DAY 89 - 100 DAYS VERIFICATION CHALLENGE

Topic: UVM Testbench Architecture

DAY 89 CHALLENGE:

1. Explain:
    i. Block level testbench
    ii. Integration Level Testbench
2. How is coverage collection done in a UVM Testbench?
3. What is factory override in UVM?
4. What is BFM?
5. How do you assign a Virtual Interface in UVM?
6. Write a code to connect sequencer & driver in the connect phase.
7. What are the sub components in uvm env? Explain in detail.

Adityagi-uvm2024

DAY : 89

Topic : Uvm Testbench Architecture

Sol^n ① Explain :

i.> Block level Testbench ii.> Integration level Testbench

- **Block level Testbench :**
- Block level Testbenches by verifying that all the connections between the Blocks are functioning correctly, that data flows properly around the SoC and that hardware interrupts and their associated interrupt service routine (ISR) software are working.
- It focuses on testing the individual components or blocks in isolation without considering how they interact with other parts of system.

- **Integration level Testbench :-**
- This level Testbench operates at a higher level compared to the block level.
- It is designed to test how different blocks or modules interact and work together within your overall system.
- Integration level testing verifies that the connections between Blocks are correct data is following properly Between them and the overall system Behaves as expected ~~if not or errors is~~ when all the components are combined.

Sol^n ② **How is Coverage collection done in a uvm Testbench ?**

- Coverage collection is an essential aspect of verifying the completeness and correctness of your design.
- To collect functional coverage in uvm, you typically create covergroups in your Testbench components (eg : Test, Sequence & monitor)
- you then instantiate & configure the covergroup within your Testbench components & you can sample the covergroup during simulation to collect coverage information "sample()" method.
- Code coverage is automatically collected by the tool itself.

Soln ③ What is factory override in UVM?

- Reusability is the main advantage of the UVM Testbench. To replace the components in TB, we need to replace the components & their objects in their parent components. UVM factory is used to do this.
- The purpose of the factory is to replace the object of one type with the its derived type from the top level test class.
- Therefore we need not touch the Testbench code. This is called overriding the components.

④ There are two types of overriding

i) Override by type & ii) override by instance.

- Before overriding, we need to register the component with factory & construct it.
- Or The purpose of the factory overriding is to replace the object of one type with the derived type from the top level test class without touching the Testbench code. This is called override the component.

Soln ④ What is BFM?

- A BFM (Bus functional model) is a more generic concept.
- The Bfm for a device interact with the DUT by both driving & sampling the DUT signals.
- Bfm describes the functionality & provides a cycle accurate interface to DUT.

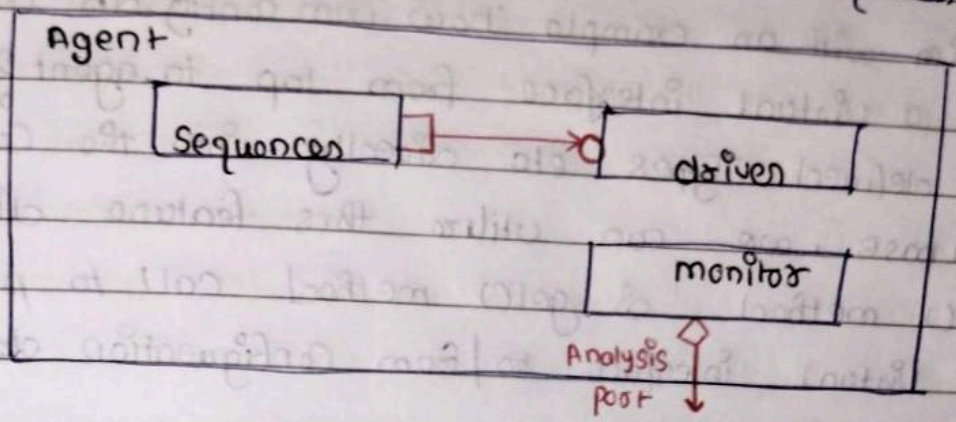
Soln ⑤ How do you assign a virtual interface in UVM?

