Fundamentals of Pre-Silicon Validation Winter-2025

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

VERIFICATION TEST PLAN

Team -1

Bhargav Chunduri - bhargavc@pdx.edu

Dhushyanth Dharmavarapu – dharmava@pdx.edu

Venkata Krishna Kumar vedantam – vedantam@pdx.edu

CONTENTS **Page No**

1. Introduction

1.1 Purpose of the verification:

The purpose of the verification plan for the asynchronous FIFO design is to methodically assess its functionality, performance, and reliability. The key objectives include:

Functional Verification

- Ensuring the FIFO correctly follows the First-In-First-Out (FIFO) data movement principle
- Validating the proper functioning of FIFO full and FIFO empty conditions.
- Checking that the read and write pointers operate correctly and meet the required conditions.

Asynchronous Operation

- Ensuring data integrity between the write and read domains.
- Verifying that the design supports asynchronous data transfer, accommodating different clock frequencies between the read and write interfaces. Metastability Condition
- Assessing the design's ability to handle metastability issues caused by asynchronous inputs and confirming the implementation of proper synchronization techniques.

Cross-Clock Domain Signals

- Detecting and verifying signals that transition between clock domains, including handshaking signals and FIFO status flags..

The verification plan aims to achieve comprehensive coverage of the asynchronous FIFO's functionality, ensuring a high level of confidence in its accuracy, reliability, and performance across various usage scenarios.

1.2 Specifications for the design

Sender Clock Frequency = 500MHZ = fc

Write Idle Cycles = 2 (Data will be written on every consecutive clock cycle)

Write Burst Size = 1024

- Receive Clock Frequency = 225 MHz = fB

  Read Idle Cycles = 1 (A single data item is read every three clock cycles.)

  Since, fc > fB and

  Given Burst Length = 1024

- The number of idle cycles between two consecutive writes is 2 clock cycles. This means that after writing one data item, module A waits for two clock cycles before initiating the next write. Therefore, it can be understood that one data is written every three clock cycles.

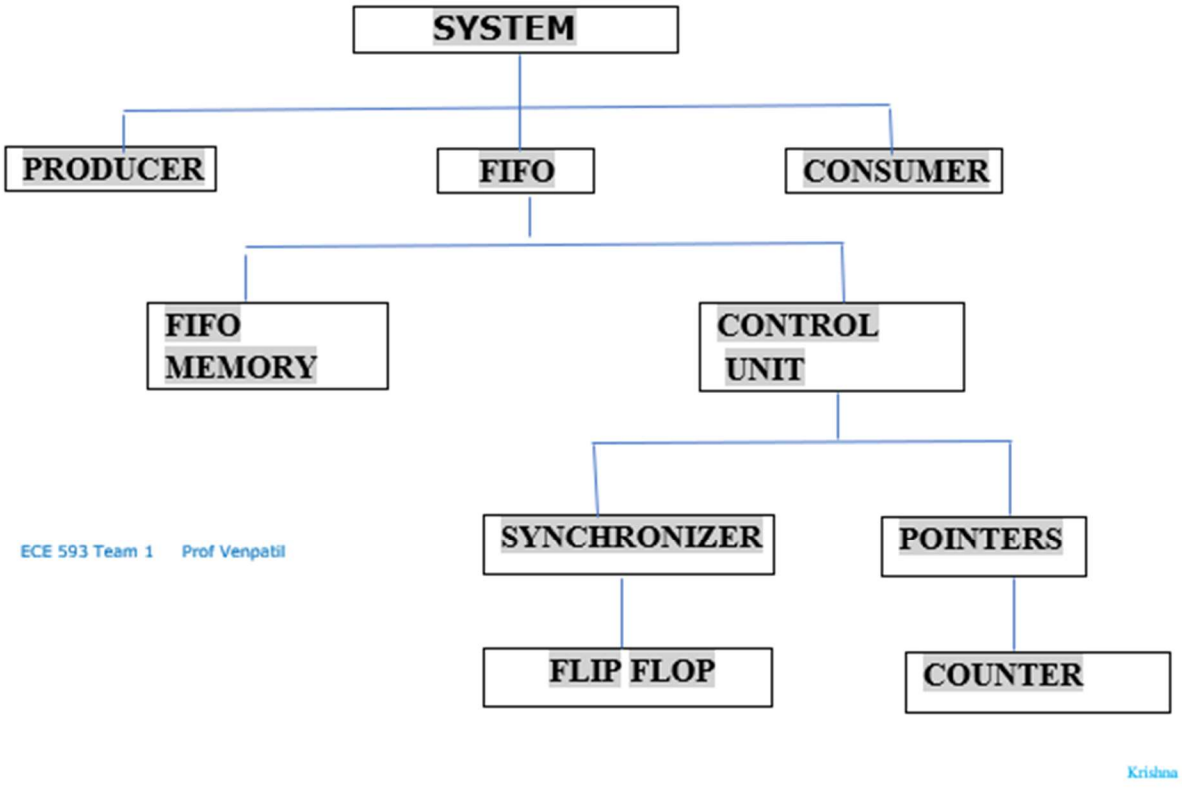
- The number of idle cycles between two consecutive reads is 1 clock cycle. This means that after reading one data item, module B waits for one clock cycle before initiating the next read. Therefore, it can be understood that one data is read every two clock cycles.

