Fundamentals of Pre-Silicon Validation || Spring -2024

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

VERIFICATION TEST PLAN

Project Team:
Jaswanth Pallappa, jaswanth@pdx.edu
Brunda Marpadaga, brunda@pdx.edu
Maha Lakshmi Potabattuni, mahalak@pdx.edu
Akhila Veedula, akhilav@pdx.edu

Acknowledgement

The block diagram and the implementation of gray code counters was taken from the paper "Clifford E. Cummings and Peter Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, March 2002, Section TB2, 3rd paper." and modified for this specifications of the project. Also available at www.sunburst-design.com/papers

1. Introduction

1.1 Objective of the verification:

The objective of the verification plan for the asynchronous FIFO design is to systematically validate the functionality, performance, and robustness of the FIFO implementation. The primary goals are as follows:

Functional Verification –

- Verifying FIFO works properly for the movement of data based on first in first out.
- Verifying fifo-full, fifo-empty conditions work properly.
- Verifying read pointer, write pointer is satisfying the conditions.

Asynchronous Operation –

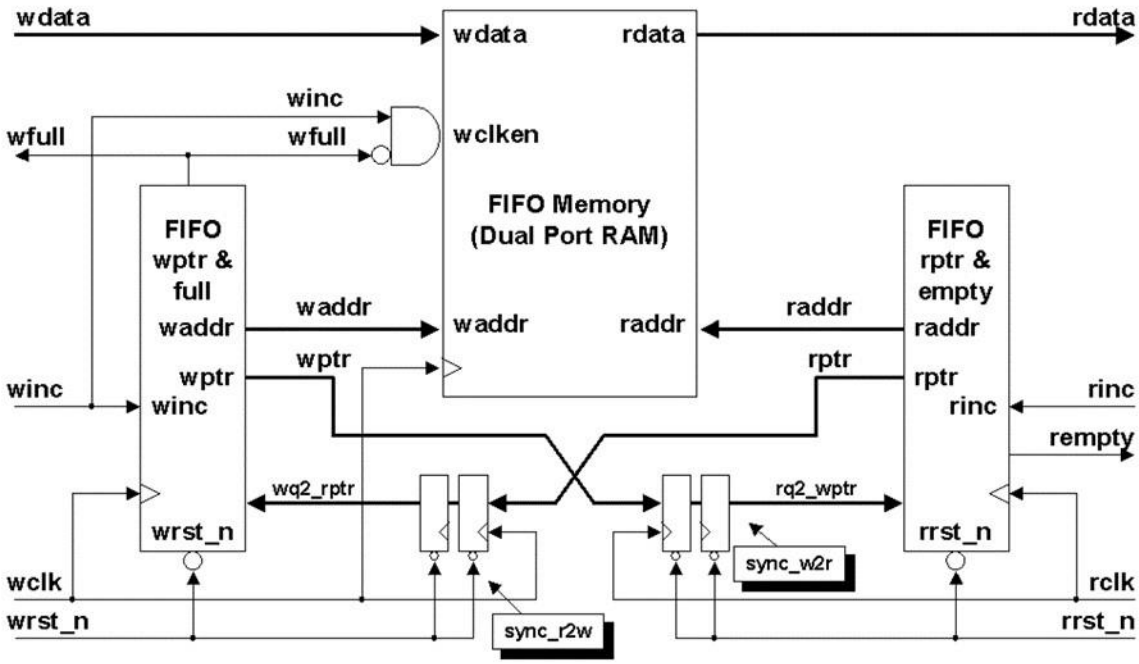
- Validating the data is the same between write and read domains.
- Verify that the design handles data transfer asynchronously, allowing for variations in clock frequencies between the read and write interfaces. ***Metastability Condition***
- Evaluate the design's capability to manage metastability challenges arising from asynchronous inputs and confirm the integration of suitable synchronization methods.

Cross-Clock Domain Signals –

- Identify and validate signals that cross clock domains, such as handshaking signals or flags indicating FIFO status.

The verification plan aims to provide comprehensive coverage of the asynchronous FIFO's functional and ensure a high level of confidence in its correct operation and performance across diverse usage scenarios.

