VERIFICATION TEST PLAN

ECE-593: Fundamentals of Pre-Silicon Validation

Maseeh College of Engineering and Computer Science Winter 2025

Github link: https://github.com/naidumaheshchowdary/Team_15_Async_FIFO

Team 15:

Alaina Anand Nekuri (PSU ID: 959604917)
Mahesh Naidu (PSU ID: 917048048)
Siddhartha Kaushik Gatta (PSU ID: 946785223)
Venkata Sai Dhilli (PSU ID: 980087156)

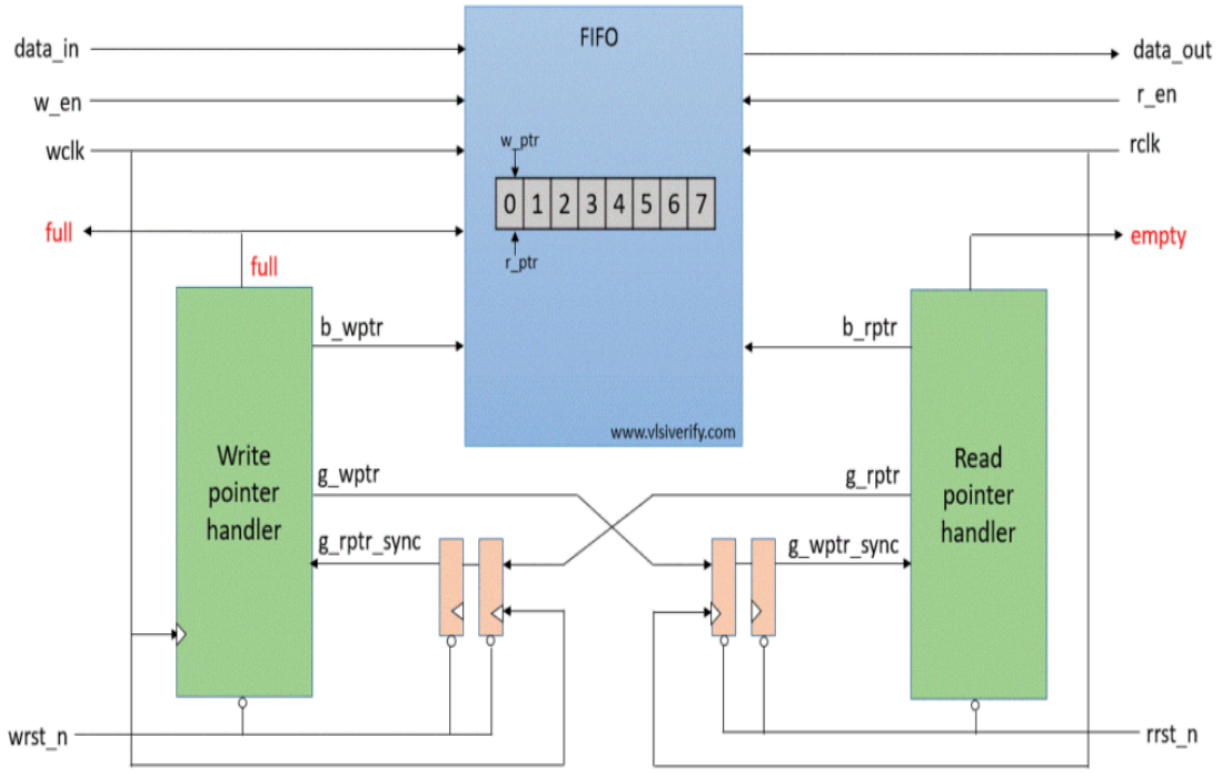
Table of Contents

2 Introduction:

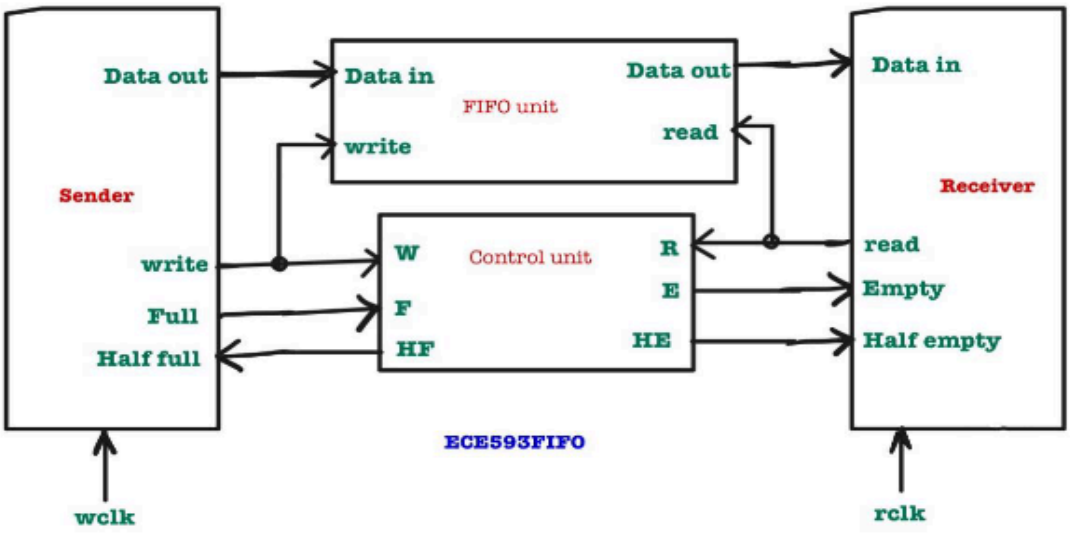
2.1 Objective of the verification plan

The objective of the verification plan is to ensure the Asynchronous FIFO correctly transfers data between independent write 120 MHz and read 50 MHz clock domains while maintaining data integrity. The testbench validates that data written into the FIFO is correctly read in a first-in, first-out (FIFO) manner. It checks that the full flag prevents writes when the FIFO is full and the empty flag blocks reads when empty. The design's ability to handle idle cycles (4 cycles between writes, 2 cycles between reads) is also verified. Random data is generated and stored in a queue expected_data to compare with actual FIFO output rd_data. The simulation runs through multiple iterations, ensuring FIFO functionality under different conditions before termination.

