Verification Plan

Team 5 Asynchronous FIFO

May 2, 2025

ECE 593
Dr. Venkatesh Patil

Team 5 Members:
Kai Roy, Pavan Gaddam, SuryaTeja Purma, Gene Hu

1 Table of Contents

1 Table of Contents

2 Introduction

- 2.1 Team 5 Github Repository Link
- 2.2 Objective of the verification plan
- 2.3 Top Level block diagram
- 2.4 Specifications for the design

3 Verification Requirements

- 3.1 Verification Levels
  - 3.1.1 Hierarchy Level
  - 3.1.2 Controllability and Observability
  - 3.1.3 List of Interfaces and Specification

4 Required Tools

- 4.1 List of required software and hardware toolsets needed.

5 Risks and Dependencies

- 5.1 Risks
- 5.2 Mitigation

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
  - 6.1.2 List of functions that will not be verified / need to be implemented
  - 6.1.3 List of critical functions and non-critical functions for tapeout

7 Tests and Methods (TEST CASES)

- 7.1.1 Testing methods to be used
- 7.1.2 State the PROs and CONs for each and why you selected the method for this DUV.
- 7.1.3 Testbench Architecture; Component used (list and describe Drivers, Monitors, scoreboards, checkers etc.)
- 7.1.4 Verification Strategy
- 7.1.5 What is your driving methodology?
- 7.1.6 What will be your checking methodology?
- 7.1.7 Testcase Scenarios (Matrix)
  - 7.1.7.1 Basic Tests
  - 7.1.7.2 Complex Tests
  - 7.1.7.3 Regression Tests (Not at this point yet)
  - 7.1.7.4 Any special or corner cases test cases

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 Roles and Responsibilities
10 Schedule
11 References

2 Introduction

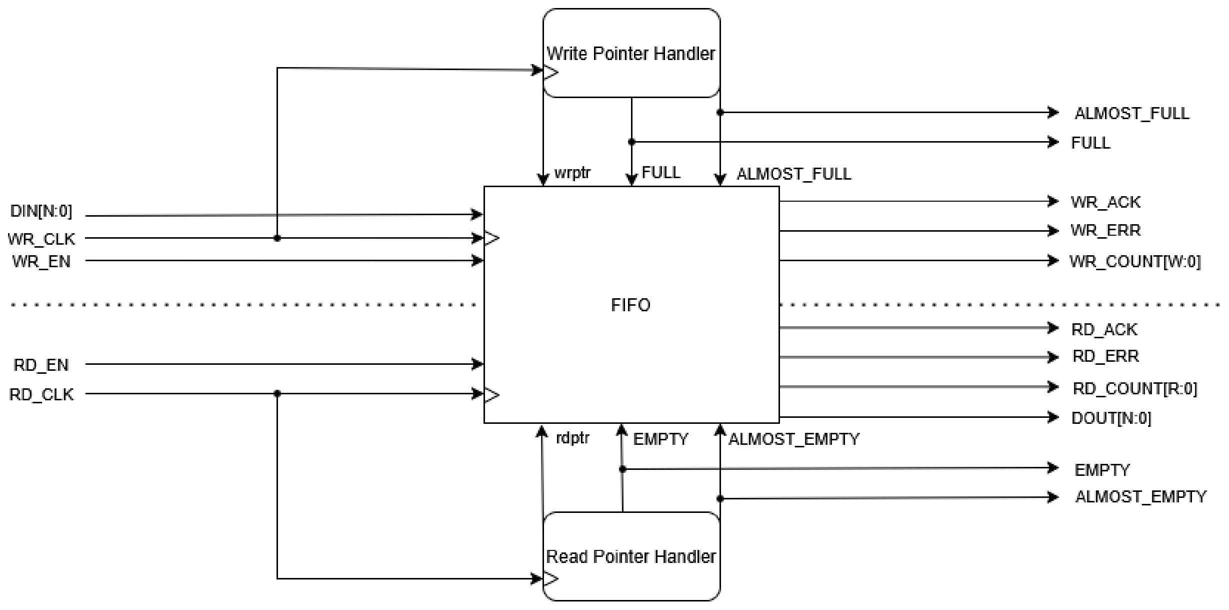
2.1 Team 5 Github Repository Link

https://github.com/xbicai/ECE593_Project

2.2 Objective of the verification plan

The Purpose of this document is to provide members of the project a guideline for how the asynchronous FIFO should function. By having a plan to follow, each member will be able to put in work that will help reach the end goal.

2.3 Top Level block diagram



2.4 Specifications for the design

Data can be safely passed from one asynchronous clock domain to another using FIFO. When two clock domains are asynchronous to one another, an asynchronous FIFO design occurs when data values are written from one clock domain to a FIFO buffer and read from the same FIFO buffer from another clock domain.