1.2 Top Level block diagram



1.3 Specifications for the design

- Sender Clock Frequency = 500 MHz = fA

  Write Idle Cycles = 2 (For every consecutive clock cycle, data will be written) Write Burst Size = 1024

- Receive Clock Frequency = 225 MHz = fB

Read Idle Cycles = 1 (For every 3 clock cycles, single data is read)

Since, fA > fB and
Given Burst Length = 1024

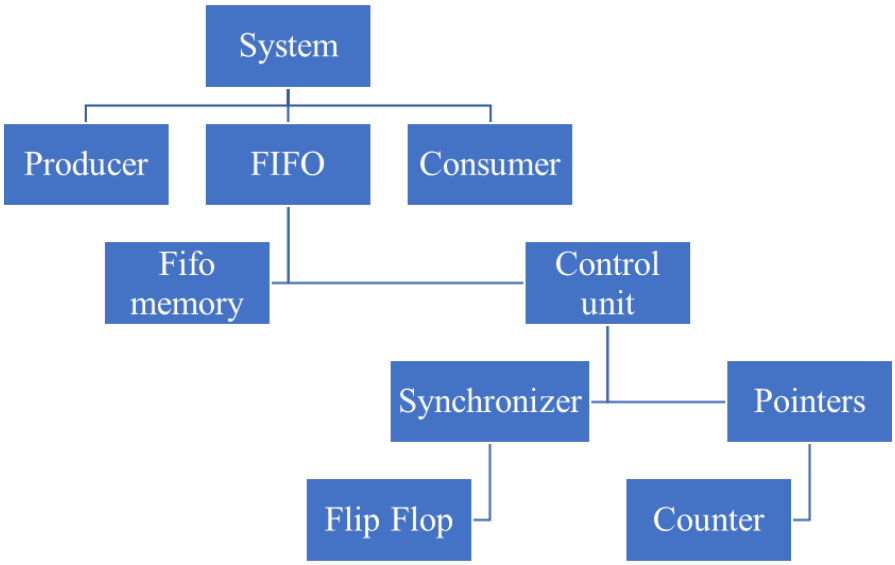
- The no. of idle cycles between two successive writes is 2 clock cycle. It means that, after writing one data, module A is waiting for two clock cycle, to initiate the next write. So, it can be understood that for every **three** clock cycles, one data is written.
- The no. of idle cycles between two successive reads is 1 clock cycle. It means that, after reading one data, module B is waiting for one clock cycle, to initiate the next read. So, it can be understood that for every **two** clock cycles, one data is read.
- Time required to write one data item = 3 * 1/500 MHz = 6nS
- Time required to write an entire burst of data = 1024* 6 nS. = 6,144 nS
- Time required to read one data item = 2 *1/225 MHz = 9 nS • So, for every 9 nS, the module B is going to read one data in the burst.
- In a period of 6144 n sec, 1024 number of data items can be written.

- Number of data items can be read in a period of 6144 n sec = (6144 n sec/9) = 682
- Remaining number of bytes to be stored in FIFO = 1024 – 682 = 342
- Therefore, minimum depth of FIFO = 342

2 Verification Requirements

2.1 Verification Levels

Verification is done at top level.



3 Required Tools

3.1 List of required software and hardware tool sets needed.

Hardware tool sets :

None

Software Tool sets:

QuestaSim

Windows OS

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

Source Code :

- top.sv
- fifo_memory.sv
- write_ptr.sv

- read_ptr.sv
- syncr2w.sv
- syncw2r.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

Scripts :

