UVM_HIERARCHY

Top Level TB (tb_top):

Responsibilities:

- Instantiate the UVM testbench components.
- Connect the DUT to the testbench environment.
- Set up the simulation environment.
- Connect the DUT to the UVM components using appropriate interfaces.
- Control simulation flow.

Environment (env):

Responsibilities:

- Coordinate interactions between different agents.
- Manage global configuration and resources.
- Instantiate the UVM scoreboard and other analysis components.
- Manage and distribute test configuration.
- Instantiate and connect agents.
- Monitor the progress of the simulation.

Sequencer (fifo_sequencer):

Responsibilities:

- Generate sequences of transactions for the Asynchronous FIFO.
- Control the flow of transactions to the respective agents.
- Create sequences for various write and read scenarios.
- Control the rate and order of transactions.
- Interface with the sequencer to manage the transaction flow.

Sequence (fifo_sequence):

Responsibilities

- Develop the test sequences.
- (fifo_base_sequence.sv and fifo_test_sequence.sv) that focus on both the reset sequence and functional test scenarios.

Sequence Item (fifo_sequence_item):

Responsibilities:

- A fifo_sequence_item represents a single transaction in the testbench. It encapsulates the data and control information that will be passed between different components of the testbench.
- Store transaction-specific data such as write data, read data, and unique identifiers.
- Implement randomization for generating diverse transactions during simulation.
- Randomly set values for parameters like write data, read data, and unique identifiers.
- Ensure the sequence item is compatible with the DUT's interface requirements.
- Contain fields corresponding to the signals on the DUT's interface.

Monitor (fifo_monitor):
Responsibilities:

- Monitor the transactions on the FIFO interface.
- Collect data for scoreboarding and analysis.
- Observe and capture relevant signals from the FIFO interface.
- Extract data for subsequent scoreboarding.

Scoreboard (fifo_scoreboard):
Responsibilities:

- Compare expected results with the actual results.
- Raise objections or report errors when discrepancies are found.
- Receive data from the monitor.
- Compare monitored data with expected results.
- Raise objections or report errors for further investigation.

Driver (fifo_driver):
Responsibilities:

- Drive transactions onto the Asynchronous FIFO interface.
- Convert high-level transactions from the sequencer into low-level signals for the DUT.
- Translate transactions into signals compatible with the DUT interface.
- Drive these signals onto the FIFO interface.

Agent (fifo_agent):
Responsibilities:

- Combine the sequencer, driver, and monitor for a specific interface.
- Interface with the DUT.
- Instantiate the sequencer, driver, and monitor for a specific FIFO interface.
- Provide a unified interface to the DUT.

Implementation Approach:

1. Testbench Architecture:

- Top Module (async_fifo1_tb_uvm).
- Instantiates the UVM test fifo_test.
- Includes the asynchronous FIFO (async_fifo1) as the design under test (DUT).
- Connects the UVM test to the asynchronous FIFO via the async_fifo_if interface.

2. UVM Components:

2.1. UVM Test (fifo_test):

- File: test.sv
- Extends uvm_test.
- Instantiates the test environment (fifo_env).
- Configures and starts the required test sequences.

2.2. UVM Environment (fifo_env):

- File: env.sv
- Extends uvm_env.
- Instantiates and configures the FIFO agent (fifo_agent).
- Optionally, instantiates and configures the scoreboard (fifo_scoreboard).
- Manages the overall environment setup.

2.3. UVM Agent (fifo_agent):

- File: agent.sv
- Extends uvm_agent.
- Instantiates the driver (fifo_driver), monitor (fifo_monitor), and sequencer (fifo_sequencer).
- Connects the driver to the sequencer through the sequencer's seq_item_port.

2.4. UVM Driver (fifo_driver): • File: driver.sv

- Extends uvm_driver#(fifo_sequence_item).
- Drives transactions to the FIFO based on sequences generated by the sequencer.
- Implements the drive task to generate and drive transactions.
- Randomizes relevant fields in fifo_sequence_item when driving transactions.
- Monitors FIFO status signals and adjusts transaction generation accordingly.
  - Drives the following signals to the asynchronous FIFO:

    - wdata: Drives the write data.
    - winc: Drives the write increment signal.
    - rdata: Drives the read data.
    - rinc: Drives the read increment signal.
    - Monitors the status signals:
    - wfull: Monitors the write full status.
    - rempty: Monitors the read empty status.

