VERIFICATION TEST PLAN

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

Project Name: Design and Verification of Asynchronous FIFO

Members: Moulya, Subramanya, Aadithya, Rakshith

Date: 24/2/2025

Github_link: Team_10_AsyncFIFO_w25_ECE593

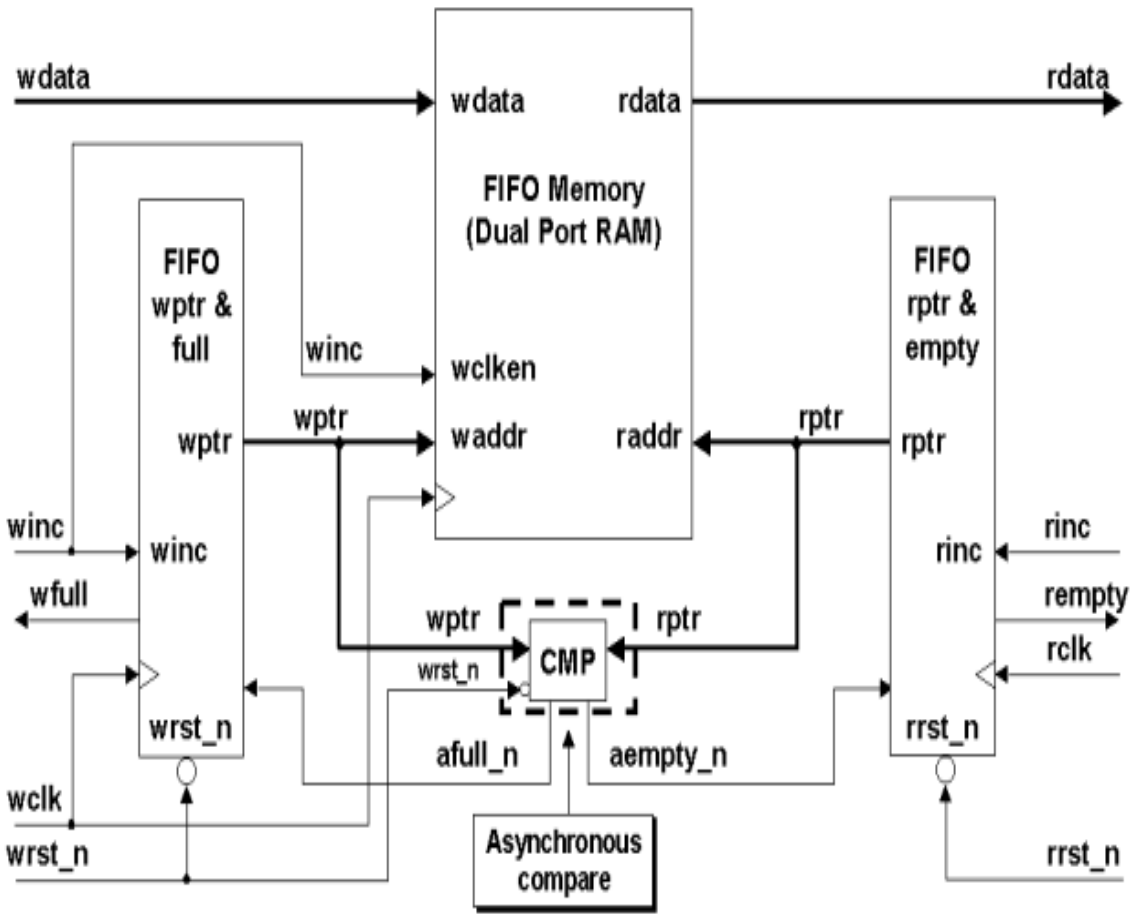
1 Table of Contents

2 Introduction:

2.1 Objective of the verification plan

The purpose of this verification plan for the asynchronous FIFO (First-In, First-Out) design in SystemVerilog is to ensure its correct functionality, reliability, and performance across a broad range of operational conditions. The plan aims to validate that corner cases and potential error scenarios are properly managed while thoroughly testing asynchronous data transfer, storage, and retrieval. It includes comprehensive verification of asynchronous read and write operations, data integrity, clock synchronization between read and write operations, and various FIFO states such as full, empty, half-full, and half-empty conditions. Additionally, adherence to defined interface protocols will be assessed. Through simulations, the plan will verify both functional accuracy and performance, ensuring the FIFO operates as intended within a larger system.

