Fundamentals of Pre-Silicon Validation || Spring -2024

Implementation and Verification of Asynchronous FIFO using both Class based and UVM methodologies.

UVM VERIFICATION TEST PLAN

Project Team:

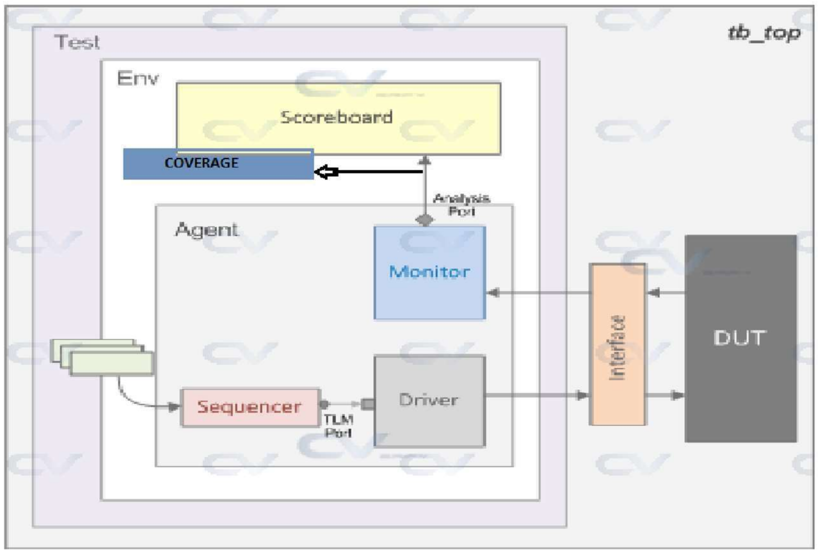
Jaswanth Pallappa, jaswanth@pdx.edu

Brunda Marpadaga, brunda@pdx.edu

Maha Lakshmi Potabattuni, mahalak@pdx.edu

Akhila Veedula, akhilav@pdx.edu

UVM TestBench Hierarchy:



Sequence Item: This is a collection of all inputs that are randomized and defined according to the design needs and requirements. All the inputs that need to be randomized are taken from the Device Under Test (DUT). The actual randomization of all these inputs happens in the Sequence and these are then sent through the sequencer to the driver.

Sequence: A sequence is a collection of sequence items and control logic that defines a specific verification scenario. Sequences control the generation of stimulus to drive the DUT and monitor its response. Sequences can be hierarchical and modular, allowing for easy reuse and scalability. In the UVM test bench environment, two different sequences are used for two different operations which depend upon read and write enable. The fifo_sequence_wr and sequence_fifo_rd are the two sequences for read and write operations.

Sequencer: The sequencer controls the generation and sequencing of stimulus transactions. It coordinates with the testbench to select and execute sequences based on the test requirements. The sequencer communicates with the driver to transmit stimulus to the DUT and receives responses from the monitor. Build phase and connect phase are the phases of UVM used in the Sequencer to connect to the driver and pass on the item from sequencer to the driver.

Driver: The driver is responsible for converting transaction-level stimulus generated by the sequencer into signals or transactions suitable for transmission to the DUT. It drives the actual interface signals to stimulate the DUT. There are Build Phase, Connect Phase and Run Phase responsible for connecting and responsible for performing different operations and for example:

connecting with the DUT in build phase and driving the signals from DUT in run phase inside the driver.

Agent: An agent represents a logical entity responsible for interfacing with a specific part of the DUT. It typically consists of a driver, monitor, sequencer, and possibly a scoreboard. Agents abstract away the details of the interface protocol and provide a clean interface for stimulus generation and response monitoring. Build phase and connect phase and run phase along with creating a new constructor for the agent class which calls the create method for the sequencer , driver, and monitor using the factory registration methods from the uvm_object class in build phase and also used the connect phase to connect the port of driver to the sequencer export.