run.do

4 Risks and Dependencies

5 Functions to be Verified.

6 Tests and Methods

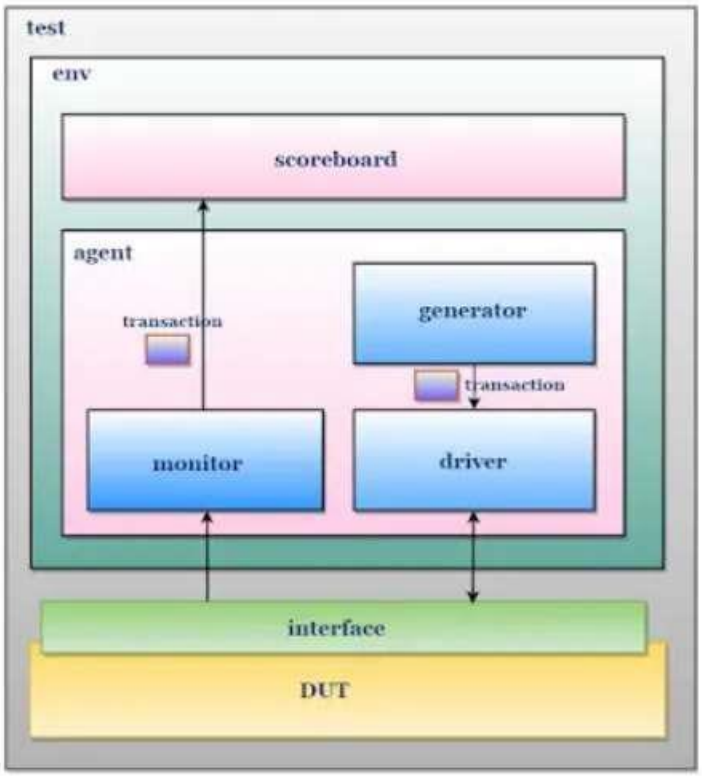
6.1.1 Testing methods to be used: Black/White/Gray Box.

Black Box Testing:

- Testing the asynchronous operation with concurrent read and write operations
- Testing the flags is asserting properly in the specified scenarios.
- Testing the read and write operations for data integrity.
- Testing the burst length is going into fifo within the given specifications.
- Testing the design is working in the specified frequency and able to generate the exact waveforms related to the design **White Box Testing:**
- For Assertions in future testbenches, this testing will be used to have access to internal variable values.

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

For Milestone 2, we plan to use the testbench architecture shown below. More details will be provided with Milestone 2 submission.



6.1.3 Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.)

We chose simulation for testing the functionality considering the nature of the design and the ease of upgrading the testbenches for future milestones.

VERIFICATION TEST PLAN:

Functional Verification:

- Efficient testing of fundamental features such as write/read operations, overflow/underflow scenarios, and proper control signal handling is part of the functional verification process for an asynchronous FIFO.
- Extra care is taken to ensure correct synchronization and handle any possible metastability problems by considering the asynchronous behavior between clock domains. To verify

performance and resilience, test cases include a variety of scenarios, such as boundary conditions and stress testing.

- The correct operation of the FIFO is confirmed, and verification environments are set up with the aid of simulation tools.

Corner Case Testing:

Test Case Scenarios:

Test Full: This test case involves writing data to all available locations within the FIFO until it becomes full. The verification process includes monitoring the FIFO's full flag or status to confirm that it accurately reflects when the FIFO is indeed full. Additionally, the behavior of the FIFO when attempting to write data when it is full should be observed. It should either block further writes, generate an error, or employ some other predefined behavior.

Test Empty: In this test case, the FIFO is initially empty, and data is read from it until it becomes empty again. Similar to the "Test Full" case, the empty flag or status of the FIFO needs to be monitored to ensure it correctly indicates when the FIFO is empty. Also, the behavior of the FIFO when attempting to read from it when it is empty should be observed, such as blocking reads, generating an error, or employing a specific behavior.

Test Full Error: This test case verifies the behavior of the FIFO when attempting to write data to it while it is already full. The expected behavior here is that the FIFO should either raise an error, assert a full flag, or employ some other mechanism to indicate that a write operation cannot be performed due to the FIFO being full.

Test Empty Error: Similar to the "Test Full Error" case, this test case verifies the behavior of the FIFO when attempting to read data from it while it is empty. The expected behavior is that the FIFO should raise an error, assert an empty flag, or use some other mechanism to indicate that a read operation cannot be performed due to the FIFO being empty.

Test Concurrent Write-Read: This test case involves concurrently writing and reading data to/from the FIFO. It verifies the concurrent operation of both read and write ports of the FIFO and ensures that there are no race conditions or other synchronization issues. The behavior of the FIFO under simultaneous read and write operations should be observed to ensure correct operation.