2.2 Top Level block diagram



2.3 Specifications for the design

The Asynchronous FIFO V3.0 is designed to enable reliable data transfer between two clock domains operating at different frequencies. It supports high-speed, low-latency communication with robust handling of full, empty, and boundary conditions. It performs Dual-clock asynchronous operation (separate write and read clocks), Configurable FIFO depth and data width, Support for full, empty, almost full, and almost empty flags, Gray code pointer synchronization for metastability reduction, Reset synchronization across both clock domains and Optional error detection for overflow/underflow-conditions.

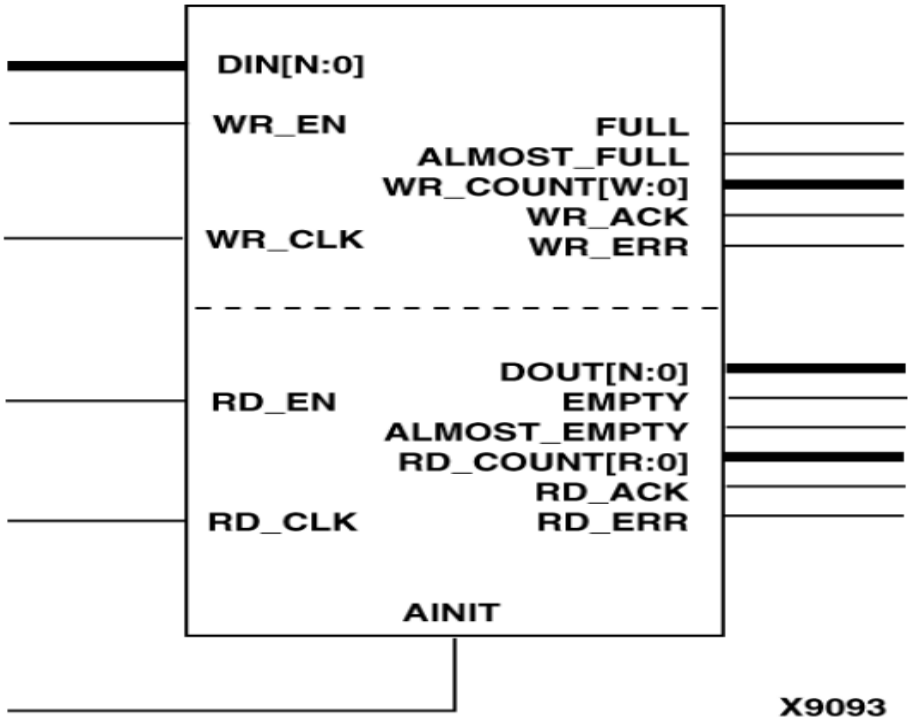


Fig : Core Schematic

DATA_WIDTH = 8

FIFO_DEPTH = 2^6 = 64

3 Verification Requirements

3.1 Verification Levels

3.1.1 What hierarchy level are you verifying and why?

Verification will be conducted at the module level, just below the top level, as the asynchronous FIFO design consists of six separate files, each containing a single module.

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

The modular structure allows for an efficient evaluation of each component's functionality, enabling precise control over stimulus through input and output configurations

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

Ensure proper handshaking and adherence to FIFO interface specifications.

4 Required Tools

4.1 Questa Sim

4.2 GitHub

5 Risks and Dependencies

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

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

Basic Read/Write Operations – Ensure correct FIFO behavior, including data integrity and asynchronous read/write handling.

Status Flags (Full, Empty, Almost Full/Empty, Half-Full) – Validate correct flag assertions under various FIFO states.

Clock Domain Synchronization – Verify proper handling of independent read and write clocks.

Boundary & Corner Cases – Test FIFO behavior under extreme conditions (writing to full, reading from empty, reset functionality).