- The Time required to write one data item = 3 * 1/500 MHz = 6nS
- The Time needed to write a complete burst of data.= 1024* 6 nS. = 6,144 nS
- The time needed to read a single data item is 2 * 1/225 MHz, which equals 9 ns. Therefore, every 9 ns, module B reads one data item from the burst.
- Within a duration of 6144 ns, 1024 data items can be written.
- The number of data items that can be read within 6144 ns is calculated as (6144 ns / 9 ns) = 682.
- The remaining number of bytes to be stored in the FIFO is 1024 - 682 = 342.
- Hence, the minimum required depth of the FIFO is 342.

2. Verification Requirements

2.1 Verification Levels

The verification is performed at the top level.



3 Required Tools

3.1 List of necessary software and hardware tools.

Software Tool sets:

QuestaSim

Windows OS

3.2 The organization of run directories and the computer resources utilized.

Source Code :

- Asynch_fifo_top.sv
- fifomem.sv
- writeptr_full.sv
- readptr_empty.sv
- syncread2w.sv
- syncwrite2r.sv
- interface.sv

Testbench code:

- trans_fifo.sv
- gen_fifo.sv
- driv_fifo.sv
- mon_fifo.sv
- scb_fifo.sv
- environment.sv
- testbench.sv
- test.sv
- fifo_pkg.sv

Script :

- run.do

4. Tests and Methods

Types of testing methods to be applied: Black Box, White Box, and Gray Box.

Black Box Testing:

- Verifying asynchronous operation by performing concurrent read and write actions.
- Ensuring proper assertion of flags in the defined scenarios..

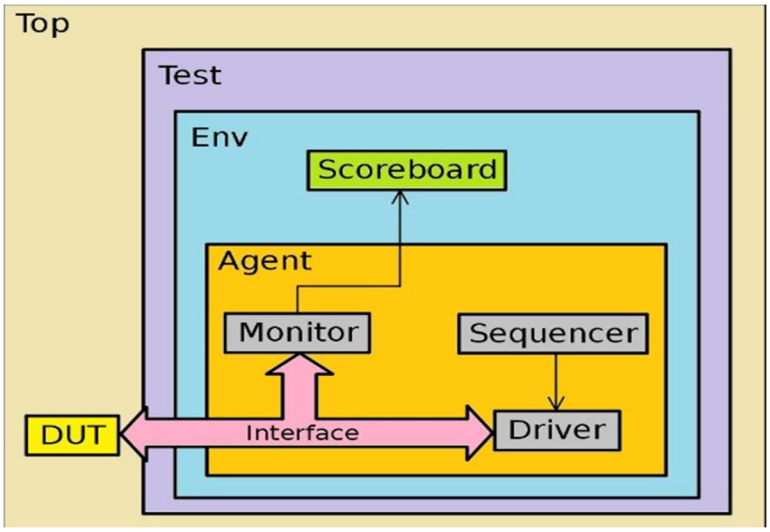
- Validating data integrity during read and write operations.
- Checking that the burst length fits within the FIFO as per specifications
- Confirming that the design functions at the specified frequency and accurately generates the expected waveforms.