Coverage Metrics:

- Functional coverage, code coverage, and assertion coverage are the three types of coverage metrics for an Asynchronous FIFO.
- Functional coverage monitors how well test scenarios capture all the necessary behavior of the FIFO, while code coverage counts the proportion of code lines, branches, and conditions that are tested by the verification tests, guaranteeing thorough testing of the design.
- Assertion coverage assesses how well assertions capture design properties and identify violations. These metrics help determine how thorough the verification effort was, where more testing might be needed, and how confident one can be in the accuracy and resilience of the FIFO implementation.

Verification Environment:

We create a class based verification environment for an

Asynchronous FIFO involves several steps to ensure thorough testing and validation of the design.

Testbench Architecture:

- Define the overall testbench architecture following the class-based verification, including components such as generators, drivers, monitors, and scoreboards.
- Organize the testbench hierarchy to facilitate modularization, scalability, and reusability.
- Implement separate transactions for the FIFO's input and output interfaces, encapsulating the functionality of the driver, monitor for each interface.
- Configure the packetsto interface with the FIFO design, handling data transactions, protocol checks, and synchronization between clock domains.

Transaction Class:

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

Mailbox:

- Mailbox is used to transfer data from one element of testbench to another element synchronously.

Interface:

- An interface is used to group all the ports of the design together as a bundle. It makes all the input and output signals available at a single place.
- ***Virtual interface*** is used to connect the dynamic testbench architecture with a non-dynamic DUT.

Generator:

- Develop packets to generate stimulus for write and read operations, covering various scenarios such as empty, full, overflow, and underflow conditions.
- Implement tasks and events to control the generation and scheduling of sequence items, ensuring proper synchronization and coordination with the rest of the testbench.

Driver and Monitor:

- Create drivers responsible for driving stimulus to the FIFO's input interface, including data, control signals, and timing constraints.
- Develop monitors to capture and analyze transactions on the FIFO's output interface, verifying data integrity, protocol compliance, and timing constraints.

Scoreboard:

- Implement a scoreboard to compare the expected and observed behavior of the FIFO, ensuring correctness and completeness of test results.
- Verify data consistency, FIFO occupancy, and control signal interactions between the input and output interfaces.

Environment:

- Environment contains all the elements of a testbench architecture. It instantiates all the classes and mailboxes.
- All the main tasks in the classes are called in the environment for synchronization.

Test:

- The main purpose of the test is to synchronize the dynamic environment to the testbench.
- It is written in form of program rather than module or function.

Testbench:

- The testbench connect the design with the testbench architecture.
- It instantiates interface, test and DUT and connects them according to the required port mapping.

Stimulus Generation:

Inputs are generated using randomization and constraints

9. Resources requirements

Team members and tasks allocation:

- **Brunda**: She will focus on implementing and testing the core functionality of the asynchronous FIFO design. This includes refining the write functionality with two write idle cycles idle cycles according to specifications, designing and testing the FIFO memory, and ensuring the correct functionality of the generator and monitor modules. Additionally, she will contribute to writing the verification plan document and providing support in other areas as needed.

- **Jaswanth:** He will be responsible for implementing and verifying the read functionality of the asynchronous FIFO design, ensuring one read idle cycle as per requirements. He will thoroughly test the design with various test cases, write the Design Specification document, and develop the driver and scoreboard functionality. He will also contribute to testing other modules and aiding as required.

- **Maha Lakshmi**: She will lead the integration efforts by implementing the design through interfaces and ensuring proper connectivity with the top module. Additionally, she will calculate and document the depth of the FIFO, write the scoreboard, and verify the percentage thoroughly. She will also work on creating the overall environment, tests, and testbench top module. She will collaborate with other team members to ensure seamless integration and functionality.

- **Akhila Veedula:** Akhila will join the team to assist in various tasks related to the asynchronous FIFO design verification. Her responsibilities will include contributing to the implementation and testing of both read and write functionalities, supporting the development of test cases, and assisting in documentation tasks. Akhila will work closely with coverage and assertion aspects of the verification plan ensuring the overall development of the project.

.

REFERENCES:

- **https://github.com/teekamkhandelwal/asynchronous_fifo/blob/main/r_pointer_epty.v**
- **http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf**
- **The UVM Primer**
- **https://www.youtube.com/watch?v=eeU2zpgXv1A&list=PLigQ6Cc3qFpI_WTgqtDXi_Msk3yRuKGGJ**