- This can be done using the configuration database, using config_db, the virtual interface can be set in the top level component in a particular name and using the same name it can be get in its lower level components.

uvm_config_db #(virtual intf)::set(null, "KEY1", "KEY2", pif);

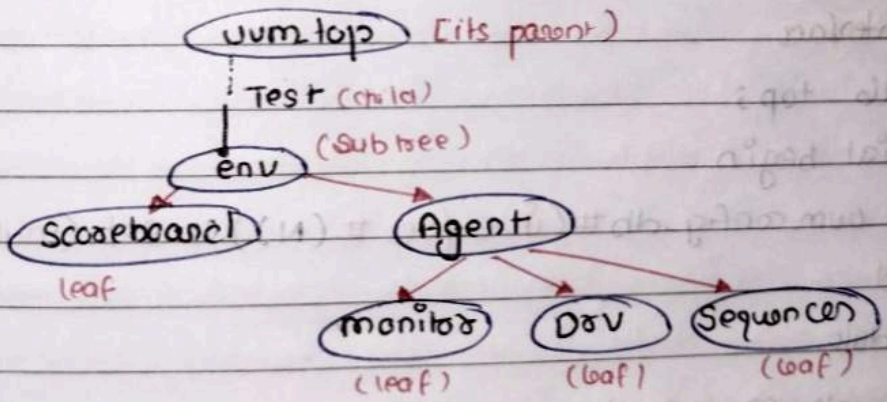
Soln ⑥ write a code to connect sequencer & driver in the connect phase.

⊙ in the connect_phase, connect driver and sequencer components




```
function void connect_phase (uvm_phase phase);
    super.connect_phase(phase);
    drv.seq_item_port.connect(sqr.seq_item_export);
endfunction
```

Soln ⑦ what are the sub components in uvm env? Explain in detail.



- Environment contain agent, scoreboard and also coverage, in environment component the object of subcomponent is created.
- agent: its responsible for interfacing between the tb & the dut. It contains driver & monitor to interact with the DUT signal.
- Scoreboard: The scoreboard compare the actual output or behaviour of the DUT with the expected behaviour defined by the Test.
- Coverage: Calculate the coverage metrics to verify the number of possible values covered in verification.

DAY 90 - 100 DAYS VERIFICATION CHALLENGE

Topic: UVM Testbench to DUT Connections

DAY 90 CHALLENGE:

1. Explain with an example how uvm_config_db can be used to pass a virtual interface from *top* to *agent*?
2. Write a code for virtual interface in uvm which has:
    i. Port declarations
    ii. Signal & Variable declarations
    iii. Modports
    iv. Other interfaces
    v. Any other system Verilog code as need be
3. Illustrate with an example how to use parameterized interfaces with the uvm_config_db
4. How can you connect abstract class with a concrete class in UVM?

Date
Page

Adityag_uvm2024

DAY : 90

Topics : uvm Testbench To DUT Connections

Soln ① Explain with an example how uvm_config_db can be used to pass a virtual interface from top to agent?

- user defined Types etc directly into the configuration database. we can utilize this feature directly & use set() method & get() method call to place and retrive the virtual interface to/from configuration database.

Soln ② Illustrate with an example how to use parameterized interface with uvm_config_db.

- Parameterized interface passing throug config_db:-

→ module top;
initial begin
uvm_config_db#(interface #(N))::set(null, "KEY1", "intf", pif);
end
endmodule

→ class driver extends uvm_driver;
virtual intf#(N) v_intf;

if(!uvm_config_db#(intf#(N))::get(this, "", "", v_intf);
`uvm_error("CONFIG_ERROR", "failed to set VIF in config_db")
endclass

Soln ④ How can you connect abstract class with a Concrete class in uvm?

- in uvm, you can abstract class with a concrete class by inherit the abstract class in the concrete class using factory override.
- The abstract class can contain method and properties that can be used in both the abstract & concrete classes. The concrete class can then override any method or extends any properties from class.

Soln② write a code for virtual interface in uvm which has:

1.> port declaration:

- Port declaration include clock (clk), reset (rst), data (data) & control signals (valid & ready).
- signal and variable declaration:
- it include internals signals and variable used within the interface.
- modports:
- modports master & slave define different view of the interface.
- another interface:
another interface is also included within the virtual interface.
- Constructor() initializes the other interface instance.

→ interface:

```
interface intf (logic input clk, rst);
  logic [3:0] data;
  logic [3:0] addr;
```

→ modport:

```
modport master_mp (input clk, input rst, data,
                   output rdata);
```

→ interface other_interface;

```
// define other interface signals/methods
endinterface
  function new();
    other_intf inst = new();
  endfunction
endinterface
```