Performance Metrics – Evaluate FIFO throughput, latency, and depth utilization.

Protocol & Interface Compliance – Ensure proper handshaking and adherence to FIFO interface specifications.

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

FIFO Memory Array Check
FIFO Memory Full Flag Detection
Write and Read Pointer increment
Full Flag generation
Empty Flag generation

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

TBD

7 Tests and Methods

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

White Box

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

TBD

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

Generator: Produces randomized or directed test data (e.g., write/read commands, data values) to cover functional scenarios like full/empty conditions, clock variations, etc.

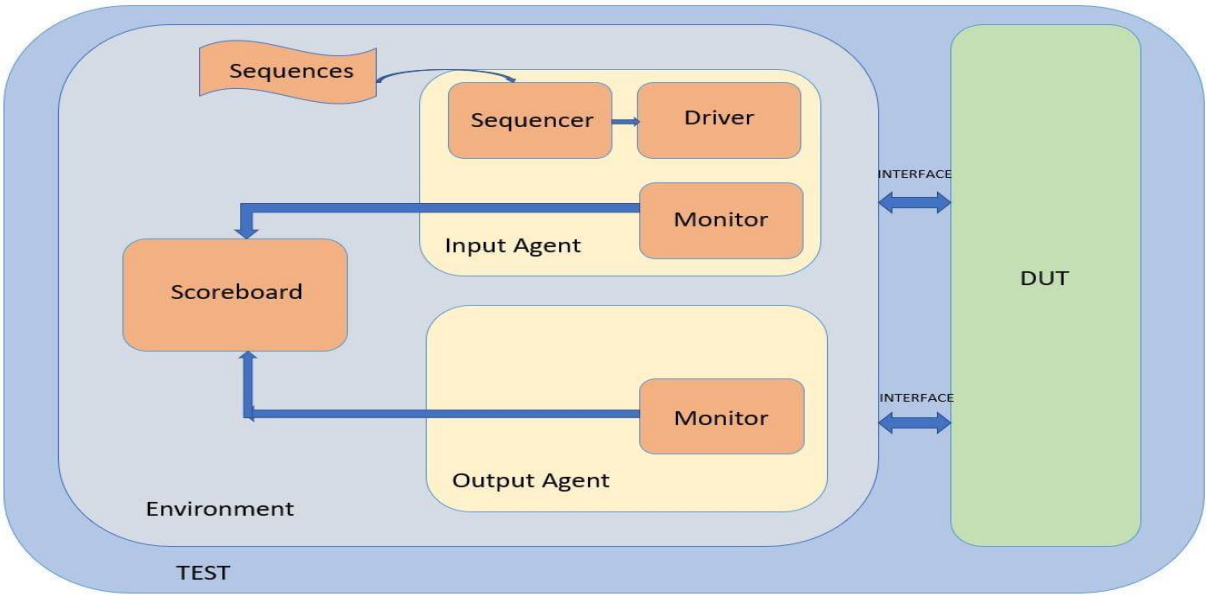
Driver: Drives generated stimuli into the FIFO DUT (Design Under Test) through the interface, managing write enables, data signals, and read requests.

Interface: Connects the driver, monitor, and FIFO DUT, handling read/write signals, clocks, and data lines for smooth communication.

