# VERIFICATION TEST PLAN

# (SESSION 1 - TEAM 7)

## Fundamentals of Pre-Silicon Validation Winter - 2024

## Project Name: ASYNCHRONOUS FIFO

## PORTLAND STATE UNIVERSITY


Members:

Sai Teja Gali saitejag@pdx.edu

Abhishek Musku abhimusk@pdx.edu

Sandeep Goud Abbagouni sandeepa@pdx.edu

# 1 Introduction:

## 1.1 Objective of the verification plan

Designing an asynchronous FIFO (First-In-First-Out) queue requires careful consideration of various aspects to ensure robust operation across different clock domains. The verification plan for an asynchronous FIFO is critical to ensure that the design meets its intended specifications and handles corner cases without issues. Here are the key objectives of a verification plan for an asynchronous FIFO:

1. Functional Correctness

- **Data Integrity**: Ensure that the data read from the FIFO is exactly the same as the data written into it, maintaining order.

- **Pointer Integrity**: Verify that read and write pointers are correctly updated and maintained across different clock domains without loss of synchronization.

- **Full and Empty Flags**: Check the accuracy of full and empty flags under various conditions, ensuring they correctly represent the FIFO's state.

2. Clock Domain Crossing (CDC) Verification

- **Metastability**: Ensure that the FIFO design properly handles metastability issues that can arise when signals cross from one clock domain to another.

- **Data Coherency**: Verify that data remains coherent when transferred between the write clock domain and the read clock domain.

- **Stable Convergence**: Test for stable convergence of CDC signals, ensuring that metastable states resolve within a bounded number of clock cycles.

3. Overflow and Underflow Conditions

- **Overflow Handling**: Verify the FIFO's behavior when an attempt is made to write to a full FIFO, ensuring no data corruption or loss.

- **Underflow Handling**: Check the FIFO's response to a read attempt from an empty FIFO, ensuring that the system behaves as expected without crashing or corrupting data.

4. Boundary Conditions

- **Initialization and Reset**: Validate that the FIFO initializes correctly and can be reliably reset under all conditions.

- **Corner Cases**: Test the FIFO at its boundary conditions, such as nearly empty, nearly full, switching from full to not-full, and vice versa.

5. Formal Verification

- **Formal Property Checks**: Use formal verification methods to prove certain critical properties of the FIFO, such as the absence of deadlocks and proper functionality of control logic.

