VERIFICATION TEST PLAN

ECE-593: Fundamentals of Pre-Silicon Validation
Maseeh College of Engineering and Computer Science
Winter, 2025

Project Name: Verification of Asynchronous FIFO

Members: Satyajit Samhaji Deokar,
Siddesh Dinesh Patil,
Aakash Siddharth Bhupal Hanumanthrao,
Sai Ganesh Reddy Charian

Date: 1/31/2025

1 Table of Contents

2 Introduction:....4

2.1 Objective of the verification plan....4

2.2 Top Level block diagram....4

2.3 Specifications for the design....4

3 Verification Requirements....4

3.1 Verification Levels....4

3.1.1 What hierarchy level are you verifying and why?....4

3.1.2 How is the controllability and observability at the level you are verifying?....4

3.1.3 Are the interfaces and specifications clearly defined at the level you are verifying. List them. 4

4 Required Tools....4

4.1 List of required software and hardware toolsets needed....4

4.2 Directory structure of your runs, what computer resources you will be using....4

5 Risks and Dependencies....4

5.1 List all the critical threats or any known risks. List contingency and mitigation plans....4

6 Functions to be Verified....4

6.1 Functions from specification and implementation....4

6.1.1 List of functions that will be verified. Description of each function....4

6.1.2 List of functions that will not be verified. Description of each function and why it will not be verified....4

6.1.3 List of critical functions and non-critical functions for tapeout....4

7 Tests and Methods....4

7.1.1 Testing methods to be used: Black/White/Gray Box....4

7.1.2 State the PROs and CONs for each and why you selected the method for this DUV....4

7.1.3 Testbench Architecture; Component used (list and describe Drivers, Monitors, scoreboards, checkers etc.)....4

7.1.4 Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.) Describe why you chose the strategy....4

7.1.5 What is your driving methodology?....4

7.1.6 What will be your checking methodology?....4

7.1.7 Testcase Scenarios (Matrix)....4

8 Coverage Requirements....4

8.1.2 Assertions....4

9 Resources requirements....4

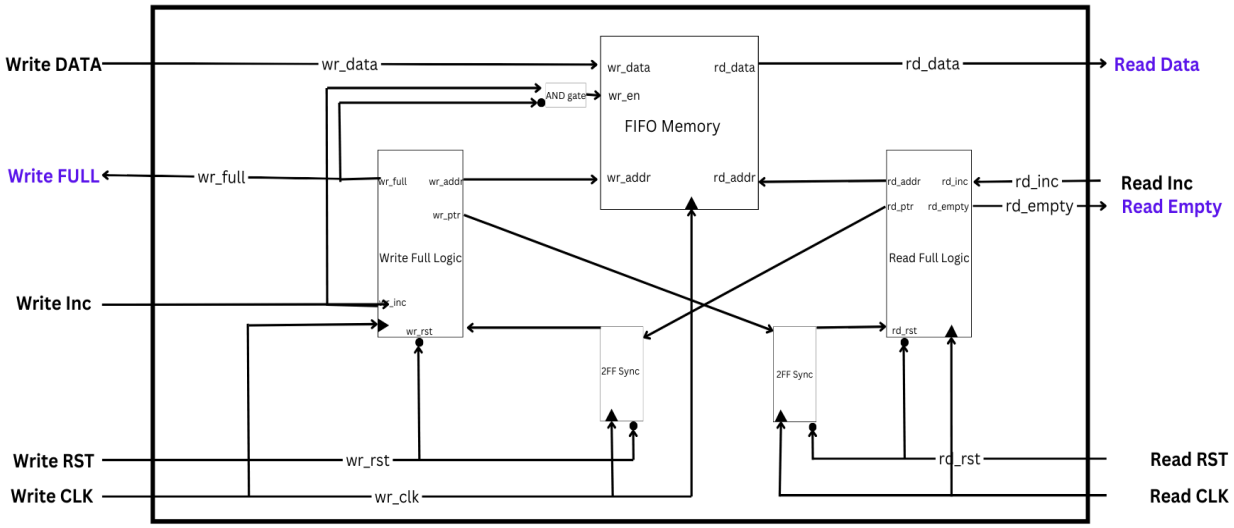
9.1 Team members and who is doing what and expertise....4
10 Schedule....4
10.1 Create a table with plan of completion. You can use the milestones as a guide to fill this....4
11 References Uses / Citations/Acknowledgements....4

