ECE-593

FUNDAMENTALS OF PRE-SILICON VALIDATION

Portland State UNIVERSITY

FINAL PROJECT

Milestone-4

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

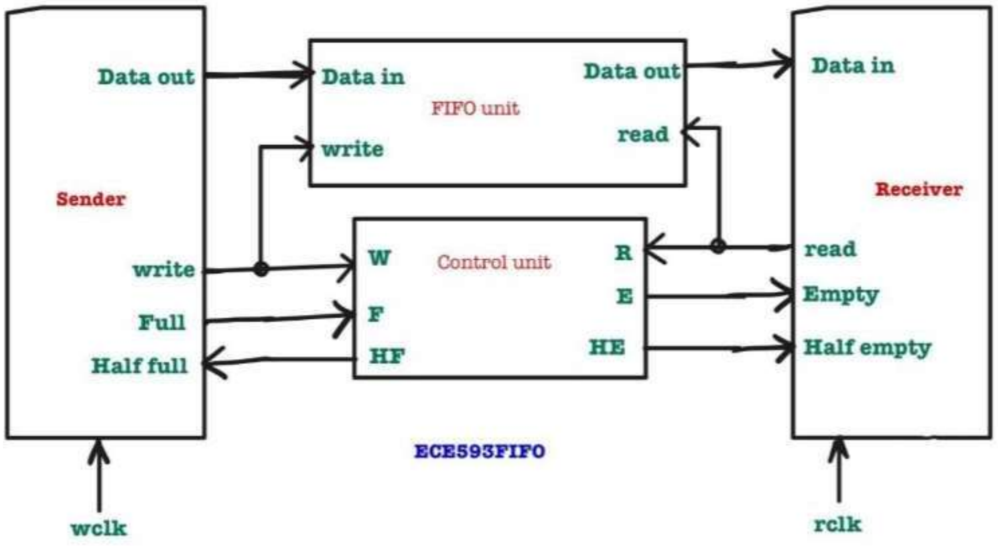
I. Design Overview

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.



High Level Block Diagram of the Design System

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 on data is read.

Time required to write one data item = 1 * (1/240MHz) = 4.166 ns. Time

required to write the data in the burst = 512*4.166 = 2132.992ns. Time

required to read one data item = 3 * (1/400MHz) = 7.5ns

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. Required Tools

QuestaSim is used for simulation environment and debugging for the project. Verification is done through transcripts, and waveforms, and using known good module, checking through scoreboard.

IV. Milestone 4 updates

We are transitioning to UVM environment from class based methods, incorporating UVM components, fulfilling the Milestone-4 requirements as mentioned by professor.

V. Verification Environment

UVM Testbench Architecture Development

The Universal Verification Methodology (UVM) testbench architecture is the foundational step for creating a robust verification environment. The architecture includes defining the various components and their interactions to facilitate efficient and reusable verification of the design under test (DUT). The primary components of the UVM testbench architecture are:

Test:
- Top-level component controlling the simulation and initiating the test sequences.
- Defines and configures different test scenarios.
- Sets up the environment, starts sequences, and manages simulation flow.

Environment:
- Encapsulates all other components, providing a hierarchical structure.
- Contains agents, monitors, scoreboards, and other verification components.
- Ensures communication and coordination among components.

Agent:
- Represents a verification unit that includes a sequencer, driver, and monitor.
- Handles a specific interface or protocol of the DUT.
- Write Agent and Read Agent included with full, half-full, empty, half-empty conditions.

Sequencer:
- Controls the flow of stimulus sequences to the driver.
- Manages the ordering and timing of sequences.
- Coordinates with the driver to execute sequences correctly.

Driver:

- Converts sequences into actual pin-level signals for the DUT.
- Takes instructions from the sequencer.
- Simulates real-world interactions with the DUT.

