UVM Testbench Hierarchy Overview

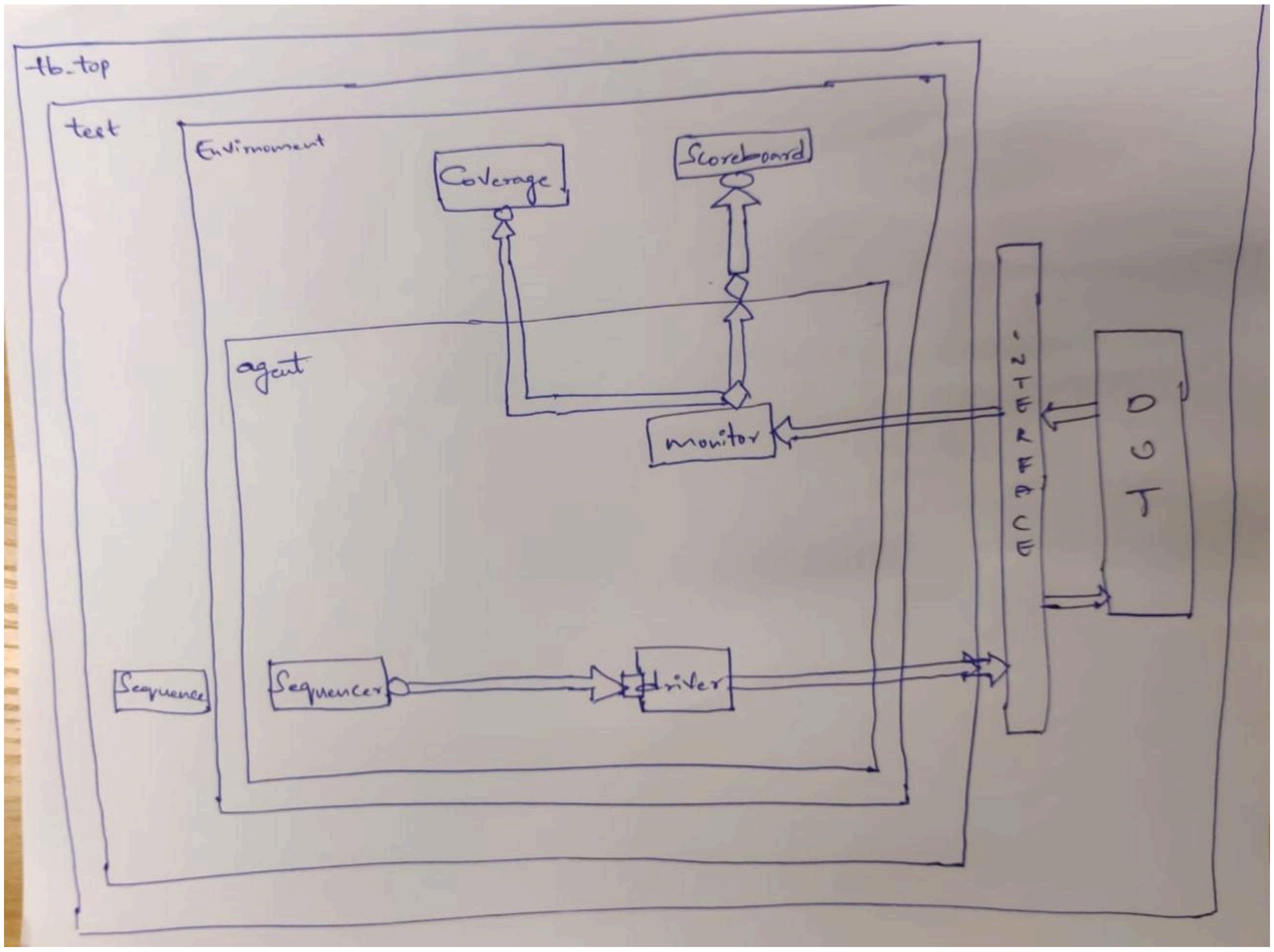
Project Progress:

The team was not able to complete the project by milestone 5 due to some bugs that were difficult to debug. Specifically, we had problems with the FULL and EMPTY flag. During the class based TB and UVM TB, both these flags were not correct, and we weren't able to find the cause until 2/27 (due date). The problem is described in detail in the verification plan document, but basically the write and read pointers were wrapped before reaching the last entry in the array. Due to this bug, we weren't really able to pass most of our test cases, but we did pass basic ones such as writes followed by reads, FIFO order, etc. Also, the code and functional coverage are both close to 100%. So hopefully the team will be able to figure out this bug before 6/03 and implement a fix. On the bright side, our bug injection RTL code is ready.

Verification Strategy:

Verifying an asynchronous FIFO is challenging due to the involvement of two independent clock domains: one for writing and another for reading. The key goal of this milestone is to develop a test environment using the UVM (Universal Verification Methodology) framework. The verification strategy focuses on rigorously testing the FIFO's behavior during asynchronous data writing, storage, and reading. Additionally, it ensures proper handling of edge cases and error conditions, confirming the design's robustness and reliability.

Testbench Architecture; Component used (list and describe Drivers, Monitors, scoreboards, checkers etc.)



Testbench architecture used will include:

- Testbench Top
    - o Highest level of the testbench Hierarchy
    - o Contains Environment instantiation and configuration
    - o instantiates the DUT and the test sequence
- Environment
    - o Contains the agent and components for driving stimulus and checking DUT responses
    - o Instantiates the agent
- Agent
    - o Responsible for interfacing with DUT interface
    - o Contains components to drive stimulus and collect responses
        - ▪ Agent Sequencer
            - Generates sequence of transactions
        - ▪ Agent Driver
            - Drives the stimulus to the DUT
        - ▪ Agent Monitor

- Monitors signals on the interface of the DUT
- Collects data for analysis and scoreboard

- Scoreboard
  - o Compares expected results with actual results from DUT
  - o Raises error flags on data mismatch
- Sequences
  - o Contains sequence of transactions that represent specific test scenarios o Sequence Item represents a single traction
    - ▪ Contains information for driving and checking
    - ▪ Each test contains separate sequence

Gray Code : Using gray code is a standard technique to ensure reliable and error-free data transfer between asynchronous domains. This minimizes the risk of metastability and sampling errors, thereby enhancing the reliability of the FIFO operation.

Write Pointer : A binary write pointer keeps track of the location where the next data should be written.. The write pointer, which operates in the write clock domain, is converted to Gray code before being passed to the read clock domain. This ensures that the read logic gets a stable and correct pointer value

Read Pointer : A binary read pointer keeps track of the location from where the next data should be read. The read pointer, which operates in the read clock domain, is converted to Gray code before being passed to the write clock domain

Memory Array : The FIFO buffer uses a memory array to store data. Data is written to the memory in the write clock domain and read from the memory array in the read clock domain.

Implementation: Expected Data Flow

- **Testbench Top-Level (TB Top)**
  Begins the testing process by triggering the test sequence.

- **Environment Setup**
  Responsible for creating and setting up the agent with necessary configurations.

- **Agent Responsibilities**

  - Produces and manages test sequences.

  - Sends transaction data to the DUT (Design Under Test) interface.

    - Observes and records the DUT's output behavior.

- **Scoreboard Function**
  Compares the input data with the DUT's output and reports any mismatches or unexpected behavior