2 Introduction:

2.1 Objective of the verification plan:

- Module Level Verification (FIFO functionality)
- System Level Verification (Integration with larger system)

2.2 Top Level block diagram



2.3 Specifications for the design

Design Features	
1	Theoretically unbounded, but in practice, limited by the simulator and hardware constraints supports data widths upto [2^64 -1] bits; Parameterized by 'DATASIZE'.
2	Theoretically unbounded, but in practice, limited by the simulator and hardware constraints supports memory depths upto [2^64 -1] bits; Parameterized by 'DEPTH'
3	Fully synchronous, and independent clock domains for the Read and Write ports
4	Supports FULL and EMPTY status flags
5	Two optional output signals for indicating Write-Full and Read-Empty of the memory port.

3 Verification Requirements

3.1 Verification Levels

- Module Level Verification (FIFO functionality)
- System Level Verification (Integration with larger system)

3.1.1 What hierarchy level are you verifying and why?

3.1.2 How is the controllability and observability at the level you are verifying?

- Write and read transactions will be controlled via UVM driver.
- FIFO status flags will be observed via UVM monitor and scoreboard.
- Coverage metrics will be collected using functional coverage and assertion-based verification.
- Error injection techniques will be employed to evaluate FIFO behavior under corner-case scenarios.

3.1.3 Are the interfaces and specifications clearly defined at the level you are verifying. List them.

- Write and Read Data Bus
- Clock and Reset signals
- Status flags (Full, Empty, Almost Full, Almost Empty)

4 Required Tools

4.1 List of required software and hardware tool sets needed.

- Simulation Tools: Mentor Questa
- Debugging Tools:Wave, Sim

4.2 Directory structure of your runs, what computer resources you will be using.

- Source Files Directory
- Testbench Directory
- Logs and Reports Directory

5 Risks and Dependencies

5.1 List all the critical threats or any known risks. List contingency and mitigation plans.

- Metastability concerns due to asynchronous clock domains
- Potential deadlocks or race conditions
- Incorrect flag generation causing data loss

6 Functions to be Verified.

6.1 Functions from specification and implementation

- Data write and read operations

- Handling of FIFO full and empty conditions
- Synchronous and asynchronous reset behavior

6.1.1 List of functions that will be verified. Description of each function

- Functions Verified: Carried out extensive checks on fundamental FIFO operations like read/write actions, flag management (empty/full), and boundary condition handling, ensuring reliability across different scenarios.
- Testing Approach: Performed in-depth testing to verify correct functionality under various conditions, including edge cases and concurrent read/write activities, employing detailed testbenches to cover all possible scenarios.

6.1.2 List of functions that will not be verified. Description of each function and why it will not be verified.

Functions Not Verified:

- Non-critical Operations: Debugging, logging, and extra performance features that don't influence the core FIFO functionality.
- Design-specific Features: Optional features or unused control signals that aren't required for the FIFO's operation.
- Reason for Non-verification: These functions don't affect the core FIFO operations and are outside the scope of this project, allowing the focus to remain on verifying the essential FIFO functionality.

6.1.3 List of critical functions and non-critical functions for tapeout

- Critical: Data integrity, flag behavior, metastability handling
- Non-Critical: Performance optimizations

7 Tests and Methods

7.1.1 Testing methods to be used: Black/White/Gray Box.

- Black Box: Validate FIFO as a standalone unit.
- White Box: Check internal register states.
- Gray Box: Combination of both approaches.

