

ECE-593
FUNDAMENTALS OF PRE-SILICON VALIDATION



FINAL PROJECT
Milestone-2
Verification Plan

**Implementation and Verification of Asynchronous FIFO
using both Class-based and UVM methodologies.**

SECTION-2

GROUP-4

Github: https://github.com/Srikar0306/Pre_Silicon_Validation_Final_Project

Sai Rohith Reddy Yerram (ID: 909940739)

Rakshita Joshi (ID: 929533031)

Suhail Ahamed Subairudeen (ID: 903514017)

Srikar Varma Datla (ID: 953450016)

Table of Contents

- I. Design Overview
- II. FIFO Depth Calculation
- III. Verification Environment
- IV. Required Tools
- V. Verification Plan
- VI. Contributions
- VII. Resources
- VIII. Additional Comments

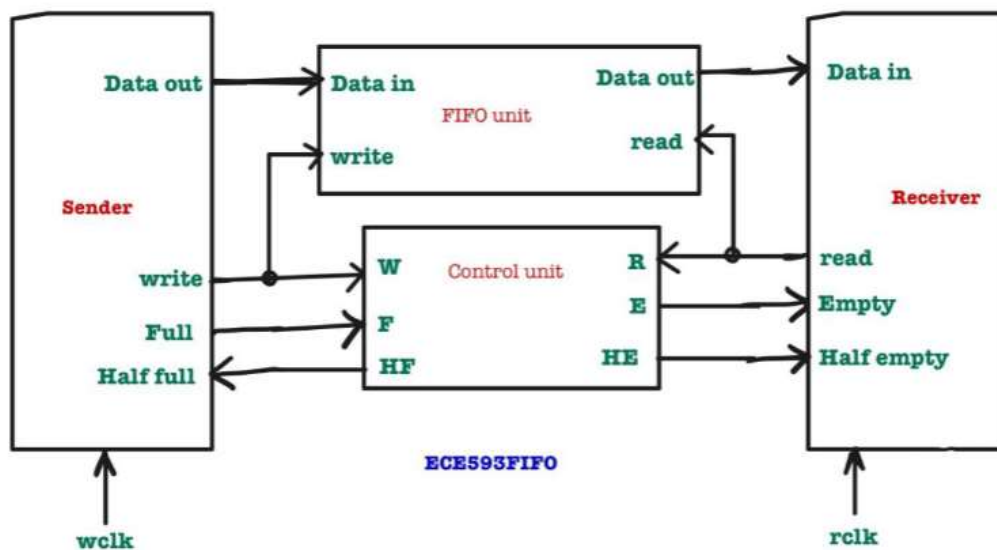
I. Design Overview

The asynchronous FIFO design encompasses several key modules and components to facilitate efficient data transfer, storage, and retrieval. The central component of the design is the FIFO module itself, which manages the queueing and asynchronous data transfer operations. It ensures the correct handling of incoming data, storage within the FIFO, and subsequent retrieval in a first-in-first-out manner.

Additionally, two synchronizer modules are integrated into the design to facilitate the seamless synchronization of signals between different clock domains. These synchronizers play a crucial role in maintaining data integrity and avoiding metastability issues when transferring data between asynchronous clock domains.

To provide essential status information about the FIFO, modules indicating half empty, half full, full, and empty states are included. These modules enable the monitoring of the FIFO's occupancy status, allowing external systems to make informed decisions based on the current state of the FIFO.

Furthermore, a class-based testbench has been developed to thoroughly verify the functionality and performance of the asynchronous FIFO design. This testbench utilizes randomized bursts, as specified in the milestone 2 requirements, to comprehensively validate the FIFO under various operational conditions. By executing 20-50 randomized bursts, the testbench ensures robust testing coverage, including corner cases and error scenarios, to certify the reliability and functionality of the FIFO design.



Level Block Diagram of the Design System

High

Understanding the FIFO design involves grasping the functionality of FIFO pointers.

