
    function void build-phase (uvm-phase phase);
            super.build-phase (phase);
            ap-port = new ("ap-port", this);
    endfunction

    function void write (tx-class tx);
        ------
    endfunction

    task run-phase (uvm-phase phase);
     forever begin
        tx-class tx;
       //implement Comparision logic
     end
    endtask
```

- Compare actual output with Correct outputs.
- often involves sending & receiving data: a queue is commonly used.
- Requires write functions. for analysis ports.

Soln 5) what is an in-order and out-of-order Scoreboard?

- in-order Scoreboard. transactions or events are processed and checked in the order they are received or generated.
- its usefull for the design whose output is the same as driven stimuli. The Comparator will Compare the expected and actual output streams in Same order.
- The out-of order scoreboard is usefull for the design whose output is different from driven input stimuli.
- Based on the I/P stimuli reference model will generate the expected outcomes of ~~OUT~~ any order. unmatched & transaction generated from the Input Stimulus until the corresponding output has been received from the DUT to be Compare.

Soln 6) Explain below Component is a uvm Testbench with code
i> top ii> Test iii> env

i> Top:

- This is the topmost file. which Connect the DUT & Testbench.
- it Consist of DUT, Test & interface instances.
- The interface connects the DUT and Testbench
- or its Refers to the Top-level module or component with in a uvm_Testbench. and Control the overall verification process.

```
include "uvm_pkg.sv".
import uvm_pkg::*;
module top;
 bit clk, rst
 class_intf pif ( clk, rst);

  initial begin
     clk = 0;
   forever #5 clk = ~clk
  end

  initial begin
        run_test ("Base_test");
   end
```

```
initial begin
    uvm_resource_db #(virtual class_intf) :: set ("GLOBAL", '*', intf,
end                                                           null)
endmodule
```

- Top-level wrapper for the entire Testbench.
- run_test function executes specific tests from the testclass.
- Set the virtual interface required by various components.

② Test :

- Specific type of uvm Component that represent test cases or test scenario. each Test has its own verification enviornmnt

```
class base_test extends uvm_test;
  env_class env;
    'uvm_component_utils (env_class )

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

   function void build_phase (uvm_phase phase);
          super. build_phase (phase);
       env = env_class :: type_id :: create ("env", this);
   endfunction

      Class write_rd_base_test extends base_test;
      'uvm_component_utils (wr_rd_base_test)
   //Constructor
          NEW_comp
   // Build_phase
      function void build_phase (uvm_phase phase);
              super. build_phase (phase);
      endfunction
    task  run_phase (uvm_phase phase);
          wr_rd_base test seq     base_test_seq;

    base_test_seq = wr_rd_base_test_seq :: type_id :: create ("base_test_seq");

      phase . raise_objection ( this );
      phase . phase_done . set_drain_time ( this , 100 );
      wr_rd_seq . start ( env . agent . sqr );
        phase . drop_objection ( this );
  endtask
endclass
```

(7) **Env** :- The enviroment is a container component for grouping higher level components like agents & scoreboard.
• agent, coverage & scoreboard should be encapsulated with in enviroment.

```
  class    enviroment   extends   uvm_env;
        agent    magent;
        scoreboard   sbd;

        uvm_component_utils ( enviroment )

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

        function  void  build_phase (uvm_phase phase);
              magent  =  agent :: type_id :: create ("magent", this);
        endfunction

        function  void   connect_phase (uvm_phas phase);
              super . connect_phase (phase);
          magent . mon . ap_port . connect (scb . imp_port);
        endfunction
endclass
```