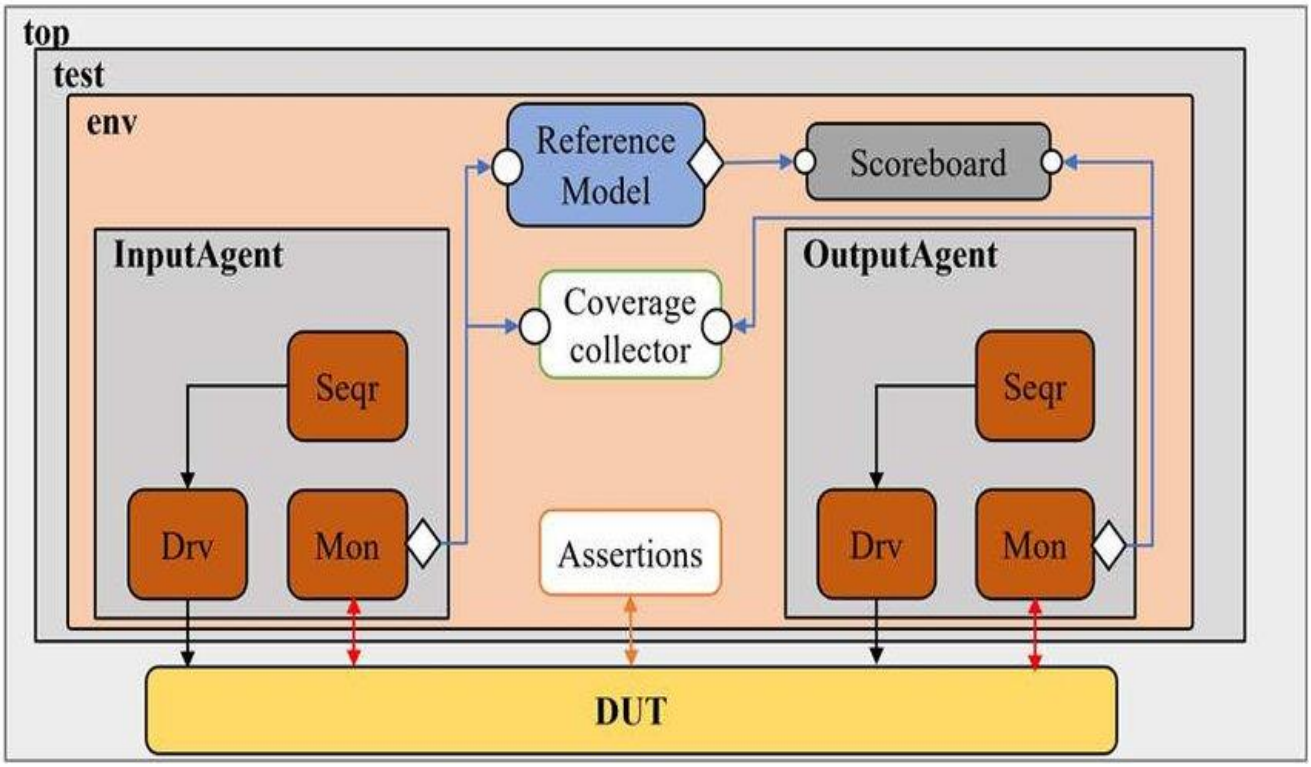
Monitor: Observes FIFO inputs/outputs, capturing transactions (write data, read data, flags) for result analysis.

Scoreboard: Compares monitored outputs with expected results to verify correctness, checking data integrity, order, and error handling

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



UVM Implementation of Asynchronous FIFO



1. **uvm_test (Test)**

   The highest-level component that controls the entire testbench and starts the stimulus.

   Implementation for FIFO:

   Instantiates the uvm_env (environment).

   Configures clock and reset signals for write and read domains.

   Initiates sequences to drive write and read operations.

   Manages different test scenarios like: Normal operation, FIFO full condition, FIFO empty condition, Asynchronous reset handling, and Simultaneous read and write operations.

2. **uvm_env (Environment)**

   Encapsulates all verification components, including write agent, read agent, scoreboard, and monitors.

   Implementation for FIFO:

   Contains two uvm_agent instances:

   Connects these agents to the DUT FIFO interface.

   Includes a uvm_scoreboard to validate data integrity.

   i. Write Agent (for the write clock domain).
   ii. Read Agent (for the read clock domain).

3. **uvm_agent (Agents)**

   a. Write Agent (uvm_agent)

   Generates and drives write transactions to the FIFO.

   Components:

   a. **uvm_driver** – Sends write transactions to FIFO.
   b. **uvm_sequencer** – Generates stimulus sequences for write operations.
   c. **uvm_monitor** – Captures write-side transactions for analysis.