White Box Testing:

- In future testbenches, assertions will be utilized to access internal variable values for testing.

4.1. Testbench Architecture:

Components used, including Drivers, Monitors, Scoreboards, Checkers, etc.
Below Block Diagram is taken from internet source



5. Verification Strategy:

Simulation was selected for functionality testing due to the design's nature and the flexibility it offers for updating testbenches in future milestones.

5.1 Verification Plan:

5.1.1 Functional Verification:

- The functional verification process for an asynchronous FIFO includes efficient testing of key features such as write/read operations, overflow/underflow conditions, and correct handling of control signals.
- Special attention is given to ensuring proper synchronization and addressing potential metastability issues by accounting for the asynchronous behavior between clock domains. To assess performance and robustness, test cases cover various scenarios, including boundary conditions and stress tests
- The proper functioning of the FIFO is validated, and the verification environment is established using simulation tools.

5.2 Corner Case Testing:

5.2.1 Test Case Scenarios:

Test Full:

This test case involves filling the FIFO by writing data to all available locations until it reaches its capacity. The verification process includes tracking the FIFO's full flag or status to ensure it correctly indicates when the FIFO is full. Additionally, the FIFO's response to attempts to write data when full should be examined. It should either prevent further writes, trigger an error, or follow some other predefined behavior.

Test Empty:

this test case, the FIFO starts off empty, and data is read from it until it becomes empty once more. Similar to the "Test Full" scenario, the empty flag or status of the FIFO must be monitored to ensure it accurately reflects when the FIFO is empty. Additionally, the FIFO's response to attempts to read from it when empty should be observed, such as blocking reads, generating an error, or following a specific behavior.

Test Full Error:

This test case checks the FIFO's response when attempting to write data while it is already full. The expected outcome is that the FIFO should either trigger an error, assert a full flag, or use another method to indicate that the write operation cannot be completed because the FIFO is full.

Test Empty Error:

Similar to the "Test Full Error" case, this test case checks the FIFO's response when attempting to read data while it is empty. The expected behavior is that the FIFO should either trigger an error, assert an empty flag, or use another method to indicate that the read operation cannot be executed because the FIFO is empty.

6. Coverage Metrics:

- For an Asynchronous FIFO, the three main coverage metrics are functional coverage, code coverage, and assertion coverage.
- Functional coverage tracks how effectively test scenarios address all required behaviors of the FIFO, while code coverage measures the percentage of code lines, branches, and conditions tested by the verification, ensuring comprehensive testing of the design.
- Assertion coverage evaluates how well assertions capture design properties and detect violations. These metrics help assess the thoroughness of the verification process, identify areas needing additional testing, and gauge confidence in the accuracy and robustness of the FIFO implementation.

7. Verification Environment:

Developing a class-based verification environment for an Asynchronous FIFO involves multiple steps to ensure comprehensive testing and validation of the design.

7.1 Testbench Architecture:

- Define the overall testbench structure using class-based verification, incorporating components like generators, drivers, monitors, and scoreboards.
- Organize the testbench hierarchy to promote modularity, scalability, and reusability.
- Implement distinct transactions for the FIFO’s input and output interfaces, encapsulating the functionality of the driver and monitor for each interface.
- Configure the packets to interface with the FIFO design, managing data transactions, protocol validation, and synchronization between clock domains.