7.1.2 State the PROs and CONs for each and why you selected the method for this DUV.

Black Box Testing

- PROs: Assesses FIFO functionality based solely on inputs and outputs, simulating realistic scenarios without needing to understand the internal design.
- CONs: Limited insight into internal states may result in missing certain edge cases tied to the internal logic.
- Reason: Best for validating the FIFO's overall operation, such as ensuring proper read/write handling and correct flag behavior.

White Box Testing

- PROs: Provides visibility into the internal logic, verifying internal states, transitions, and boundary conditions to identify implementation flaws.
- CONs: Requires detailed access to the design and knowledge of its internals, which can be time-consuming.

- Reason: Essential for verifying the accuracy of internal states and flag management, including state transitions and register behavior.

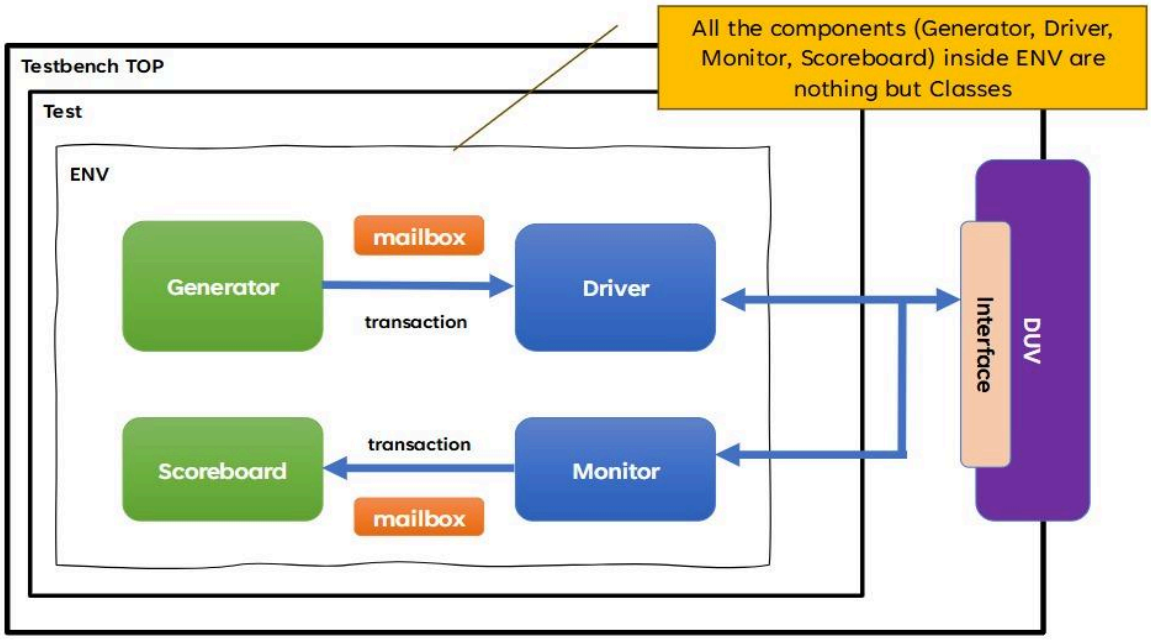
Gray Box Testing

- PROs: Merges the strengths of Black and White Box testing, offering a holistic view of both functionality and internal design.
- CONs: More complex and demanding, requiring a balance of knowledge of internal logic and external functionality.
- Reason: Provides thorough validation by combining both approaches, ensuring comprehensive testing of the FIFO design.

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

- Driver: Generates write/read transactions.
- Monitor: Captures transactions for validation.
- Scoreboard: Compares expected vs. actual outputs.

Include general testbench architecture diagram and how it relates to your design



SystemVerillog testbench architecture

7.1.4 Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.) Describe why you chose the strategy.

- Dynamic Simulation: Functional correctness verification.
- Formal Verification: Ensure no corner cases are missed.