The next word to be written is always shown by the write pointer. The write pointer increases and the empty flag is cleared as soon as the first piece of data is written to the FIFO. The current FIFO word to be read is always shown by the read pointer. When the read and write pointers are equal, the FIFO is empty.

FIFO Depth Calculation:
Double the frequency of write and half of the read
120 MHz is the sender clock frequency.
Three idle cycles separate two consecutive writes, and the receiver clock frequency is 50 MHz.
There are two idle cycles in between two consecutive reads.
Given that the sender clock frequency is less than the receiver clock frequency, write burst = 1024.

Three clock cycles separate two consecutive writes, indicating that after writing one piece of data, the write module waits three clock cycles before starting the next write. This means that one piece of data is written every four clock cycles.
Writing one data item takes 4 * (1/120MHz) = 33.33 ns.
It took 34,129.92 = 34,129 ns to write the data in the burst.
One data item's reading time is equal to 3 * (1/50MHz) = 60 ns.
Thus, the read module will read one data item from the burst every 60 ns.
In 34,129 ns, the number of data pieces that can be read is 34,129/60 = 568.81 = 568 items. 1024 – 568 = 456 is the remaining number of bytes that must be kept in the FIFO.

3 Verification Requirements

3.1 Verification Levels

3.1.1 Hierarchy Level

This project will be verified at the **block level** since we want to verify the entire asynchronous FIFO module. Such parts include data correctness, proper clock timing, and flag handling (empty/full).

3.1.2 Controllability and Observability

Controllability will be good because the asynchronous FIFO's inputs will be directly controlled by the testbench. Observability will be good too, because the flags such as empty/full can be monitored directly by the testbench or QuestaSim.

3.1.3 List of Interfaces and Specification

4 Required Tools

4.1 List of required software and hardware toolsets needed.

- QuestaSim

5 Risks and Dependencies

5.1 Risks

- Asynchronous Clock Synchronization: Metastability in pointer synchronization across clock domains.
- Coverage Gaps: Missing corner cases (e.g., simultaneous read/write with maximum dept

5.2 Mitigation

- Use constrained-random tests to stress clock domain crossings.
- Implement comprehensive coverage metrics and review reports weekly.
- Schedule tool access and optimize testbench for simulation efficiency.

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

Function	Description
Write Operation	Data is being written in correctly when **wen** high and **fifo_full** low
Read Operation	Data is being read out correctly when **ren** high and **fifo_empty** low
Reset	When **wrst_n_i** is asserted the write reg/signals are reset to initial state When **rrst_n_i** is asserted the read reg/signals are reset to initial state
Data Integrity	FIFO correctly outputs the same data in the same order it was put in
Full / Emply Flags	**fifo_full** is asserted when fifo is full **fifo_empty** is asserted when fifo is empty
Data width and memory depth	Supports data widths up to 256 bits Supports memory depths of up to 65,535 locations

6.1.2 List of functions that will not be verified / need to be implemented

Function	Why
Almost full / empty flags	Not implemented yet
Read / write requests are rejected without affecting FIFO state	Not implemented yet
Write / read acknowledge flags	Not implemented yet

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

To keep it short for now, everything that is implemented so far is critical, and the unimplemented functions listed above are non-critical.

7 Tests and Methods (TEST CASES)

7.1.1 Testing methods to be used

The team will be using gray box testing over black or white box testing.

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

	Pros	Cons
Black	• Quickly validate functionality • No need to understand internal signals	• Will miss internal bugs • Low controllability/observability of internal signals
Gray	• Have some knowledge of internal signals ○ Leads to better detection of internal bugs	• Still don't see every internal signal or structure, which means some bugs might be missed
White	• Highest controllability/observability	• High costs for maintenance (if one internal signal is changed, test might need a reflecting change)

The team decided on gray box testing because it provides the most balanced approach when it comes to controllability and observability. The most important signals and protocols can be tested along with their functionalities.

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

- Driver: Generates DIN, WR_EN, RD_EN, AINIT synchronized to WR_CLK and RD_CLK. Uses random DIN values.
- Monitor: Observes DOUT, FULL, EMPTY, ALMOST_FULL, ALMOST_EMPTY, WR_ACK, WR_ERR, RD_ACK, RD_ERR, WR_COUNT, RD_COUNT.
- Scoreboard: Tracks written data and compares with read data to verify integrity. Maintains expected FIFO state (e.g., occupancy).
- Checker: Automates comparison of actual vs. expected outputs (replacing manual $display statements).

