VERIFICATION TEST PLAN

Fundamentals of Pre-Silicon Validation
Winter -2024

Project Name: Design and Verification of Asynchronous FIFO

Members: Shamshi, Sujith Krishna, Prasanna Korlapati

Date: 02/03/2024

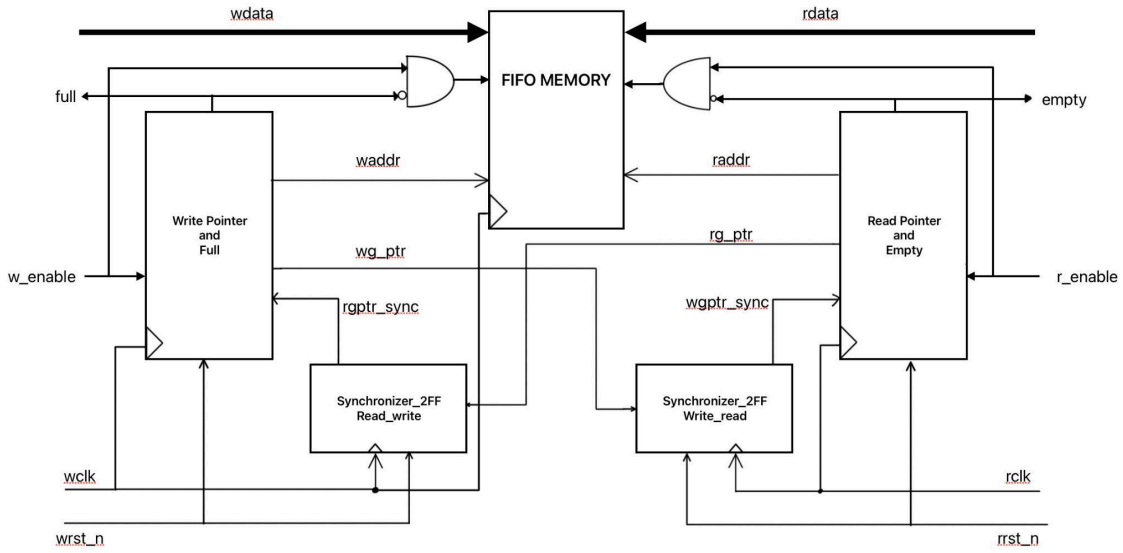
Table of Contents

2 Introduction:

2.1 Objective of the verification plan

- *To create a comprehensive verification plan for an asynchronous FIFO design using SystemVerilog constructs and UVM methodology.*
- *To ensure the correct functionality, robustness, and timing behavior of the FIFO under various operating conditions and potential error scenarios.*

2.2 Top Level block diagram



2.3 Specifications for the design

- ➔ *Data width: 8 bits*
- ➔ *FIFO depth: 228 data items*
- ➔ *FIFO width: $\log_2(228) = 8$ (rounded)*
- ➔ *Clock domains: Producer Frequency - 240MHz, Consumer Frequency - 400MHz*
- ➔ *Synchronization mechanism: At every rising edge of rclk passing Gray code converted Write pointer through 2FF synchronizers to readpointer and Gray code converted Read pointer to Write pointer at every rising edge of wclk.*
- ➔ *Reset behavior: Active Low Reset for all design blocks (producer, consumer and synchronizer).*
- ➔ *Metastability handling: Using 2FF synchronizers between producer and consumer.*
- ➔ *Timing requirements: wclk - 4.17ns and rclk - 7.5ns*

3 Verification Requirements

3.1 Verification Levels

3.1.1 What hierarchy level are you verifying and why?

- *Top-level verification of the entire asynchronous FIFO module.*
- *Justification: Ensures integration of all components and interactions between clock domains.*

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

- *Controllability: Achieved through UVM sequences and drivers for write and read ports.*
- *Observability: Monitors for data, control signals, and status flags. Scoreboard for data comparison.*

3.1.3 Are the interfaces and specifications clearly defined at the level you are verifying?

- *Clearly defined interfaces for write and read ports, control signals, and status flags.*
- *Timing specifications for clock domains and synchronization signals.*