7.1.5 What is your driving methodology?

- Robust Design Approach: Utilized a modular architecture to enhance FIFO reliability across asynchronous clock domains, focusing on minimizing metastability and ensuring seamless data transfer.
- Effective Synchronization: Employed Gray code pointers and dual-stage synchronizers to facilitate safe and efficient clock domain crossings.
- Test-Driven Verification: Adopted a verification-first strategy, developing SystemVerilog/UVM testbenches to validate functionality, performance, and corner cases early in the design phase.
- Continuous Optimization: Refined FIFO architecture iteratively through performance assessments, CDC verification, and timing analysis to maximize operating frequency and reduce latency.
- Handling Critical Scenarios: Anticipated and resolved key edge cases, including simultaneous read/write operations, near-full and near-empty states, and clock variations, to ensure system robustness.
- Comprehensive Documentation & Review: Maintain thorough design documentation and actively participate in design reviews to align with industry best practices and project timelines.

7.1.5.1 List the test generation methods (Directed test, constrained random)

- Directed Testing: Specific scenarios (reset conditions, flag checks).
- Constrained Random Testing: Randomized read/write sequences.

7.1.6 What will be your checking methodology?

- Synchronization Verification: Ensured the correctness of Gray code pointer transitions and dual-stage synchronizers for reliable clock domain crossings.
- Metastability Assessment: Conducted timing analysis, including setup and hold checks, to minimize the risk of metastability issues.
- Functional Evaluation: Designed SystemVerilog/UVM testbenches to rigorously test FIFO operations, such as read/write accuracy, flag transitions, and reset behaviors.
- Clock Domain Crossing (CDC) Validation: Utilized formal verification and static timing analysis to identify and resolve CDC-related challenges.
- Extreme Condition Testing – Simulated and examined FIFO responses under challenging scenarios like concurrent read/write operations, near-full or near-empty states, and fluctuating clock signals.
- Debugging and Design Review – Conducted comprehensive design reviews, waveform inspections, and debugging processes to enhance reliability and meet project specifications.

7.1.6.1 From specification, from implementation, from context, from architecture etc

- From Specification: Verified that the FIFO meets the required data width, depth, and latency constraints as defined in the project specifications. Ensured compliance with power, speed, and area limitations to align with design requirements.
- From Implementation: Conducted functional simulations using SystemVerilog/UVM to validate read/write operations, flag transitions, and reset behavior. Performed linting and synthesis checks to confirm that the design meets logical and structural requirements.
- From Context: Assessed FIFO behavior when integrated into the larger system, ensuring smooth interaction with upstream and downstream components. Verified clock domain crossing (CDC) mechanisms to prevent data corruption and ensure reliable operation across different clock domains.
- From Architecture: Analyzed the FIFO structure, buffer management, and pointer synchronization to ensure optimal performance and efficiency. Ensured that Gray code-based synchronization techniques were implemented correctly to prevent metastability issues.
- From Timing Analysis: Conducted setup and hold time verification to minimize metastability risks during asynchronous clock transitions. Performed static timing analysis (STA) to check for timing violations and optimize operating frequency.
- From Verification: Used formal verification and constrained-random testing to validate FIFO performance under various scenarios. Examined critical edge cases, including simultaneous read/write operations, near-full and near-empty states, and unexpected clock variations.
- From Debugging and Review: Analyzed simulation waveforms to debug functional issues and ensure correct FIFO operation. Conducted peer design reviews and static code analysis to enhance design robustness and maintain quality standards.

7.1.7 Testcase Scenarios (Matrix)

7.1.7.1 Basic Tests

Test Name / Number	Test Description/ Features
1.1.1 Read	Check basic read operation
1.1.2 Write - Read	Ensure correct data transfer
1.1.3 Reset	Check reset behavior

7.1.7.2 Complex Tests

Test Name / Number	Test Description/ Features
1.2.1 Concurrent Read-Write and Boundary Condition Test	Concurrent events (R+W) Conditions: fifo_full/fifo_empty/always_full/always empty etc.
1.2.2 Simultaneous Read-Write Operation Test	Validate simultaneous read/write
1.2.3 Metastability	Cross-clock domain checks