7.1.4 Verification Strategy

The team decided on **dynamic simulation** since it's suitable for verifying asynchronous behavior, flag updates, and handshake signals. Formal verification may be added for pointer synchronization.

7.1.5 What is your driving methodology?

The driving methodology will be a mix of directed tests and constrained random testing depending on the need of the tests.

- Directed Tests: Reset, basic write/read, fill/empty scenarios.
- Constrained Random: Random DIN, random delays between read/write operations.

7.1.6 What will be your checking methodology?

- From specification: Verify flags, handshake signals, and count vectors.
- Automated checkers compare actual vs. expected outputs.

7.1.7 Testcase Scenarios (Matrix)

7.1.7.1 Basic Tests

Number	Name	Test Description/ Features	Status
1.1.1	Standard write/read	Write 01234567 in Read 01234567 out	Not done
1.1.2	Random write/read stress test	Write many random bursts of data in, then confirm FIFO correctly reads these bursts out. Do this over a long period (write a lot, read a lot, write a lot...)	Not done
1.1.3	Fill	Write into FIFO until it's FULL, **fifo_full** should be asserted	Not done
1.1.4	Drain	Read from FIFO until it's EMPTY, **fifo_empty** should be asserted	Not done
1.1.5	Write when full	Write into FIFO when **fifo_full** is asserted; FIFO should block write	Not done
1.1.6	Read when empty	Write into the FIFO when **fifo_empty** is asserted; FIFO should block read	Not done

1.1.7	Reset	Reset when empty Reset when partially full Reset when full Make sure correct signals are reset in any instance	Not done
1.1.8	Random interval write/read	Set the timing between writes and reads to a random time, FIFO should still correctly raise FULL/EMPTY flags and have data integrity	Not done
1.1.9	Random data size write/read	Write randomly sized data into FIFO and read them out when full, check for data integrity	Not done

7.1.7.2 Complex Tests

Number	Name	Test Description/ Features	Status
1.2.1	Concurrent testing 1	Concurrent read and write Condition: **fifo_full** What should happen: Read is allowed, write is blocked, **fifo_full** is low after clock	Not done
1.2.2	Concurrent testing 2	Concurrent read and write Condition: **fifo_empty** What should happen: Read is blocked, write is allowed, **fifo_empty** is low after clock	Not done
1.2.3	Concurrent testing 3	Concurrent read and write Condition: **fifo_empty** and **fifo_full** What should happen: Read is blocked, write is blocked, **fifo_empty/fifo_full** is low after clock	Not done
1.2.4	Concurrent testing 4	Concurrent read and write after inserting a single test data into fifo Condition: no flags asserted What should happen: Read out the test data, write in new data	Not done

7.1.7.3 Regression Tests (Not at this point yet)

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 test cases

Number	Name	Test Description/ Features	Status
1.4.1	WR	Check input output of W->R->W->R many times **fifo_empty** should be asserted after every read	Not done
1.4.2	WWR	Check input output of W->W->R->W->W->R many times	Not done

1.4.3	RRW	**fifo_full** should be asserted eventually Check input output of R->R->W… many times First two reads will be blocked, then it will start allowing the first read and blocking the second, empty flag is asserted correctly	Not done

8 Coverage Requirements

8.1.1.1 Describe Code and Functional Coverage goals for the DUV

The goal for code coverage in this project is 100%, we have not discussed the functional coverage yet, but it will happen at the next milestone.

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 Roles and Responsibilities

Role	Main	Helper
RTL coding	Pavan	Suryateja, Kai
Testbench coding	Kai	Suryateja, Pavan, Gene
Design specification	Gene	
Verification plan	Gene, Suryateja	

10 Schedule

Objective	Description	Date
Milestone 1	● Complete design specification ● Start initial verification plan ● Initial implementation of design ● Simple initial testbench	4/20/25

Milestone 2	• Develop class based testbench with completed interfaces ○ Has components: generator, driver, monitor, scoreboard, mailbox ○ Verify 20-50 random bursts of data • Update verification plan	5/2/25
Milestone 3	• Finalize changes to RTL • Complete class based testbench ○ Finish coverage here as well • Add reports for code and functional coverage • Update verification plan	N/A
Milestone 4	• Develop UVM TB • Update verification plan to include UVM plan • Utilize UVM_MESSAGE and UVM_LOGGING to create logs and reports about the data	N/A
Milestone 5	• Complete UVM architecture, environment, and testbench • All test cases completed with coverage reflected • Create RTL bug as a bug injection and verify • Finalize all documents and presentation	5/25/25

11 References

- Xilinx Asynchronous FIFO V3.0 Design Specification.
- SystemVerilog IEEE 1800-2017 Standard.
- ECE-593 Course Materials, Winter 2025.