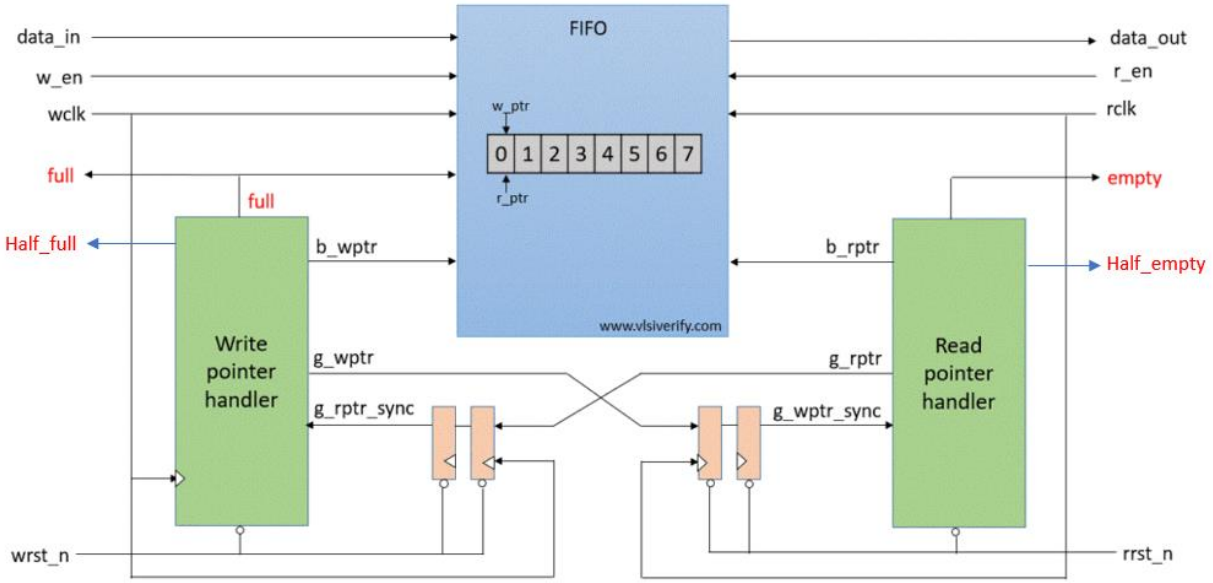
6. Scalability and Parameterization

- **Configurability Tests**: If the FIFO is parameterized (e.g., depth, width), verify its operation over the full range of intended configurations.

7. Conclusion

The verification plan should be detailed and comprehensive, covering all aspects of the FIFO's operation, including functional correctness, clock domain crossings, boundary conditions, and performance metrics. This thorough verification ensures that the asynchronous FIFO will perform reliably in the target application, handling data transfer between different clock domains efficiently and safely.

## 1.2 Top Level block diagram




**Asynchronous FIFO**

## 1.3 Specifications for the design & Design selection:

We choose FIFO1 design over FIFO2. FIFO1 uses a 2-flop synchronizer on both sender and receiver end which helps in avoiding metastability condition in read pointer flipflops and write pointer flipflops. This design is simple and doesn't involve the use of additional comparator component. In FIFO1, we include an additional bit as MSB to both read and write pointer to check the full and empty conditions of a FIFO.

The asynchronous FIFO design is specified to contain 256 entries with a data width of 8 bits each. This FIFO is designed to allow for consistent data transfer between two separate clock domains, ensuring data integrity and synchronization across various clock rates. The FIFO architecture will include robust clock domain crossover (CDC) methods to mitigate metastability and provide uninterrupted data transmission without loss or corruption. With a 130-entry depth, the FIFO provides adequate buffering to withstand bursty data traffic, reducing the risk of overflow under high throughput conditions. The 8-bit width arrangement accommodates a wide range of data types, including simple control signals and more complicated byte-oriented data structures.

Full and empty flags will be provided to allow the surrounding logic to efficiently control the FIFO's status, preventing underflows and overflows. The design will also contain methods for reset & startup to ensure consistent operation under all scenarios. This specification describes a FIFO that is not only optimized for performance and low power consumption, but also designed with scalability and ease of integration in mind, making it appropriate for a wide range of applications that require asynchronous communication between different clock domains.

# 2 Verification Requirements

## Verification Levels

### 2.1.1 **Hierarchy Level**

-The unit level verification is primarily focused on the block or module level, specifically the Asynchronous FIFO. Reason is the Asynchronous FIFO is a complex component with its own set of functionalities, making it a crucial block to verify independently.

### 2.1.2 **Controllability and Observability**

The controllability at the Asynchronous FIFO level is moderate. Controlling inputs, such as write enable, write data, and read enable, can be manipulated to observe different scenarios. Observability is reasonably good. The status outputs, flags, and read data can be easily monitored to assess the internal state of the FIFO during verification.

### 2.1.3 **Interfaces and Specifications**

- **Interfaces**: fifo interfaces

- Includes write enable and write data signals.

- Consists of read enable and read data signals.

- The clock and reset signals for the FIFO.

- Indicators like empty, full, Half_empty, Half_full

-**Specifications**:

- **Write Operation**: Specify the behavior when a write operation is initiated, ensuring data is correctly stored.

- **Read Operation**: Define the behavior when a read operation is initiated, ensuring correct data retrieval.

- **Flag Conditions**: Clearly define the conditions under which status flags (empty, full,halfFull,halfEmpty) are asserted or deasserted.

- **Asynchronous Aspect**: Explicitly specify and verify the asynchronous nature of the FIFO, ensuring proper functioning in asynchronous environments.