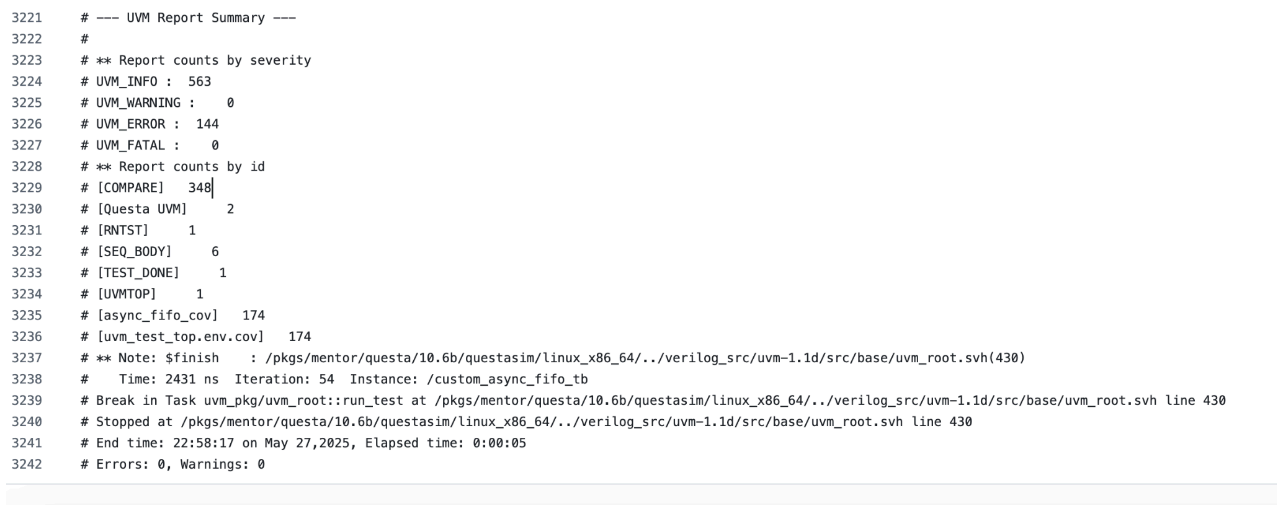
Test Scenarios:

TEST CASES	DESCRIPTION
FIFO Reset	The reset functionality in the driver is used to verify whether the FIFO returns to its default or initial state.
FIFO Full	Data is written into the FIFO until it reaches maximum capacity, then it is verified whether the 'full' flag gets asserted.
FIFO Empty	It is verified that the 'empty' flag gets asserted when no new data is written to the FIFO and the existing data is completely read.

FIFO Depth Calculation:
Double the frequency of write and half of the read
120 MHz is the sender clock frequency.
Three idle cycles separate two consecutive writes, and the receiver clock frequency is 50 MHz.
There are two idle cycles in between two consecutive reads.

Given that the sender clock frequency is less than the receiver clock frequency, write burst = 1024.
Three clock cycles separate two consecutive writes, indicating that after writing one piece of data, the
write module waits three clock cycles before starting the next write. This means that one piece of data is
written every four clock cycles.
Writing one data item takes 4 * (1/120MHz) = 33.33 ns.
It took 34,129.92 = 34,129 ns to write the data in the burst.
One data item's reading time is equal to 3 * (1/50MHz) = 60 ns.
Thus, the read module will read one data item from the burst every 60 ns.
In 34,129 ns, the number of data pieces that can be read is 34,129/60 = 568.81 = 568 items. 1024 – 568 = 456 is the remaining number of bytes that must be kept in the FIFO.

Transcript:



```
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :  563
# UVM_WARNING :    0
# UVM_ERROR :  144
# UVM_FATAL :    0
# ** Report counts by id
# [COMPARE]   348
# [Questa UVM]     2
# [RNTST]     1
# [SEQ_BODY]     6
# [TEST_DONE]     1
# [UVMTOP]     1
# [async_fifo_cov]   174
# [uvm_test_top.env.cov]   174
# ** Note: $finish    : /pkgs/mentor/questa/10.6b/questasim/linux_x86_64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 2431 ns  Iteration: 54  Instance: /custom_async_fifo_tb
# Break in Task uvm_pkg/uvm_root::run_test at /pkgs/mentor/questa/10.6b/questasim/linux_x86_64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
# Stopped at /pkgs/mentor/questa/10.6b/questasim/linux_x86_64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh line 430
# End time: 22:58:17 on May 27,2025, Elapsed time: 0:00:05
# Errors: 0, Warnings: 0
```

Transcript attached in zip file.

References:

1. Professor examples from slides
2. Verification academy