4 Required Tools

4.1 List of required software and hardware tool sets needed.

- *SystemVerilog simulator with UVM support (e.g., QuestaSim, VCS, Xcelium)*
- *UVM library*
- *Coverage tools (e.g., Questa CoverCheck, Synopsys VCS Coverage)*
- *Waveform viewer*

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

- *Directory structure: We'll use Git as our directory.*
- *Computer resources: We'll use a laptop that has Intel core I5 10th generation processor that will run QuestaSim version 10.7 to simulate and verify our design.*

5 Risks and Dependencies

5.1 List all the critical threats or any known risks. List contingency and mitigation plans.

Risks:

- ➔ *Metastability issues*
- ➔ *Synchronization failures*
- ➔ *Timing violations*
- ➔ *Race conditions*

Contingency and Mitigation Plans:

- ➔ *Thorough verification of metastability handling mechanisms*

- ➔ *Robust synchronization logic*
- ➔ *Static timing analysis*
- ➔ *Careful design of clock domain crossing strategies*

6 Functions to be Verified.

6.1 Functions from specification and implementation

- ➔ *Basic read/write operations*
- ➔ *Full/empty indication*
- ➔ *Reset behavior*
- ➔ *Metastability handling*
- ➔ *Internal data storage*
- ➔ *Pointer management*
- ➔ *Synchronization logic*

6.1.1 List of functions that will be verified. Description of each function.

As part of our design and verification we will be verifying the following functions:

***FIFO Full** : When a FIFO reaches its maximum capacity, it's considered "full." Further data writes may lead to overflow, risking data loss. Detecting fullness is crucial for preventing overflow and managing data flow effectively, often requiring flow control mechanisms to regulate input.*

***FIFO Empty**: A FIFO is "empty" when it contains no data. Attempting to read from an empty FIFO may result in underflow, causing retrieval errors. Detecting emptiness is vital for preventing underflow and ensuring continuous data flow, often necessitating waiting mechanisms or signaling data producers to prevent processing delays.*

***Write Operation:** The function responsible for writing data into the FIFO buffer. It ensures proper handling of data inputs, including data storage, address pointer management, and synchronization with the write clock domain.*

***Read Operation:** The function responsible for reading data from the FIFO buffer. It manages data retrieval, updates read pointers, and synchronizes data output with the read clock domain.*

***Clock Domain Crossing:** Functions and mechanisms for synchronizing data transfer between asynchronous clock domains. These functions ensure that data is transferred safely and reliably across different clock domains, preventing metastability and timing violations.*

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

Functional Correctness Across All Operating Conditions: *Ensuring functional correctness under all possible operating conditions, including edge cases and corner scenarios, can be challenging and may require exhaustive testing and verification efforts. We try to cover the majority of cases but still few corner cases might be missed in the process.*

Race Conditions: *Identifying and verifying potential race conditions in the design, such as conflicting data accesses or inconsistent pointer updates, can be intricate and may require extensive testing and analysis.*

Timing Closure Across Clock Domains: *Achieving timing closure and ensuring proper setup and hold times across asynchronous clock domains can be challenging and may require iterative timing analysis and optimization*

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

Critical Functions for Tapeout:

- ➔ *Data integrity*
- ➔ *Synchronization*
- ➔ *Metastability handling*
- ➔ *Timing behavior*

Non-Critical Functions:

- ➔ *Functions with Redundant Verification*
- ➔ *Low-Frequency or Rarely Used Features*

7 Tests and Methods

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

Both Black and White Box testing will be used.

7.1.2 State the PROs and CONs for each and why you selected the method for this DUV.

PROs:

- *Leverages knowledge of internal structure for targeted testing*
- *Enables verification of internal states and corner cases*
- *Facilitates debug and coverage analysis*

CONs:

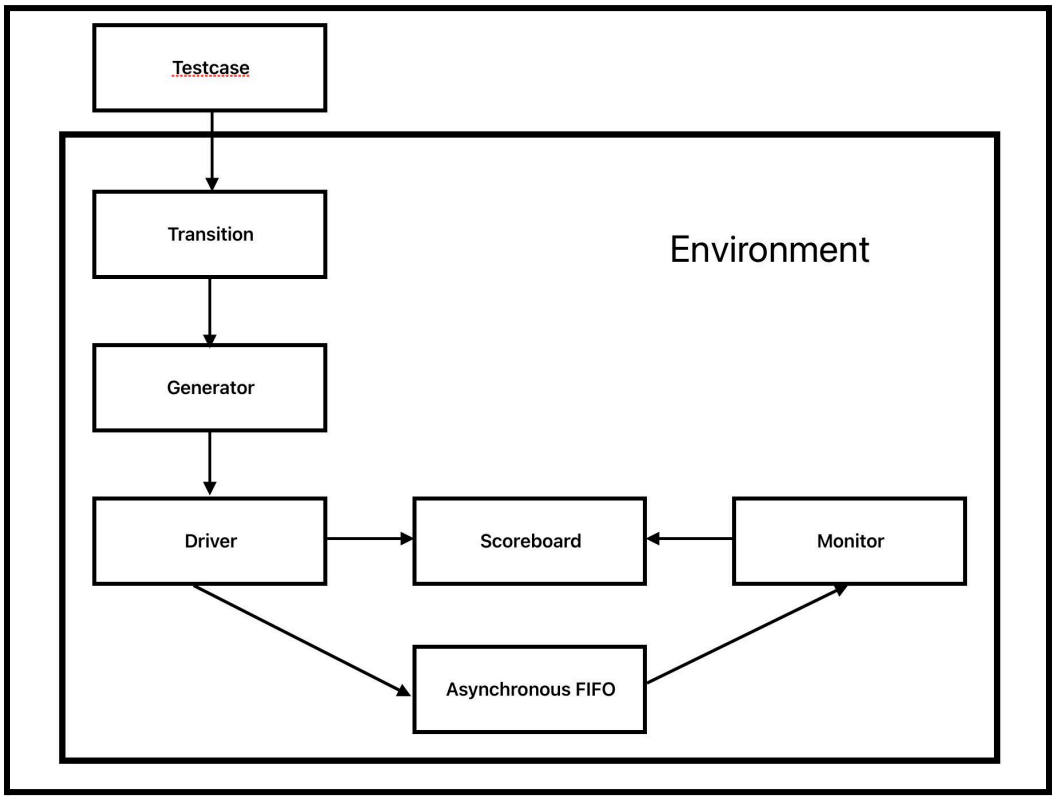
- *Requires some knowledge of implementation*
- *May increase maintenance if design changes*

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

Components Used:

- ➔ *Drivers: Write driver, Read driver.*

➔ *Monitors: Write monitor, Read monitor, Internal signals monitor.*
➔ *Scoreboard: Data comparison and checking.*
➔ *Generator: Generates Stimulus.*



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

Describe why you chose the strategy.

Dynamic simulation will be the primary strategy. It accurately models system behavior, enabling dynamic interactions among components. By mimicking operational conditions, it facilitates extensive testing of functionalities, error handling, and performance metrics. Dynamic simulation provides visibility into design behavior, allowing rapid identification and resolution of issues. Its flexibility accommodates iterative refinement and verification of evolving design iterations.

7.1.5 What is your driving methodology?

7.1.5.1 List the test generation methods (Directed test, constrained random)

Constrained random test generation will be used. This approach automated test case generation, enhancing productivity and enabling rapid verification cycles. By generating diverse scenarios and corner cases, it improves the likelihood of detecting subtle bugs and vulnerabilities.

7.1.6 What will be your checking methodology?

7.1.6.1 From specification, from implementation, from context, from architecture etc

- ➔ *Scoreboard: Data integrity checks*
- ➔ *Assertions: Property checking*
- ➔ *Functional coverage: Verification completeness*

7.1.7 Test Case Scenarios (Matrix)

7.1.7.1 Basic Tests

Test Name / Number	***Test Description/ Features***
1.1.1	*Check basic read operation*
1.1.2	*Check basic write operation*
1.1.3	*Check reset behavior*
1.1.4	*Check full/empty flags*

7.1.7.2 Complex Tests

Test Name / Number	***Test Description/ Features***
1.2.1	*Concurrent read and write operations*
1.2.2	*full FIFO, empty FIFO, almost full/empty*
1.2.3	*Metastability checks*
1.2.4	*Timing verification*