   Functionality:

   d. Randomly generates data and writes to FIFO.
   e. Ensures writes are blocked when FIFO is full.
   f. Operates under the write clock domain**.**

   b. Read Agent (uvm_agent)

   Generates read transactions and collects data from FIFO.

   Components:

   - **uvm_driver** – Sends read transactions to FIFO.
   - **uvm_sequencer** – Generates stimulus sequences for read operations.
   - **uvm_monitor** – Captures read-side transactions for checking.

- **Functionality:**
  - Reads data from FIFO.
  - Ensures correct order of data retrieval (FIFO behavior).
  - Handles empty condition correctly.
  - Operates under the **read clock domain**.

4. **uvm_driver (Driver)**

Converts high-level uvm_sequence_item transactions into pin-level signals.
Implementation for FIFO:
Write Driver: Drives wdata, wr_en, and wr_clk signals to the FIFO.
Read Driver: Drives rd_en and rd_clk signals to FIFO.
Ensures timing constraints between write and read domains.

5. **uvm_sequencer (Sequencer)**

Provides transactions to the driver in an ordered or randomized manner.
Implementation for FIFO:
Write Sequencer: Generates sequences with randomized write data.
Read Sequencer: Generates read transactions based on FIFO status.
Controls transaction flow to ensure stimulus diversity.

6. **uvm_monitor (Monitor)**

Observes DUT transactions without affecting operations and sends data to the scoreboard.
Implementation for FIFO:
Write Monitor: Observes wdata, wr_en, and full signals. Sends captured transactions to the scoreboard.
Read Monitor: Observes rdata, rd_en, and empty signals. Ensures data integrity across clock domains.

7. **uvm_scoreboard (Scoreboard)**

Compares expected vs. actual data to verify FIFO behavior.
Maintains a reference model (Queue) to track written and expected read data.
Compares FIFO outputs against the expected data.
Checks for: Correct ordering of reads (FIFO behavior), Loss of data across clock domains, and Underflow/Overflow handling.

8. **uvm_sequence (Sequence)**

Purpose: Defines test scenarios with ordered stimulus transactions.
Implementation for FIFO:
Write Sequence: Generates varying patterns of data writes.
Read Sequence: Controls read enable and verifies data output.
Includes corner cases like: Full and empty conditions, Randomized back-to-back operations, and Clock skew and jitter conditions.

9. **uvm_sequence_item (Transaction)**

Purpose: Represents a single transaction (stimulus item) in the testbench.
Implementation for FIFO:
Write Transaction Fields: wdata (write data), wr_en (write enable), wr_clk (write clock)
Read Transaction Fields: rdata (read data), rd_en (read enable), rd_clk (read clock)
These fields are randomized in the test sequences.

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

Dynamic Simulation

7.1.5 What is your driving methodology?

TBD

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

Constrained Randomization

7.1.6 What will be your checking methodology?

Class based testbench architecture, Code Coverage, Functional Coverage (both Data and Control oriented coverage)

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 Top Module: Write and Read	Check basic write and read operation
1.1.2 Reset Operation	Reset operation from top module must propagate through the entire design

7.1.7.2 Complex Tests

Test Name / Number	Test Description/ Features
1.2.1 FIFO Memory Array Check	Check Read and Write into Array at module level
1.2.2 FIFO Memory Full Flag Detection	Checks to see if an asserted full flag
1.2.3 Write and Read Pointer increment	Write Pointer and Read pointer must increment on w_inc and r_inc
1.2.4 Full Flag generation	Ensures Full flag generation logic functions properly on write pointer wrap around
1.2.5 Empty Flag generation	Ensures Empty flag generation logic functions properly on read pointer equal to write pointer

7.1.7.3 Regression Tests (Must pass every time)

Test Name / Number	Test Description/Features
1.3.1 Top Module: Write and Read	Check basic write and read operation
1.3.2 Reset Operation	Reset operation from top module must propagate through the entire design
1.3.3 FIFO Memory Array Check	Check Read and Write into Array at module level
1.3.4 FIFO Memory Full Flag Detection	Check Read and Write into Array at module level
1.3.5 Write and Read Pointer increment	Write Pointer and Read pointer must increment on w_inc and r_inc

7.1.7.4 Any special or corner cases testcases

Test Name / Number	Test Description
1.4.1 Read to Write Synchronizer pipeline Write to Read Synchronizer pipeline	Ensures a Pointer passed is synchronized during clock domain crossings
1.4.2 TBD	Bug injection and testing scenario

8 Coverage Requirements

8.1.1.1 Describe Code and Functional Coverage goals for the DUV

Aiming to achieve Code coverage of over 96% and Functional Coverage of over 90%

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

TBD

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.