Monitor:

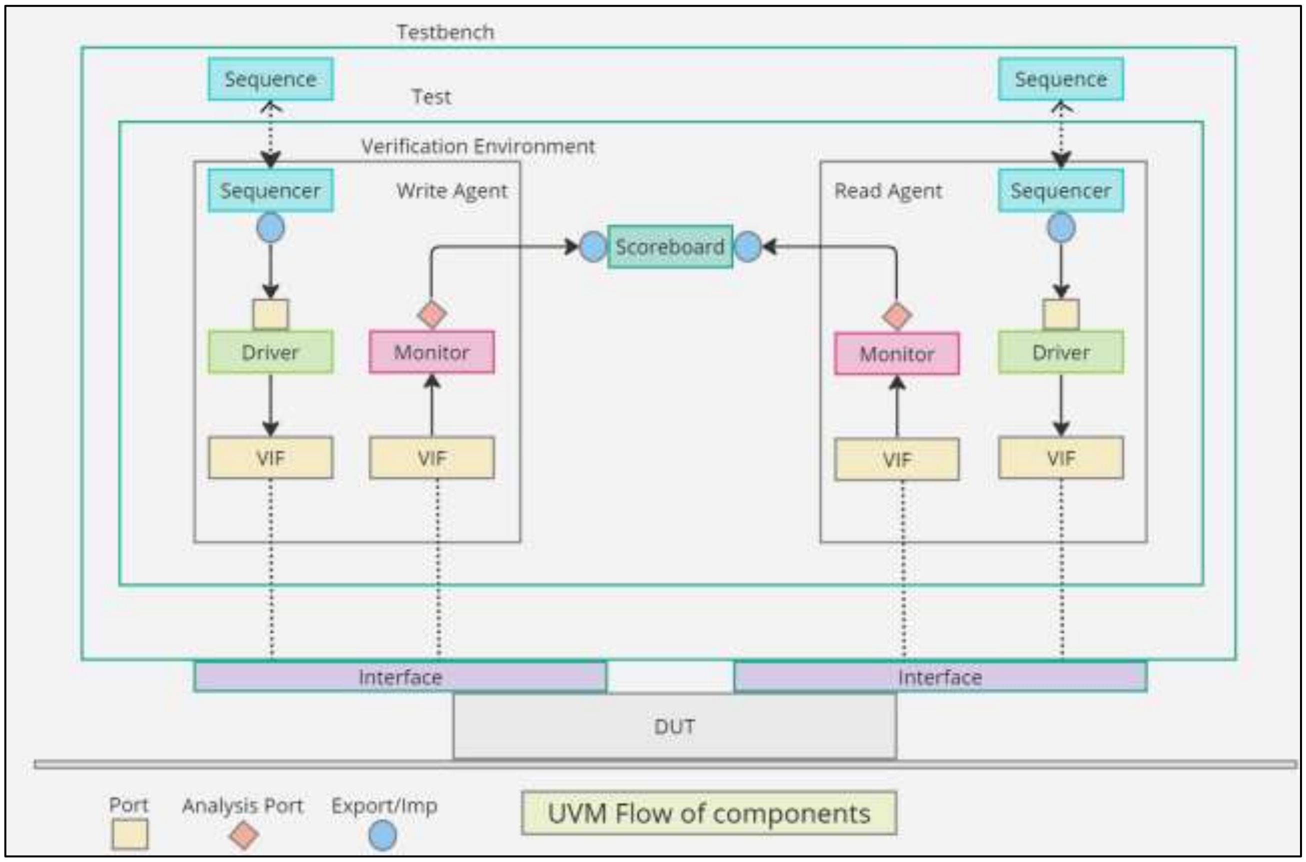
- Observes DUT signals and collects data for analysis.
- Operates passively, capturing DUT responses to stimuli.
- Sends collected data to other components, such as the scoreboard.

Scoreboard:

- Analyzes and compares DUT outputs with expected results.
- Performs checks to ensure DUT behavior meets specifications.
- Identifies mismatches and logs errors.

Interface:

- Connects testbench components with the DUT.
- Defines signal-level connections for communication.
- Includes signal declarations and bus functional models (BFMs).



VI. Verification Plan

Testcase scenario

- 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 the universal verification methodology, we have checked the functionality of the design.

VII. Contributions

Suhail Ahamed Subairudeen: Calculations for the project based on design specifications, writing flag conditions, including UVM components

Sai Rohith Reddy Yerram: Writing the design based on the specifications, writing UVM components

Rakshita Joshi: UVM testbench creation, creating verification plan document.

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

VIII. 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/fifodepthcalculationm adeeasy2.pdf
4. https://ieeexplore.ieee.org/abstract/document/7237325

IX. Additional Comments:

We strive to include more test cases and make our design more robust by the final project presentation. We will incorporate coverage reports in our final project presentation, aiming to achieve 100% code and functional coverages.