7.1.7.3 Regression Tests (Must pass every time)

Test Name / Number	Test Description/Features
1.3.1 Baseline Functionality Test	Tests that should always pass
1.3.2 Full/Empty Handling	Ensure FIFO doesn't overflow or underflow

7.1.7.4 Any special or corner cases testcases

Test Name / Number	Test Description
1.4.1 Special Case and Edge Condition Test	Special Case testing tests and conditions
1.4.2 Bug Injection and Fault Tolerance Test	Bug injection and testing scenario

8 Coverage Requirements

8.1.1.1 Describe Code and Functional Coverage goals for the DUV

- Code Coverage Goals: Ensure that every line of code is executed at least once (statement coverage) and all decision branches are tested (branch coverage).
  - Achieve path coverage by testing all possible execution paths in the design to ensure no paths are left untested.
  - Track signal transitions (toggle coverage) to ensure that signals change states correctly during testing.
- Functional Coverage Goals: Verify that the FIFO design behaves as expected in key scenarios, including read/write operations and flag behaviors such as empty/full conditions.
  - Cover corner cases like simultaneous read/write operations, FIFO overflow/underflow, and ensure that the FIFO operates within performance limits like latency and throughput.

8.1.1.2 Formulate conditions of how you will achieve the goals. Explain the Covergroups and Coverpoints and your selection of bins.

- Directed testing will be used to validate specific edge cases, like full/empty conditions and boundary states, by focusing on targeted functionality.
- Random testing will help achieve broader coverage by simulating unexpected conditions and scenarios to ensure robustness.
- Covergroups will be defined to track critical signals such as write-enable, read-enable, and FIFO full/empty flags during test execution.
- Bins will be used within the covergroups to monitor specific signal values and states, like write attempts during full FIFO conditions and near-empty/near-full states.

8.1.2 Assertions

- State and Flag Validation: Assertions ensure proper FIFO operation by validating state transitions, such as moving from full to empty only after a read operation. They also check that full/empty flags are asserted correctly—full when the FIFO reaches capacity, and empty when it contains no data.
- Error Detection and Consistency: These assertions monitor FIFO behavior during read/write operations, flagging errors when the state or flags are inconsistent. This includes detecting issues like an incorrect empty flag being set when there's still data in the FIFO or unexpected state transitions, improving the reliability of the design.

8.1.2.1 Describe the assertions that you are planning to use and how it will help you improve the overall coverage and functional aspects of the design.

- Functional Assertions: Monitor FIFO empty/full flags to ensure they are correctly asserted when the FIFO is empty or full. Verify that read and write operations happen only under valid conditions (e.g., no read from empty FIFO, no write to full FIFO).
- Boundary Condition Assertions: Detect FIFO overflow or underflow by asserting when invalid operations occur, such as reading when empty or writing when full. Ensure data integrity by asserting that data written to the FIFO is accurately read out.
- Increased Coverage: Assertions enhance both functional and code coverage, ensuring critical conditions like simultaneous operations and boundary states are properly checked during simulation.
- Early Bug Detection: Assertions immediately flag design violations, reducing debugging time by quickly identifying errors during simulation and ensuring design specifications are met.
- Improved Functional Verification: Assertions automate the enforcement of design constraints and specifications, ensuring reliable FIFO operation under all conditions, such as correct flag handling and operational correctness.

9 Resources requirements

9.1 Team members and who is doing what and expertise.

- Aakash Siddharth - Design debugging and Verification
- Satyajit Deokar - RTL Design and verification
- Siddesh Patil - RTL Design and verification
- Ganesh Reddy - Synchronising in design and Assertions in Verification

10 Schedule

10.1 Create a table with a plan of completion. You can use milestones as a guide to fill this.

Milestone	Expected Completion
Design and Conventional Testbench	Week 0
Class based Verification using SV	Week 1
Running Random Test Cases	Week 2
Adding Functional Coverage	Week 3
Adding Assertions and Running Final Results	Week 4
Class based Verification using UVM	Week 5

11 References Uses / Citations/Acknowledgements

- http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf
- https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/