Planning to use immediate assertion in the scoreboard to do the comparison of outputs coming from DUV with that of self-checking logic

9 Resources requirements

9.1 Team members and who is doing what and expertise.

Tasks	**Done by**	**Planned Date**	**Status**
1. Design specification document	Aaditya and Subramanya	Jan - 30 - 2025	Done
2. Verification Plan	Moulya and Rakshith	Jan - 30 - 2025	Done
3. Design implementation: 1) fifo_mem	Moulya and Rakshith	Feb – 15 -205	Done
2) synchronizer_r2w	Aaditya and Subramanya	Feb – 16- 2025	Done
3) synchronizer_w2r	Aaditya and Subramanya	Feb – 16- 2025	Done
4) rptr_handler	Aaditya and Subramanya	Feb – 18- 2025	Done

5) wptr_handler	Aaditya and Subramanya	Feb – 18- 2025	Done
6) Top Module	Moulya and Rakshith	Feb – 18- 2025	Done
Conventional Testbench	Moulya and Rakshith	Feb – 18- 2025	Done
Scoreboard (Self Checking)	Aaditya and Subramanya	Feb – 25- 2025	Done
Code Coverage (SV TB)	Aaditya and Subramanya	Feb – 25- 2025	Done
Functional Coverage (SV TB)	Moulya and Rakshith	Feb-27-2025	Done
UVM TB Components	Moulya and Rakshith Aaditya and Subramanya	Feb-27-2025	Done
Code and Functional Coverage for UVM TB	Moulya and Rakshith Aaditya and Subramanya	Feb-27-2025	Done
Add few scenarios of Bug-Injection	Rakshith and Aadithya	Mar-3-2025	Done
Verify the error/bug-injection functionality	Rakshith and Aadithya	Mar-3-2025	Done
Finalize documents and presentation	Moulya and Subramanya	Mar-3-2025	Done

10 Schedule

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

Sl No	**Tasks**	**Planned Date**	**Completed Date**	**Status**
1	Design of Asynchronous FIFO with clean build	Jan-31-2025	Jan-30-2025	Done
2	Checking the design with the basic flat testbench and complete generator block with the necessary constraints	Feb-7-2025	Feb-5-2025	Done
3	Developing the driver class and drive the data it gets from generator to the DUT through mailbox Design interface block	Feb-14-2025	Feb-10-2025	Done
4	Develop code to interaction from DUT to Monitor via Virtual Interface and send the data to the Monitor Develop self-checking testbench in the scoreboard and compare those output with that of DUV output	Feb-18-2025	Feb-15-2025	Done

5	Add the cover groups comprising necesarry coverpoints and assertion in scoreboard and generate cover report and check if the design is metting the specifcation.	Feb-18-2025	Feb-17-2025	Done
6	Develop whole TB using UVM	Feb-28-2025	Feb-24-2025	Done
7	For UVM based TB : Add the cover groups comprising necesarry coverpoints and assertion in scoreboard and generate cover report and check if the design is metting the specifcation	Feb-28-2025	Feb-24-2025	Done
8	Add a scenario to check the error/bug-injection condition and verify the same	Mar-3-2025	Mar-1-2025	Done
9	Finalize the design and testplan document Make a presentation	Mar-3-2025	Mar-2-2025	Done

11 References Uses / Citations/Acknowledgements

https://verificationguide.com/uvm/uvm-testbench-architecture/

https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/

https://research.ijcaonline.org/volume86/number11/pxc3893347.pdf

https://zipcpu.com/blog/2018/07/06/afifo.html