VERIFICATION TEST PLAN

ECE-593: Fundamentals of Pre-Silicon Validation

Maseeh College of Engineering and Computer Science

Winter, 2025

Project Name: Asynchronous FIFO

Members:

Jyothsna Tallada

Krishnavardhan Raju

Poojith Pogarthi

Sri Chandana Reddy Vanukuri

Date: 17/04/2025

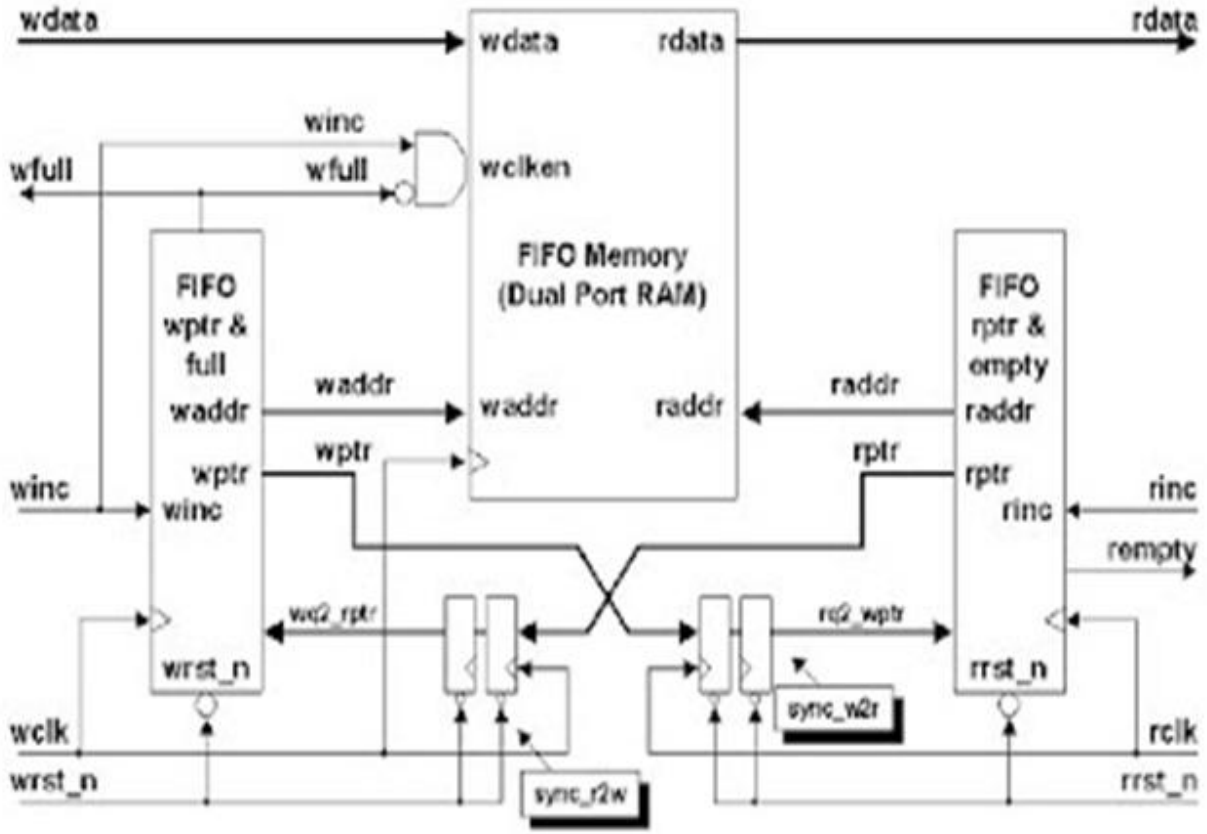
1 Table of Contents

2 Introduction:

2.1 Objective of the verification plan

The objective of this verification plan is to define a structured and systematic approach to verify the RTL design of an asynchronous FIFO. This includes ensuring functional correctness, reliable clock domain crossing (CDC) operation, and robustness under various operating conditions. The plan aims to validate that the FIFO correctly handles data transfers between asynchronous clock domains, accurately generates control signals (e.g., full and empty flags), and prevents issues such as data loss, metastability, or overflow/underflow. Additionally, it seeks to confirm compliance with design specifications, achieve comprehensive coverage metrics, and verify performance across a range of clock frequencies, phase differences, and stress scenarios, ensuring the FIFO is robust and ready for integration into larger systems.

2.2 Top Level block diagram



2.3 Specifications for the design

- Supports two independent clock domains: write clock (wr_clk) and read clock (rd_clk)
- Configurable data width (e.g., 8, 16, or 32 bits)
- Configurable FIFO depth (e.g., 16, 32, or 64 entries)
- Gray code pointers for clock domain crossing with 2-stage synchronization
- Control signals: write enable (wr_en), read enable (rd_en), full flag, empty flag
- Optional almost full and almost empty flags with programmable thresholds
- Active-low asynchronous reset (rst_n)
- Overflow and underflow protection

- Supports clock frequencies up to 200 MHz per domain

3 Verification Requirements

3.1 Verification Levels

3.1.1 What hierarchy level are you verifying and why?

We are verifying at the **module (RTL) level of the asynchronous FIFO** to ensure correctness of the FIFO logic before integration. This level is chosen to validate the complete functionality of the FIFO, including the memory buffer, Gray code pointers, synchronization logic, and control signals (e.g., full, empty flags). Verifying at the module level confirms reliable operation across asynchronous clock domains, accurate data transfers, and proper handling of edge cases like overflow and underflow. It ensures the design meets functional and performance specifications, such as clock frequency support and CDC integrity, before integration into a larger system.

