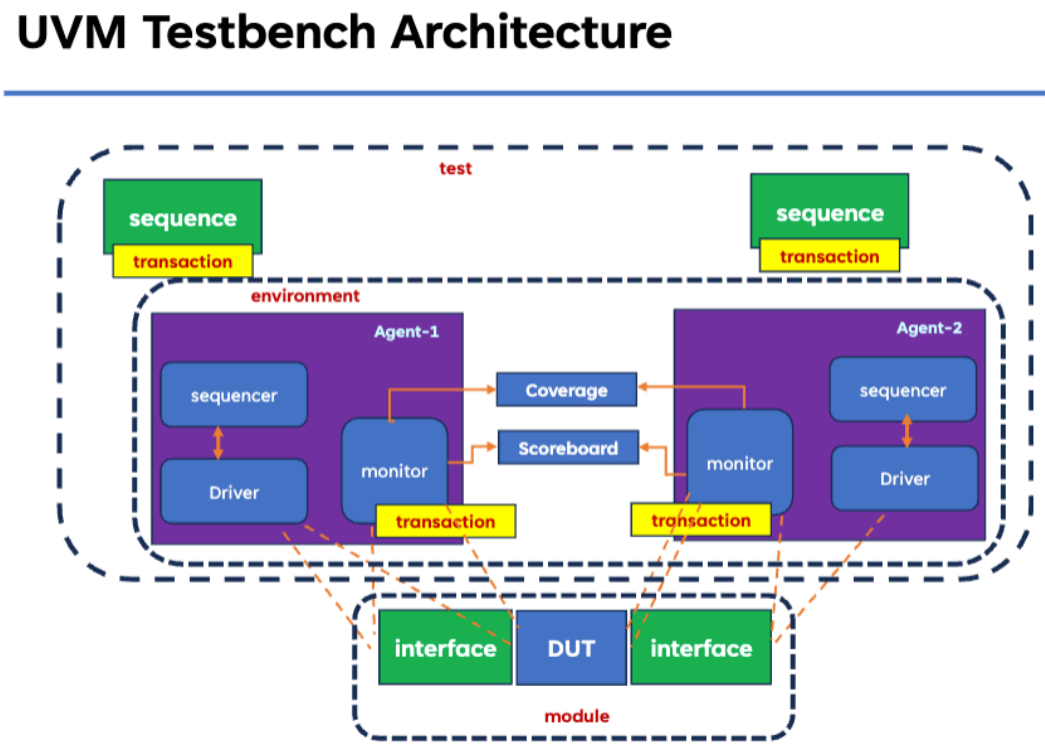
Team2: Hierarchy of UVM based testbench.

The hierarchy of UVM based testbench architecture for verifying an asynchronous FIFO design.



UVM Testbench Hierarchy

```
# Name                        Type                    Size  Value
# ------------------------------------------------------------------
# uvm_test_top                fifo_test               -     @471
#   env                       fifo_environment        -     @478
#     ragnt                   fifo_read_agent         -     @500
#       rdrv                  fifo_driver             -     @515
#         rsp_port            uvm_analysis_port       -     @530
#         seq_item_port       uvm_seq_item_pull_port  -     @522
#       rmon                  fifo_read_monitor       -     @538
#         monitor_port        uvm_analysis_port       -     @655
#       rseqr                 fifo_sequencer          -     @545
#         rsp_export          uvm_analysis_export     -     @552
#         seq_item_export     uvm_seq_item_pull_imp   -     @646
#         arbitration_queue   array                   0     -
#         lock_queue          array                   0     -
#         num_last_reqs       integral                32    'd1
#         num_last_rsps       integral                32    'd1
#     scb                     fifo_scoreboard         -     @507
#       scoreboard_port       uvm_analysis_imp        -     @669
#     wagnt                   fifo_write_agent        -     @493
#       wdrv                  fifo_driver             -     @678
#         rsp_port            uvm_analysis_port       -     @693
#         seq_item_port       uvm_seq_item_pull_port  -     @685
#       wmon                  fifo_write_monitor      -     @701
#         monitor_port        uvm_analysis_port       -     @818
#       wseqr                 fifo_sequencer          -     @708
#         rsp_export          uvm_analysis_export     -     @715
#         seq_item_export     uvm_seq_item_pull_imp   -     @809
#         arbitration_queue   array                   0     -
#         lock_queue          array                   0     -
#         num_last_reqs       integral                32    'd1
#         num_last_rsps       integral                32    'd1
# ------------------------------------------------------------------
```