Mailbox:

- A mailbox is used to transfer data synchronously between different elements of the testbench.

Transaction Class:

- It includes inputs as random variables that can be randomized from other classes or functions using the randomize SV keyword.

Interface:

- An interface is used to group all the design ports together as a bundle, making all input and output signals accessible in one place
- A **virtual interface** is employed to link the dynamic testbench architecture with a static DUT.

Generator:

- Create packets to generate stimuli for write and read operations, addressing different scenarios like empty, full, overflow, and underflow conditions.
- Implement tasks and events to manage the generation and scheduling of sequence items, ensuring proper synchronization and coordination with other parts of the testbench.

Driver:

- Create drivers to provide stimulus to the FIFO's input interface, including data, control signals, and timing constraints.

Monitor

- Develop monitors to capture and analyze transactions on the FIFO's output interface, ensuring data integrity, protocol adherence, and timing constraints.

Scoreboard:

- Set up a scoreboard to compare the expected and actual behavior of the FIFO, ensuring the accuracy and thoroughness of the test results.
- Validate data consistency, FIFO occupancy, and the interaction of control signals between the input and output interfaces.

Environment:

- The environment encompasses all components of the testbench architecture, instantiating all classes and mailboxes.
- All primary tasks in the classes are invoked within the environment to ensure synchronization.

Test:

- The primary objective of the test is to synchronize the dynamic environment with the testbench.
- It is written as a program rather than a module or function.

Testbench:

- The testbench links the design with the testbench architecture.
- It instantiates the interface, test, and DUT, and connects them based on the required ports for mapping.

Coverage:

Coverage in the context of verification refers to the measure of how thoroughly a design or system has been tested. It quantifies the extent to which various aspects of the design, such as conditions, transitions, or operations, have been exercised or triggered during simulation. Coverage helps ensure that all critical parts of the design have been tested, providing confidence in its correctness and functionality.

There are different types of coverage in hardware design verification, such as:

- **Code Coverage**: Measures how much of the design's code (e.g., RTL) has been exercised during testing. It checks if all the statements, conditions, and branches in the code have been executed at least once.
- **Functional Coverage**: Focuses on verifying that all specified behaviors of the design have been tested. This involves tracking the functional states and transitions of the design, such as various input combinations and expected outputs, to ensure that the design meets its functional requirements.
- **Toggle Coverage**: Ensures that all the signals (e.g., flip-flops) in the design have been toggled during testing, which helps verify that the circuit's logic has been thoroughly exercised.

In short, coverage is a critical metric that helps identify untested areas of a design and ensures comprehensive verification.

Stimulus Generation:

Inputs for testing the FIFO are generated using randomization and constraints.

8. Resources requirements

Team members and tasks allocation:

- **Bhargav Chunduri:** He will be focusing on implementing the core functionality of the asynchronous FIFO design. He will refine the write functionality, ensuring it works with the specified idle cycles. Bhargav will also design and test the FIFO memory, as well as ensure the correct operation of the generator and monitor modules. In addition, he will contribute to drafting the verification plan document to ensure that all write-related features are adequately tested.

- **Dhushyanth Dharmavarapu :** He will be in charge of implementing and verifying the read functionality of the asynchronous FIFO design, ensuring it meets the requirement of one read idle cycle. He will thoroughly test the design with various test cases, write the Design Specification document, and work on the driver and scoreboard functionalities to ensure the read domain operates as expected.

- **Venkata Krishna Kumar Vedantam :** He will lead the integration efforts, implementing the design through interfaces and ensuring the correct connectivity with the top module. He will calculate and document the FIFO depth, write the scoreboard, and verify the percentage thoroughly. Venkata will also create the overall test environment, develop tests, and set up the testbench top module. Throughout the process, he will collaborate with the other team members to ensure smooth integration and functionality.

9. REFERENCES:

- http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf
- The UVM Primer
- https://github.com/teekamkhandelwal/asynchronous_fifo/blob/main/r_pointer_epty.v