**I2C UVM TESTBENCH ARCHITECTURE WITH DESCRIPTION OF COMPONENTS**

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

♣ Virtual Interfaces - It provide 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.

♣ Transactions - 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 System Verilog constraints many number of tests can be created.

♣ 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. I2C agent used by the Testbench communicates with the DUT, and to create background traffic. Wishbone agent is used to drive the DUT via the DUT's wishbone interface

♣ Sequence And Sequence - A sequence is the series of transaction and 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 transaction. Driver (extension of uvm\_driver) takes the transactions from Sequencer and processes the packets of data or drives them to other component or to the DUT. It allows the addition of constraints to the data item generated in the sequence, thus bringing forth the corner cases.

♣ 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 scoreboard using uvm\_analysis\_port. Task 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.

♣ 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.

♣ Scoreboard - Scoreboard is implemented by extending uvm\_scorboard. Scoreboard has 2 analysis imports. One is used to 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 transaction class is used for comparison.

♣ Environment - Environment class is used to implement verification environments in UVM. It is extension on uvm\_env class. The testbench simulation needs some systematic flow like building the components, connection the components, starting the components etc. uvm\_env base class has methods formalize the simulation steps. All the methods inside environment class are declared virtual. Virtual interface is created in the environment and all other virtual functions of environment class are extended. Our environment is the top level of the class based part of the testbench. It also contains the virtual sequencer that is used to run sequences to coordinate the wband i2c agents sequences

♣ Testcases - The uvm\_test class defines the test scenario 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 physical interface in the testcase.

♣ Top Module - SystemVerilog interface instance is created in this module. DUT instance is created and hooked up with the interface instance. Clock generator is implemented here. run\_test method is called from here. The test name can be implicitly passed or can be passed as a command line argument during simulation. The command line argument takes greater precedence.