2.2 Top Level block diagram



Asynchronous FIFO



2.3 Specifications for the design

The design consists of a memory buffer, a write pointer, a read pointer, and control logic to handle data movement while preventing overflow and underflow conditions. The write pointer keeps track of the memory location where the next data item will be written, while the read pointer indicates the current location from which data should be read. Since the FIFO operates in two different clock domains i.e, Writing frequency of sender is 120MHz and Read Frequency of receiver is 50MHz, it is crucial to implement a synchronization mechanism to avoid metastability issues.

During initialization or reset, both the read and write pointers are set to zero, indicating that the FIFO is empty. Data is written to the location specified by the write pointer, and after every successful write operation, the pointer is incremented. Similarly, the read pointer moves forward after every read operation. The FIFO remains empty when both pointers are equal, and the empty flag is asserted to prevent unintended reads. When the write pointer wraps around and catches up with the read pointer, the FIFO is considered full, and the full flag is asserted to prevent additional writes.

One of the key challenges in asynchronous FIFO design is distinguishing between the full and empty states, as both occur when the read and write pointers are equal. To address this, an extra bit is added to each pointer, which toggles when the FIFO wraps around. If the extra bit of the write pointer differs from that of the read pointer, it indicates that the FIFO has wrapped around and is full. On the other hand, when both pointers, including the extra bit, are equal, the FIFO is empty.Another critical aspect of the design is the use of Gray-coded pointers for synchronization. When transferring data between different clock domains, binary counters can lead to metastability because multiple bits may change simultaneously. Using Gray code ensures that only one bit changes at a time, reducing the likelihood of metastability and improving the stability of pointer synchronization.

The design also includes logic to handle overflow and underflow conditions. When the FIFO is full, additional write operations are ignored to prevent data corruption. Likewise, when the FIFO is empty, read operations are blocked to avoid retrieving invalid data. The FIFO operates efficiently by ensuring that the read pointer always points to the next valid data, minimizing the number of clock cycles required for data retrieval.

Overall, the design of the Asynchronous FIFO focuses on efficient memory utilization, proper synchronization, and robust pointer management to provide reliable data transfer across different clock domains.

3 Verification Requirements

3.1 Verification Levels

3.1.1 What hierarchy level are you verifying and why?

- The code is verifying the module-level functionality of the asynchronous FIFO.
- This is the initial stage (Checkpoint 1), where the focus is on basic read/write operations, pointer behavior, and flag assertions (full/empty).

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

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

- Yes, the interfaces are clearly defined:

  Inputs:

  - wr_en, wr_clk, wr_reset, wr_data (write interface).
  - rd_en, rd_clk, rd_reset (read interface).

  Outputs:

  - rd_data (read data).
  - fifo_full (full flag).
  - fifo_empty (empty flag).

4 Required Tools

4.1 List of required software and hardware toolsets needed.

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

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 Write/Read: Ensure data is written and read correctly.
- Full/Empty Flags: Verify FIFO full and empty conditions.
- Pointer Increment: Check write and read pointer updates.
- Reset Operation: Ensure FIFO initializes correctly.
- Cross-Clock Sync: Validate data transfer between clock domains.
- Concurrent R/W: Test simultaneous read and write operations.

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

- Error Handling: Overflow/underflow checks deferred.
- Power-On Reset: Advanced verification for later.
- Performance Metrics: Throughput/latency not tested yet.

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

7 Tests and Methods

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

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.)

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
	Check basic read operation

7.1.7.2 Complex Tests

me / Number	scription/ Features
	rent events (R+W) ons: fifo_full/fifo_empty/always_full/always empty etc.

7.1.7.3 Regression Tests (Must pass every time)

me / Number	scription/Features
	at should always pass

7.1.7.4 Any special or corner cases testcases

me / Number	scription
	Case testing tests and conditions
	ection 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.

Alaina Anand Nekuri - Gone through the design specifications and have implemented the synchronization read to write module and top module.

Mahesh Naidu – Have gone through the design specifications and have done the synchronization, synchronization write to read module, synchronizer module.

Venkata Sai dhilli – Have gone through the design specifications and tested the design.

Siddhartha Kaushik Gatta– Have gone through the design specifications, calculated the FIFO depth calculations, FIFO memory module.

10 Schedule

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

	Tasks	**Timeline**
1	Design specs, Verification Plan, RTL implementation, basic testbench.	Week 1-4
2	Class-based TB, transactions, generator, driver, randomized data tests.	Week 4-7
3	Finalize RTL, complete class-based TB, coverage reports.	Week 8-10
4	Develop UVM TB, add UVM plan, implement logging.	Week 11-13
5	Complete UVM TB, final coverage, bug injection, final docs, presentation.	Week 14-16

11 References Uses / Citations/Acknowledgements

[1] https://hardwaregeeksblog.wordpress.com/wp-content/uploads/2016/12/fifodepthcalculationmadeeasy2.pdf

[2] http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf

[3] https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/

[4] https://ieeexplore.ieee.org/document/10090696