VERIFICATION TEST PLAN

(TEMPLATE)

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

Project Name:
Team12_AsynchronousFIFO_S25_ECE593
Members:

Aravindh Nanjaiya Latha (aravindh@pdx.edu)

Rishi Gunda (rgunda@pdx.edu)

Rohit Bonigala (rohit@pdx.edu)

Shreya Umesh Shetty(shreyau@pdx.edu)
Date: 05/20/2025

1 Table of Contents

2 Introduction:

2.1 Objective of the verification plan: To verify the correct operation of an asynchronous FIFO design that transfers data between two different clock domains. The goal is to ensure data integrity, robustness against metastability, and proper implementation of control signals.

2.2 Top Level block diagram

2.3 Specifications for the design:

FIFO Depth: 16 entries

Data Width: 8 bits

Asynchronous write and read clocks

Full and empty flags

Support for simultaneous read/write in different domains

3 Verification Requirements

3.1 Verification Levels

3.1.1 What hierarchy level are you verifying and why?

Verifying at the RTL level to ensure functional correctness before synthesis. Verifying at block and top levels.

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

High controllability through dedicated testbench signals; observability through monitoring internal FIFO states via debug outputs.

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

- Write Interface: wr_clk, wr_en, wr_data, full
- Read Interface: rd_clk, rd_en, rd_data, empty

4 Required Tools

4.1 List of required software and hardware toolsets needed.

Simulator: ModelSim/QuestaSim or Icarus Verilog

Language: SystemVerilog

Git for version control

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

/src	- SystemVerilog source code
/tb	- Testbench
/logs	- Simulation outputs
/docs	- Reports and plans

5 Risks and Dependencies

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

- Clock domain crossing errors (mitigation: double flop synchronizers)
    - Incorrect full/empty logic (solution: assertions and corner-case tests)
    - Tool mismatch or version issues (use common simulator like ModelSim)

6 Functions to be Verified.

6.1 Functions from specification and implementation

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

- Write data into FIFO
- Read data from FIFO
- Full and empty flag logic
- Pointer wrap-around
- Synchronization between clock domains

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

- Dynamic depth scaling (not in scope)
- Parameterized data width beyond 8-bit (fixed for this project)

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

Critical: All implemented functions are critical for functionality.

7 Tests and Methods

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

White Box – internal signal access required for corner cases

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

- White Box: + fine-grain control, – intrusive
- Black Box: + realistic, – hard to trace bugs
  White box selected due to requirement for signal-level visibility.

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

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

- Driver: Sends data to FIFO inputs
- Monitor: Observes outputs and flags

- Scoreboard: Compares expected vs actual outputs
- Checker: Validates flag assertions

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

Dynamic Simulation using directed tests – best for small projects with manageable state space.

7.1.5 What is your driving methodology?

Directed tests for predictable behavior; corner cases covered.

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

Directed tests for predictable behavior; corner cases covered.

7.1.6 What will be your checking methodology?

Assertions + scoreboard checks from spec expectations.

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

7.1.7 Testcase Scenarios (Matrix)

7.1.7.1 Basic Tests

Test Name / Number	Test Description/ Features
1.1.1	Check basic read operation
1.1.2	Check basic write operation

7.1.7.2 Complex Tests

Test Name / Number	Test Description/ Features
1.2.1	Concurrent events (R+W) Conditions: fifo_full/fifo_empty/always_full/always empty etc.
1.2.2	Write during full / Read during empty conditions

7.1.7.3 Regression Tests (Must pass every time)

Test Name / Number	Test Description/Features
1.3.1	Tests that should always pass
1.3.2	Reset recovery test

7.1.7.4 Any special or corner cases testcases

Test Name / Number	Test Description
1.4.1	Special Case testing tests and conditions
1.4.2	Bug injection and testing scenario

8 Coverage Requirements

8.1.1.1 Describe Code and Functional Coverage goals for the DUV

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

8.1.2 Assertions

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.

9 Resources requirements

9.1 Team members and who is doing what and expertise.

Team Member	Responsibilities	Focus Areas
Aravindh	- Write pointer logic & synchronization - Build initial SystemVerilog module	RTL design, Clock Domain Crossing (CDC)
Rohit	- Read pointer logic & flag generation (full/empty) - Assertions in TB	Pointer control, SV Assertions
Rishi	- FIFO memory logic - Scoreboard and checker development	Memory modeling, Verification components

Team Member	Responsibilities	Focus Areas
Shreya	- Testbench setup - Corner cases, test matrix, and final documentation	Testing strategy, Reporting, Git manager

10 Schedule

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

11 References Uses / Citations/Acknowledgements

Gitlink: https://github.com/aravindh-nanjaiya-latha/Team12_Asynchronous-FIFO_S25_ECE593