7.1.7.3 Regression Tests (Must pass every time)

Test Name / Number	***Test Description/Features***
1.3.1	*Basic functionality tests*
1.3.2	*Corner case tests*
1.33	*Metastability tests*

7.1.7.4 Any special or corner cases test cases

Test Name / Number	***Test Description***
1.4.1	*Asynchronous clock domain crossings*
1.4.2	*Bug injection and testing scenarios*

8 Coverage Requirements

8.1.1.1 Describe Code and Functional Coverage goals for the DUV

Code Coverage:

- ➔ *Statement coverage*
- ➔ *Branch coverage*
- ➔ *Toggle coverage*

Functional Coverage:

- ➔ *Data patterns*

- ➔ *Corner cases*
- ➔ *Concurrent operations*
- ➔ *Metastability scenarios*

8.1.1.2 Formulate conditions of how you will achieve the goals. Explain the Covergroups and Coverpoints and your selection of bins.

As of now we have planned on covering all the scenarios that fall under our design specification categories that need to be thoroughly verified and implemented. Certain areas include checking for address

8.1.2 Assertions

Assertion based verification has been planned to be implemented to monitor if we are meeting conditions such as reading, writing followed by a certain number of clocks etc… Majorly assertions has been planned to verify,
- Functional Correctness
- Empty and Full Conditions
- Pointer Tracking
- Error Detection

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.

- ➔ *SVA assertions.*
- ➔ *SystemVerilog immediate assertions*
- ➔ *Concurrent Assertions*

By using both immediate and Concurrent assertions there is continuous monitoring for the conditions that we plan on making sure should be met throughout the design and verification phase. Assertions help as built-in checks, validating critical design properties and ensuring compliance with specifications. By defining assertions across various functional paths and corner cases, designers systematically capture and verify expected behaviors, thereby enhancing coverage metrics. Assertions also aid in detecting and localizing bugs early in the development cycle, fostering more robust and reliable designs. Their systematic application not only improves verification efficiency but also instills confidence in design correctness, contributing to higher-quality products and smoother development processes.

9 Resources requirements

9.1 Team members and who is doing what and expertise.

Sujith Krishna *- Logic discussion, Writing design modules, Debugging, Class Based Verification, Code coverage*
Shamshi - *Logic Discussion, DUT verification, Documentation, Class Based Verification,Environment setup*
Prasanna *- Logic Discussion, DUT verification, Documentation, Assertions, Interfaces, Functional coverage*
FYI: Up until now we have worked as a team, everyone involved during logic discussion, design implementation, DUT test bench and documentation(we used google docs to work collaboratively).

10 Schedule

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

Week	Date	Implementation/ Action Plan for the week
1	*19-01-2024*	*Understanding implementation and methodologies implemented in paper1 and paper2(sunburst).*
2	*26-01-2024*	*Design of modules(Logic implementation in design, DUT implementation to verify design).*
3	*02-02-2024*	*Debugging design and making sure basic checks are implemented.*
4	*09-02-2024*	*Assertions, Interface, Class based verification*
5	*16-02-2024*	*Environment creation, scoreboard, checkers, Randomization implementation, UVM based verification planning etc…*
6	*23-02-2024*	*UVM Based verification, Coverage and Functional verification etc…*
7	*01-03-2024*	*UVM Based verification code debugging etc…*

11 References Uses/Citations/Acknowledgements

1. *Title: Simulation and Synthesis Techniques for Asynchronous FIFO Design.*
   Author: Clifford E. Cummings.
   Date: 2002
   Source: Simulation and Synthesis Techniques for Asynchronous FIFO Design

2. *Title: Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons.*
   Author: Clifford E. Cummings, Peter Alfke
   Date: 2002
   Source: Asynchronous FIFO Design with Asynchronous Pointer Comparisons

3. *Title: Crossing clock domains with an Asynchronous FIFO*
   Author: David Harris, Sarah Harris
   Date: 2018
   Source: Crossing clock domains with an Asynchronous FIFO

4. *Title: Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs*
   Author: Clifford E. Cummings
   Date: 2001
   Source: Synthesis Techniques for Designing Multi-Asynchronous Clock Designs