Monitor: Monitoring the activity on the DUT's interfaces or signals of interest is the monitor's responsibility. It records events or transactions that take place on designated interfaces by serving as an interface between the DUT and the testbench environment. The monitor continually gathers input and output data from the DUT and transforms it into transaction-level data that may be used by other testbench components to process. Monitor uses Build Phase to ensure proper connection with the interface using uvm_config_db and prompts an error if it is failed to connect with the interface and also uses connect phase and run phase to connect with the other components in uvm verification and to get the signals from the DUT through the interface.

Scoreboard: The scoreboard is in charge of confirming that the DUT is acting appropriately by contrasting its output with the predicted output that the testbench generates. It serves as a benchmark model by which the behavior of the DUT is verified. Transaction-level data is sent from the monitor to the scoreboard, which then compares it with the testbench's predicted outcomes. A reference fifo queue is declared in the scoreboard as trans. The data_in from the seq_item is pushed into the queue and popped into another temp_data. The temp_data and the actual data_out are compared. There is a write method to push the values into the queue and task read method to get the values from the reference queue and compare the output with the actual data_out. Scoreboard contains an analysis port and that is connected to different components in the uvm testbench hierarchy. An object is created for this in the scoreboard build phase along with the connect phase and run phase , which majorly focus on checking the actual values with the expected values during read and write operation.

Environment: The environment represents the primary container for organizing and managing the verification components. It includes agents, drivers, monitors, sequencers, scoreboard, and other necessary modules. The environment is responsible for creating and configuring these components and ensuring proper communication among them. Environment follows the same pattern as the agent does but environment calls the create method for agent, scoreboard and coverage using the factory registration methods in the build phase and also connects the monitor analysis port to the scoreboard and coverage analysis port through the hierarchical instantiation in connect phase.

Test: The testbench encapsulates the entire verification environment and orchestrates the execution of tests. It coordinates the setup, execution, and teardown of tests, as well as manages communication between various components of the testbench. The testbench is typically implemented as a subclass of “uvm_environment” or “uvm_component”. The two sequences are used to generate the tests for read and write separately. The actual raise objection and drop objection will be defined in this test run phase where the actual run phase begins in the uvm testbench hierarchy. Uvm_test calls the create function using the factory override method in the build phase.

Testbench Top: At the highest level of the hierarchy is the test. A test represents a specific verification scenario or test case that exercises certain functionalities or features of the DUT. Tests are usually defined as individual classes extending the “uvm_test” base class provided by UVM. This is the top level class in the entire uvm testbench hierarchy and no phases are defined in this class and only reset and clocks are generated and the DUT instantiation takes place in this class.

Coverage: UVM coverage refers to the metrics and measures used in the Universal Verification Methodology (UVM) to assess the completeness of verification efforts during digital design verification. The extract phase and report phase are defined in the coverage class which are the last phases as described below in the uvm phases where we display the coverage score and display them by also specifying the verbosity as well. The following code snippets shows the ways of defining the coverage.

UVM Phases:

S.NO	Type of Phase		
1.	Build Phase		
2.	Connect Phase		
3.	End Of Elaboration Phase		
4.	Start Of Simulation Phase		
5.	Run phase		
	Pre-Reset	Reset	Post- Reset
	Pre-Configure	Configure	Post-Configure

	Pre- Main	Main	Post - Main
	Pre-Shutdown	Shutdown	Post-Shutdown
6.	Extract Phase		
7.	Check Phase		
8.	Report Phase		
9.	Final Phase		

IMPLEMENTATION:

As per the project we considered one agent,two sequences,one sequencer,one sequence item,one scoreboard with coverage.

- The data_in is randomized and allowed to pass through the different components such as driver,monitor,scoreboard.
- Whenever write enable is asserted,the data will be written to the fifo memory and the burst details are displayed in each component.The write method which is called in the monitor is responsible for transferring the data to the scoreboard.
- Whenever read enable is asserted, the data will be read from the fifo memory and the burst details are displayed in each component.This will happen in the run phase of the scoreboard.