ECE-593: Fundamentals of Pre-Silicon Validation

Maseeh College of Engineering and Computer Science

Winter, 2026

Project Name: Design, Implementation, and Verification of Asynchronous FIFO

Members: Monesh Kareti, Senkathir Mutharasu

Date:01/28/26

GITHUB: https://github.com/senkathir2/ECE593-Async_FIFO_G10

VERIFICATION TEST PLAN

The objective of the verification:

The primary goal of this verification plan is to comprehensively validate the design's functionality and accuracy using various verification techniques, while also implementing error detection and correction mechanisms.

Division of tasks:

Team members and task allocation:

• **Monesh Kareti** is responsible for the data entry architecture and core storage logic. His primary RTL focus includes implementing the Write Control Logic, the FIFO Memory Array, and managing the write-side pointer synchronization. He will also ensure the design meets specific timing requirements, such as the two-cycle write idle specification and overflow protection. In the verification phase, he will develop the Write-side Agent (including the driver and monitor) and the Generator modules. Additionally, he will lead the documentation of the Design Specification, ensuring the write-path functionality is fully defined and tested.

• **Senkathir Mutharasu** is responsible for the data exit logic and the overall verification infrastructure. His RTL tasks involve implementing the Read Control Logic, managing the read-side pointer synchronization, and ensuring the one-cycle read idle requirement is met. He also handles the generation of status flags like Empty and Full. On the verification side, he will build the UVM Environment and Top-Level Testbench, while developing the Read-side Agent and the Scoreboard for data validation. Finally, he will draft the Verification Plan and ensure the seamless integration of all design modules within the testbench.

Functional Verification:

The functional verification of the asynchronous FIFO focuses on checking correct data transfer and flag behavior under both normal and boundary conditions. All write and read operations are verified to ensure that data ordering, integrity, and flow control are maintained across the two clock domains. The verification also confirms that the full and empty flags, as well as overflow/underflow handling and concurrent write-read operations, behave as specified. Special emphasis is placed on the synchronization logic between clock domains so that metastability does not cause functional errors.

The main methods used for functional verification are based on UVM (Universal Verification Methodology) for creating a reusable and scalable testbench. A UVM environment is developed with sequences that generate directed and random write/read traffic patterns. SystemVerilog Assertions (SVA) are written to check key properties such as “no write when full” and “no read when empty,” as well as pointer and flag consistency. All tests are simulated using an industry-standard simulator such as ModelSim, QuestaSim, or VCS to obtain waveforms, logs, and coverage data for analysis.

Edge Case Testing:

Edge case testing targets the critical boundary conditions of the FIFO to ensure robust behavior in extreme scenarios. These tests complement normal functional testing by focusing on situations where the FIFO is full, empty, or close to those limits, as well as on illegal operations.

- FIFO Full
  In this test, data is written continuously until the FIFO reaches its maximum capacity. The expected outcome is that the full flag is asserted exactly at the full condition and that any further write attempts are either blocked or flagged as errors, with no data corruption.

- FIFO Empty
  In this test, data is read out until the FIFO becomes completely empty.
  The empty flag is expected to assert when no valid data remains, and further

read attempts should be blocked or cause an appropriate error indication without causing underflow.

- Full Error
  This test attempts to write data when the FIFO is already full. The correct behavior is that the design raises an error or keeps the full flag asserted and does not accept new data, thereby blocking the write and preserving the FIFO contents.

- Empty Error
  This test attempts to read data when the FIFO is empty. The expected result is that an error is raised or the empty flag remains asserted and no new valid data is produced on the output, so the read operation is effectively blocked.

- Concurrent Write-Read
  This test performs write and read operations at the same time to stress the concurrent behavior of the FIFO. The verification checks that data is neither lost nor corrupted, that ordering is maintained, and that synchronization between the two clock domains is correct under simultaneous access.

Coverage Metrics:

Coverage metrics are used to measure how thoroughly the FIFO has been verified and to guide additional test development. Functional coverage focuses on ensuring that all important FIFO states and transitions are exercised, including full, empty, partially filled levels, and various read/write scenarios. This includes coverage of flag transitions, occupancy ranges, and combinations of operations such as write-only, read-only, and concurrent access.

Code coverage is used to confirm that all lines, branches, and conditions in the FIFO RTL code are executed during simulation. High code coverage indicates that the tests are not leaving large portions of the implementation untested. Assertion coverage measures whether each SystemVerilog assertion has been activated under the intended conditions. Together, functional, code, and assertion coverage provide a quantitative view of verification completeness and highlight any gaps that require additional testcases.

Verification Environment

The verification environment is built using a UVM-based testbench to support modularity, reuse, and scalability. A UVM driver is responsible for sending write and read transactions to the FIFO, translating high-level sequence items into pin-level stimulus on the interface. A UVM monitor observes the FIFO ports and status flags, collecting transactions and events for checking and coverage.

A scoreboard compares the data observed at the read side against a reference model (for example, a SystemVerilog queue) to confirm data correctness and ordering. A dedicated coverage collector records functional coverage information such as flag states, occupancy levels, and operation types. Simulations are run on tools such as ModelSim, QuestaSim, or VCS, which also provide code and assertion coverage reports. SystemVerilog assertions (SVA) are integrated into the environment to continuously check properties related to full and empty conditions, protocol rules, and concurrent operations throughout all test runs.

Week	Dates	Planned Activities	Deliverables
Week 1	**Jan 29 – Feb 4**	Finalize asynchronous FIFO design specifications and FIFO depth calculations. Prepare Initial Verification Plan. Implement initial RTL using SystemVerilog constructs. Develop a simple conventional testbench to verify basic read/write functionality and data flow.	Design Specification document, Initial Verification Plan, Compiling RTL, Basic functional testbench
Week 2	**Feb 5 – Feb 11**	Complete and stabilize FIFO RTL. Begin development of class-based verification environment. Define transaction and generator classes. Implement driver and monitor components. Perform early simulations and debugging.	Stable RTL, Partial class-based verification environment
Week 3	**Feb 12 – Feb 18**	Complete class-based testbench (Transaction, Generator, Driver, Monitors, Scoreboard, Coverage). Add transcript display messages. Execute randomized tests. Generate functional and code coverage reports. Update the Verification Plan with detailed test cases. Resolve warnings/errors.	Complete class-based TB, Coverage reports, Updated Verification Plan, and compile-clean simulation
Week 4	**Feb 19 – Feb 25**	Develop UVM testbench architecture (environment, agents, sequencer, driver, monitor, scoreboard). Integrate RTL with UVM TB. Add UVM messaging and logging. Run basic UVM simulations.	Initial UVM testbench architecture, UVM simulation logs
Week 5	**Feb 26 – Mar 4**	Complete UVM environment and testbench. Add the UVM verification plan section, including architecture and hierarchy details. Implement and validate testcases. Prepare for the final verification phase.	Complete UVM TB, Updated Verification Plan (UVM section)
Week 6	**Mar 5 – Mar 14**	Execute all final testcases and achieve the required coverage. Perform bug-injection scenarios and verification. Finalize design specification, verification plan, coverage reports, and presentation materials. Prepare simulation run instructions and final submission package.	Final RTL & UVM TB, Coverage reports, Bug-injection results, Final documentation & presentation