**Testbench Components Description**

1. **Top module**

* All Verification components, interfaces are instantiated in a top module called testbench.
* It is a static container to hold everything required to be simulated and become the root node in the hierarchy.
* Simulators typically need to know the top level module so it can analyze components within the top module and elaborate the design hierarchy.
* It is required to import uvm\_pkg in order to use UVM constructs in the module.
* The interface is set as an object in uvm\_config\_db via set method and will be retrieved in the testclass using the get method.

1. **Environment**

* Environment class is used to implement verification environments in UVM.
* It is an extension of the uvm\_env class.
* The testbench simulation needs some systematic flow like building the components, connecting the components, starting the components etc.
* Uvm\_env base class has methods that formalize the simulation steps.
* All the methods inside the environment class are declared virtual.
* Virtual interface is created in the environment and all other virtual functions of the environment class are extended.
* Our environment is the top level of the class based part of the testbench.

1. **Test class**

* The uvm\_test class defines the test scenarios for the testbench for the DUT and is specified in the test.
* Testcase contains the instance of the environment class.
* This testcase creates an environment object and defines the required test specific functionality.
* Verification environment contains the declarations of the virtual interfaces.
* These virtual interfaces are pointed to the physical interfaces which are declared in the top module.
* These virtual interfaces are made to point to the physical interface in the testcase.

1. **Interface**

* It is a SystemVerilog interface.
* Interface serves as the actual link between the design under verification and the verification environment.
* The interface describes the pin-level description of the DUT.
* An interface is basically a bundle of nets or wires through which a testbench communicates with a design.

1. **Virtual interface**

* It provides a mechanism for separating abstract models from the actual signals of the design.
* A virtual interface allows the same instance or the subprogram to operate on different parts of the design.
* It dynamically controls the set of signals associated with the subprogram, this allows passing the same data over all the components.

1. **Transaction**

* Interfaces represent the input to the DUT.
* The fields and attributes of transactions are derived from the transaction’s specification.
* In a test, many data items are generated and those are sent to the DUT via driver.
* Generally data item fields are randomized using SystemVerilog constraints and many tests can be created.

1. **Agents**

* Most DUTs have a number of different signal interfaces, each of which have their own protocol.
* The UVM agent collects together a group of uvm\_components focused around a specific pin-level interface.
* The purpose of the agent is to provide a verification component which allows users to generate and monitor pin level transactions.

1. **Sequencer**

* A sequence is the series of transactions.
* Sequencer is used to control the flow of transaction generation.
* A sequence is extended from uvm\_sequence class.
* uvm\_sequencer does the generation of this sequence of transactions.
* Driver (extension of uvm\_driver) takes the transactions from Sequencer and processes the packets of data or drives them to other components or to the DUT.
* It allows the addition of constraints to the data item generated in the sequence.

1. **Driver**

* Driver is defined by extending uvm\_driver.
* Driver takes the transactions from the sequencer by using seq\_item\_port.
* These transactions will be driven to DUT as per the interface signal specifications.
* Then it sends the transaction to the scoreboard using uvm\_analysis\_port.
* Tasks for resetting DUT and configuring the DUT are also declared here.
* An instance of the driver class is created in the environment class and the sequencer is connected to it.

1. **Monitor**

* A monitor is a passive entity that samples DUT signals but doesn‘t drive them.
* A monitor collects transactions (data items).
* Extracts events, performs checking and coverage.
* Optionally prints trace information.
* Checking typically consists of protocol and data checkers to verify that the DUT output meets the protocol specification.
* Coverage is collected in the monitor.
* It is implemented by extending the uvm\_monitor class and an instance is created in the environment for hooking it up with DUT signals.
* A UVM monitor is responsible for capturing the signal activity from the design interface and translating it into transaction level data objects that can be sent to the other components.

1. **Scoreboard**

* Scoreboard is implemented by extending uvm\_scorboard.
* Scoreboard has 2 analysis imports.
* One is used for getting the packets from the driver and other from the receiver.
* Then the packets are compared and if they don't match, then error is asserted.
* Compare function of the transaction class is used for comparison.