2.5. UVM Monitor (fifo_monitor):

- File: monitor.sv
- Extends uvm_monitor.
- Observes signals from the FIFO and captures relevant data.
- Monitors signals such as wdata, winc, rdata, rinc, wfull, rempty, etc.
    - Monitors the following signals from the asynchronous FIFO:
    - wdata: Captures the write data.
    - winc: Captures the write increment signal.
    - rdata: Captures the read data.
    - rinc: Captures the read increment signal.
    - wfull: Captures the write full status.
    - rempty: Captures the read empty status.
- Optionally, captures other relevant signals for coverage and analysis.

2.6. UVM Sequencer (fifo_sequencer):

- File: sequencer.sv
- Extends uvm_sequencer#(fifo_sequence_item).
- Generates sequences of transactions based on the test scenarios.
- Controls the flow of transactions.

2.7. UVM Sequence Item (fifo_sequence_item):

- File: sequence_item.sv
- Extends uvm_sequence_item.
- Represents a transaction item with relevant fields (e.g., wdata, winc, rdata, rinc).
- May include a reset task to initialize the item's values.
- Randomizes the winc field to generate random write increment signals.
- Randomizes the wdata field to generate random write data.

3. Test Sequences

Reset Sequence (reset_seq)

- Verify the proper functionality of the asynchronous FIFO reset.
- Ensure the FIFO is in a known state after the reset.
- The reset sequence, reset_seq, focuses on testing the reset functionality of the asynchronous FIFO.
- Initiates a reset, observes the FIFO state, and ensures the FIFO is ready for subsequent transactions.

Functional Test Sequence (test_seq)

- Verify the basic functionality of the asynchronous FIFO.
- Perform read and write transactions with various data patterns.
- Check for correctness of data transactions.
- The functional test sequence, test_seq, aims to validate the fundamental operations of the asynchronous FIFO.
- Generates a mix of write and read transactions with different data patterns.
- Checks for correctness of data transactions, ensuring the FIFO behaves as expected.
- Monitors and captures relevant signals for analysis.

Execution Flow:

Initialization Phase:

Configure the test environment, agents, and DUT.
Build the UVM hierarchy.

Run Phase:

Generate and drive sequences to the DUT.
Monitor the DUT outputs.
Scoreboard verifies transactions.
Report errors and statistics.

Report Phase:

Display final results, including any errors detected.

Individual Contributions:

Mahidhar:

Responsible for the design and implementation of the monitor, test, and sequence items in the UVM testbench.
Tasks include creating the fifo_monitor.sv file, capturing relevant signals from the asynchronous FIFO, and implementing the monitoring logic. And focus on the test logic in the fifo_test.sv file,

ensuring that functional scenarios are covered, and the overall testbench behavior is verified. Prepared UVM_HIERARCHY documentation as well.

Ashwin:

Responsible for the implementation of the driver, environment, and sequence and sequencer components in the UVM testbench.
Tasks include creating the fifo_driver.sv file, which drives transactions to the asynchronous FIFO, monitors status signals, and adjusts transaction generation. And develop the fifo_env.sv file, which instantiates and configures the FIFO agent (fifo_agent) along with any other necessary components.
Additionally, worked on the fifo_sequencer.sv file, defining the sequencing logic and controlling the flow of transactions within the testbench. And develop the test sequences. (fifo_base_sequence.sv and fifo_test_sequence.sv) that focus on both the reset sequence and functional test scenarios.

Monica:

Responsible for the interface, testbench, and agent components in the UVM testbench. Tasks include creating the async_fifo_if.sv file, defining the interface for communication between the UVM testbench and the asynchronous FIFO.
Develop the async_fifo1_tb_uvm.sv file, which serves as the top module for the testbench, instantiating the UVM test and connecting it to the asynchronous FIFO.
Additionally, worked on the fifo_agent.sv file, which instantiates the driver, monitor, and sequencer, and facilitates their interconnections within the UVM environment.