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

Controllability and observability are high at the module (RTL) level of the asynchronous FIFO due to exposed interface signals and internal state accessibility. Controllability is achieved through direct access to input signals such as write enable (wr_en), read enable (rd_en), write data (wr_data), and clocks (wr_clk, rd_clk), allowing precise manipulation of the FIFO's operation. Observability is ensured via output signals like read data (rd_data), full, empty, and optional almost full/empty flags, as well as internal pointers and memory contents that can be monitored during simulation. This enables comprehensive testing of the FIFO's behavior, including data transfers, clock domain crossing, and edge cases, facilitating thorough verification.

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

FIFO Interface:

- **Inputs**: Write clock (wr_clk), read clock (rd_clk), write data (wr_data, configurable width e.g., 8/16/32 bits), write enable (wr_en), read enable (rd_en), active-low reset (rst_n).

- **Outputs**: Read data (rd_data, matching wr_data width), full flag, empty flag, optional almost full and almost empty flags.

Timing:

- Clocked by independent wr_clk and rd_clk (supporting up to 200 MHz).
- Active-low asynchronous reset (rst_n).
- Synchronization: 2-stage synchronizers for Gray code pointers across clock domains.

Specifications:

- Configurable FIFO depth (e.g., 16/32/64 entries).
- Overflow/underflow protection.
- Gray code pointers for reliable clock domain crossing.
- Optional programmable thresholds for almost full/empty flags.

4 Required Tools

4.1 List of required software and hardware toolsets needed.

- Synopsys VCS, ModelSim, or QuestaSim for simulation
- SystemVerilog or VHDL for RTL and testbench development
- GTKWave or Verdi for waveform viewing
- Synopsys Design Compiler or Xilinx Vivado for synthesis (if targeting ASIC or FPGA)
- Cadence JasperGold or Synopsys Formality for formal verification of clock domain crossing
- UVM (Universal Verification Methodology) libraries for testbench automation
- Perl or Python for scripting test generation and result analysis
- Hardware: Standard workstation with Linux/Windows OS for simulation and synthesis
- Optional: FPGA board (e.g., Xilinx Zynq) for prototyping and hardware validation

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

- **/rtl** – Contains async_fifo.sv (RTL code for the asynchronous FIFO)

- **/tb** – Contains testbench files (e.g., async_fifo_tb.sv) and test cases (e.g., basic_tests.sv, cdc_tests.sv)
- **/logs** – Stores simulation logs and coverage reports (e.g., sim_log.txt, coverage.rpt)
- **/scripts** – Includes simulation and synthesis scripts (e.g., run_sim.sh, compile.tcl)
- **/wave** – Stores waveform files for debugging (e.g., sim_wave.vcd)
- **Computer Resources**:
- Workstation with Linux (e.g., Ubuntu 20.04) or Windows OS
- Minimum 16 GB RAM, 4-core CPU (e.g., Intel i7 or equivalent) for simulation
- 500 GB SSD for storing design files, logs, and waveforms
- Optional: Access to a high-performance computing cluster for parallel simulation runs or formal verification
- FPGA board (e.g., Xilinx Zynq) for hardware prototyping, if required

5 Risks and Dependencies

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

Clock domain synchronization might cause data glitches because clocks run at different speeds

- **Mitigation**: Use a special coding trick (Gray code) and extra circuit steps to keep data safe.
- **Contingency**: Add more circuit steps if glitches still happen.

● Pointer errors could make the FIFO think it's full or empty when it's not

- **Mitigation**: Add checks and test tricky situations like reading and writing at the same time.
- **Contingency**: Create more tests to catch any pointer mistakes.

● Timing issues at fast clock speeds might mess up the FIFO's operation

- **Mitigation**: Check timing with tools and set rules to keep signals in sync.
- **Contingency**: Slow down the clock or adjust the design if timing fails.

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 operation: Check that data can be written correctly when FIFO is not full
● Read operation: Ensure data is read out in the correct order
● Full flag accuracy: Assert that Overflow is asserted when FIFO is full and no further writes are accepted
● Empty flag accuracy: Assert that underflow is asserted when FIFO is empty and no reads are allowed
● Pointer synchronization (Gray code): Verify Gray-coded write/read pointers cross domains via synchronizers without error
● Data integrity under asynchronous clocks: Confirm that data written under the write clock is read back intact under the read clock

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

Unknown as of now

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

Critical Functions :

● Write operation

● Read operation

● Full flag accuracy

● Empty flag accuracy

Non Critical Functions

Internal debug signals

7 Tests and Methods

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

Gray

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

We used $random to generate random numbers

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

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

7.1.5 What is your driving methodology?

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

7.1.6 What will be your checking methodology?

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	

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	

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	

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.

Member	Responsibilities
Jyothsna Tallada	Empty/Full logic, parts of class-based TB, UVM
Sri Chandana Reddy Vanukuri	FIFO Memory, parts of class-based TB , UVM
Poojith Pogarthi	2FF synchronizer, conventional tb,parts of class-based TB , UVM
Krishnavardhan Raju	Binary to gray and gray to binary conversion logic for synchronization of pointers, parts of class-based TB, UVM

10 Schedule

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

Week	Task
Week1	RTL Design
Week2	Test bench
Week3	Creating classes and objects

Week4	UVM
Week5	Report and project submission

11 References Uses / Citations/Acknowledgements

GITHUB LINK: https://github.com/poojith15/Team_11_Async_FIFO_gray