1. Top-Level Testbench (`testbench_uvm.sv`):

- Serves as the entry point for simulation and testbench control.
- In this top-level testbench module, it connects the DUT and Verification environment components. It also imports our definitions package which will be used for both the UVM testbench and the DUT and also includes our interface.
- Adapts the simulation flow by invoking compilation, elaboration, and execution of test sequences.
- The testbench_uvm.sv file contains:
    - DUT interface
    - interface instance
    - run_test()method
    - Virtual interface set config_db
    - Clock generation for both the producer and consumer clocks

2. Testbench Modules:

a. Definitions (`definitions.sv`):

- The "definitions" package provides parameter definitions commonly used across the asynchronous FIFO design and testbench. It includes parameters, which specify the size and behavior of the FIFO. This package ensures consistency and ease of parameter management throughout the design and testbench components.

b. Sequence Item (`fifo_sequence_item.sv`):

- The " fifo_sequence_item" class contains the required fields that are used to generate the stimulus.
- Contains the data and control fields which are used to generate stimulus and represents the communication at the level of abstract.
- The sequence_item is written by extending uvm_seq_item

c. Sequence (`fifo_sequence.sv`):

- The "fifo_sequence.sv" class is what is used to generate the stimulus and sends the stimulus to the driver via the sequencer.
- (fifo_sequence, fifo_write_sequence, and fifo_read_seqeunce): It is a collection of sequence items where items are randomized and then sends to driver. fifo_sequence does the randomization in general, fifo_write_sequence does the randomization for the write, and fifo_read_sequence dpes the randomization for the read.
- The sequence is written by extending uvm_sequence

d. Sequencer (`fifo_sequencer.sv`):

- The sequencer class is written by extending uvm_sequencer

- It controls the request and response of the sequence item between driver and sequence.

e. Driver(`fifo_driver.sv`):

- The driver class is written by extending uvm_driver
- The driver receives the stimulus from the sequence from the sequencer and drives the connected interface signals.
- The driver class serves as the component responsible for driving stimuli to the asynchronous FIFO DUT (Design Under Test).
- Declares the interface signals required for communication between the testbench and the DUT. This interface facilitates seamless communication and interaction between the testbench and the DUT during verification.
- The driving logic uses the connected sequence item to drive the connected interface DUT signals.

f. Monitor (`fifo_write_monitor.sv`) and (`fifo_write_monitor.sv`):

- The monitor class is written by extending uvm_driver
- The monitor samples the connected DUT signals from the interface and converts the signal level activity to a transaction level.
- It manages the generation of write and read transactions, ensuring proper synchronization with the FIFO's clock domains. The class samples the interface signals in a forever loop and uses either the write or read monitors to send the sampled data to the scoreboard.
- The monitor class serves as a passive observer for the asynchronous FIFO. It samples the interface signals of the FIFO design under test and constructs transaction objects representing the observed behavior. These transaction objects are then sent to the scoreboard for analysis via a port mechanism. The monitor operates continuously, sampling both the write and read interface signals, ensuring comprehensive monitoring of the FIFO's behavior during simulation.

g. Agent (`fifo_write_agent.sv`) and (`fifo_write_agent.sv`):

- The agent class is written by extending uvm_agent
- The agent is a container class that houses the driver, sequencer, and monitor.
- The instances of the driver, sequencer, and monitor are all declared in the agent class.
- Our agent class is active so all the components that were declared earlier are created in the build phase.

h. Scoreboard (`fifo_scoreboard.sv`):

- The scoreboard class is written by extending uvm_scoreboard
- The scoreboard receives the transaction from the monitor and compares it with the reference values.

- A TLM analysis port is declared and created in the scoreboard to receive the information from the monitor.
- The analysis port of the scoreboard is connected to the monitor port.
- The scoreboard class functions as the verification component for the asynchronous FIFO. It receives transaction objects representing observed behavior from the monitor via port mechanism. The scoreboard then compares the received data with the expected data stored in memory. If there is a mismatch, it reports an error, indicating the inconsistency between the expected and actual results. Additionally, it keeps track of the number of transactions processed for monitoring purposes.

h. Environment (`fifo_environment.sv`):

- The environment class is written by extending uvm_env.
- The environment class is the container class. It contains both the write and read agents, along with the score board, and the write and read monitors.

i. Test (`fifo_test.sv`):

- The test class is written by extending uvm_test.
- The test class is what defines the test scenario for the testbench.
- The test class contains the end of elaboration phase which prints out the topology of out design and also contains a report phase which will print out the test results.

Source for UVM Verification Plan:
https://verificationguide.com/uvm/uvm-testbench-architecture/#TestBench_Top