The write pointer indicates the next word to be written, starting at zero upon reset. When data is written, the write pointer increments to the next location.

Similarly, the read pointer indicates the current word to be read, also starting at zero after reset. When data is written, the write pointer increments, clearing the empty flag and allowing the read pointer to drive the first valid word onto the output port for the receiver to read immediately.

The FIFO is considered empty when both pointers are equal, and full when they are equal again after wrapping around. To distinguish between empty and full states, an extra bit is added to each pointer. If the MSBs of the pointers differ, it indicates the write pointer has wrapped more times than the read pointer. Conversely, if the MSBs are the same, both pointers have been wrapped the same number of times. With this setup, the FIFO is empty when all pointers, including the MSBs, are equal, and full when all pointers except the MSBs are equal.

II. FIFO Depth Calculation:

Sender Clock Frequency = 240MHz | Number of idle cycles between two successive writes = 0

Receiver Clock Frequency = 400MHz | Number of idle cycles between two successive reads = 2

Write Burst = 512 | Given, Sender Clock Frequency < Receiver Clock Frequency

The number of idle cycles between two successive writes is 0 clock cycles, which means after writing one data, the write module is waiting for 0 clock cycles to initiate the next write, meaning every 1 clock cycle one data is written.

The number of idle cycles between two successive reads is 2 clock cycles, which means after reading one data, the read module is waiting for 2 clock cycles to initiate the next read, meaning every 3 clock cycles one data is read.

Time required to write one data item = $1 * (1/240\text{MHz}) = 4.166 \text{ ns}$.

Time required to write the data in the burst = $512 * 4.166 = 2132.992 \text{ ns}$.

Time required to read one data item = $3 * (1/400\text{MHz}) = 7.5 \text{ ns}$

So, for every 7.5ns read module is going to read one data item in the burst.

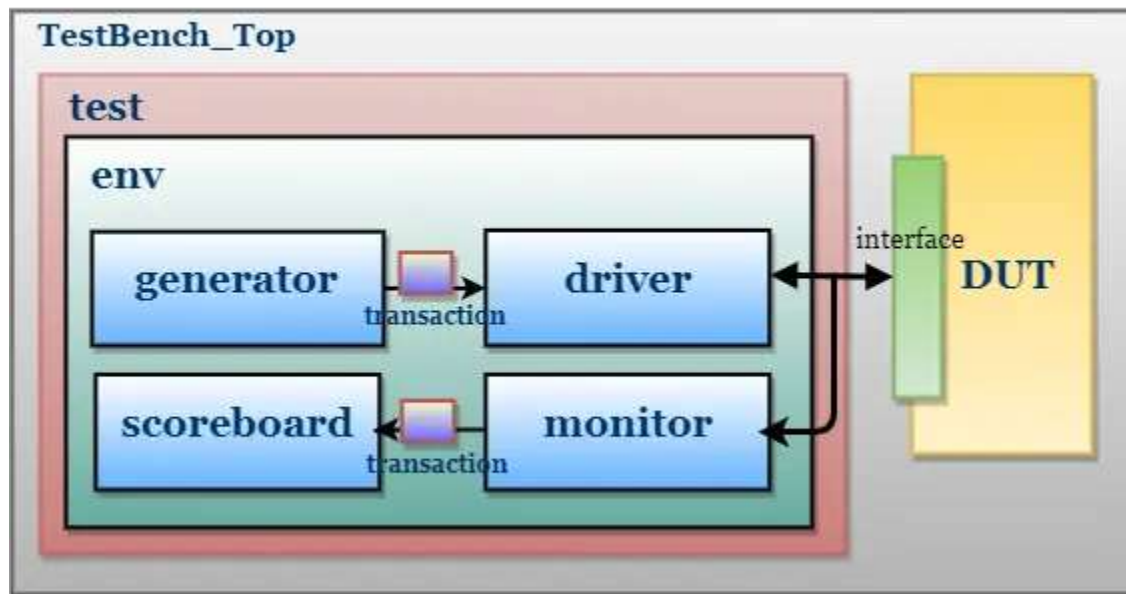
No. of data items can be read in a period of 2132.992ns = $(2132.992/7.5) = 284.398 = 284$

Remaining no of bytes to be stored in the FIFO = $512 - 284 = 228$

The minimum depth of the FIFO should be 228

III. Verification Environment

Verification Environment consists of Generator, Driver, Monitor and Scoreboard, which are connected via mailboxes. All these components inside environment are nothing but classes.



Generator: This class initiates stimuli generation for testing. It produces diverse input scenarios to thoroughly exercise the FIFO's functionality, including normal operation, edge cases, and error conditions.

Driver: Responsible for translating abstract stimuli from the Generator into concrete signals compatible with the FIFO's interface. It ensures accurate application of stimuli to the FIFO inputs, simulating real-world operation and verifying the FIFO's responses.

Monitor: The Monitor class keeps a close eye on how the asynchronous FIFO behaves during simulation. It watches signals coming in and going out of the FIFO, as well as signals inside it, to spot any mistakes or problems. It pays attention to things like when

data is moved, when the FIFO gets full or empty, or if any errors happen. By keeping track of these details, the Monitor helps us understand how the FIFO works, making it easier to check if everything is running smoothly.

Scoreboard: Compares expected FIFO behavior with observed behavior during simulation. It acts as the reference model, predicting expected outputs based on inputs and current FIFO state, identifying discrepancies or errors to highlight potential issues.

Interface: The Interface module in SystemVerilog defines a set of rules for communication between the verification environment and the Design Under Test (DUT). It encapsulates the connections and signals required for interaction, simplifying the integration process.

Interconnected via mailboxes, these modules facilitate communication and coordination, enabling seamless stimulus generation, signal translation, behavior observation, and outcome comparison, ensuring comprehensive verification of the asynchronous FIFO design.

IV. Required Tools

QuestaSim is used for SystemVerilog simulation environment and debugging for the project. Verification is done through transcripts, and waveforms.

V. Verification Plan

- Write data to the FIFO. Check to write FIFO full and half full condition.
- Read data from the FIFO. Check for read FIFO empty and half empty condition.
- Verify write and read clock periods. Verify FIFO reset functionality.
- Designed the functionality of an asynchronous FIFO across various clock domains by giving **randomized** testcases.
- Handling of metastability and synchronization between the write and read clock domains.
- Verified the proper reset and initialization of the FIFO and its internal signals.
- Performed write and read operations with different burst sizes and idle cycles, as per the design specifications.

- The write operation is performed successfully ensuring that FIFO accepts the data whenever the write signal is enabled, and the data is stored.
- The read operation is performed successfully ensuring that FIFO sends valid data whenever the read signal is enabled, and the data is retrieved.
- FIFO full condition is verified by ensuring data is written into FIFO till maximum depth is reached.
- FIFO empty condition is verified by making sure that the data read is done from the FIFO until it becomes empty.
- With a class based testbench we have checked the functionality of the design.

VI. Contributions

Suhail Ahamed Subairudeen: Calculations for the project based on design specifications, writing code for half-empty, half-full, full, and empty conditions.

Sai Rohith Reddy Yerram: Writing the design based on the specifications.

Rakshita Joshi: Writing class based testbench, creating verification plan document.

Srikar Varma Datla: Writing synchronizers, run.do, debugging the files.

VII. Resources

- 1.http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf
- 2.http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO2.pdf
- 3.<https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdf>
- 4.<https://ieeexplore.ieee.org/abstract/document/7237325>

VIII. Additional Comments:

We will keep on adding into the verification plan as the Milestones progress and will keep modifying the testcases to verify the functionality of the design as we keep achieving the goals for subsequent milestones.