# 3 Required Tools

List of required software and hardware toolsets needed

QUESTASIM, UVM packages for UVM based verification

# 4 Risks and Dependencies

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

- Implementation of Half full and half empty flags raised a design bug, which was later removed after verification through testbench.
- Interconnection issues and transfer of data using mailbox, not every generated transaction data was sent from generator to the driver in a sequential order. Had to use events to correct the transmission of data between components of the environment.
- In UVM based verification, couldn't cover all testcase scenarios because of improper sequence generation. Created multiple sequences in the test to cover all the test scenarios.

# 5 Functions to be Verified.

Functions from specification and implementation

**Write Operation**

- Description: The functionality of writing data into the FIFO.

- Verification: Verify that data is correctly stored in the FIFO under various conditions, considering different write enable and write data scenarios.

**Read Operation**

- Description: The functionality of reading data from the FIFO.

- Verification: Ensure that the correct data is retrieved during read operations, considering different read enable conditions.

**Status Flags**

- Description: Empty, Full flags, halfempty, halffull.

- Verification: Confirm that flags are correctly asserted or de-asserted based on the status of the FIFO, ensuring proper flow control.

**Reset**

-Description: Verify the proper functioning of the reset signal. Two reset signals for sender block and receiver block

- Verification: Ensure that resetting the FIFO returns it to a known initial state.

-

# 6 Tests and Methods

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

We used Gray Box, testing strategy known as "gray box" testing blends aspects of white box and black box testing. When conducting gray box testing, the tester is only partially aware of the internal operations of the system being tested.

(DUV) would be based on a number of variables, such as the DUV's nature, testing objectives, and access to internal data. Gray box testing might be a good option if it helps uncover possible problems and if it is in line with the testing objectives and having some understanding of the internal operations is helpful. By integrating the benefits of white box and black box testing approaches, it enables a more thorough testing strategy.

## 6.1 Verification Strategy:

The selected method of verification is dynamic simulation.

Considering:

The benefits of dynamic simulation:

Real-world Simulation: The asynchronous FIFO design can be examined under a variety of operating scenarios thanks to dynamic simulation, a real-world simulation technique. This includes situations where interactions between clock domains and real-time data flow are essential.

Reproduction of Complex Scenarios: Dynamic simulation is a great resource for simulating complex interactions, which are common in asynchronous finite-input feedback orders. It ensures an accurate depiction of operating conditions and allows evaluation of the behavior of the design in response to altering stimuli.

Comprehensive Functional Verification: Dynamic simulation makes it easier to execute comprehensive functional verification by testing a variety of conditions, including those involving different clock frequencies, variable input data rates, and corner cases.

## 6.2 Driving Techniques:

Test Generation Techniques

Directed Test: Focused tests are crucial for focusing on particular features and well-known edge cases in the asynchronous FIFO architecture. In order to guarantee a methodical examination of the design space, these tests can assist in validating particular features, boundary conditions, and essential routes.

Constrained Random: By offering a wider coverage of the design space, constrained random testing is a useful addition to directed testing. This strategy assists in identifying unexpected responses and corner instances that would be overlooked in directed testing alone by producing random stimuli within established restrictions.

## 6.3 According to the specification:

Justification: Verifying that the asynchronous FIFO design complies with the requirements' intended functionality involves comparing it to the specification. It acts as the foundation for functional accuracy.

From Implementation: Justification: To detect inconsistencies between the design and the actual implementation, direct checks against the implementation—such as code-level checks and assertions—are essential. This assists in identifying problems associated with misinterpretations, coding errors, or unexpected departures from the specification.

Context-aware checks take into account the larger system context in which the asynchronous FIFO functions, according to reasoning. This entails examining how one component interacts with others, managing outside stimuli, and making sure all that works well with the system as a whole.

## 6.4 Testcase Scenarios (Matrix)

- Basic Tests

| Test Name / Number | Test Description/ Features |
|---|---|
| 1.1.1 Reset_test | Testing the reset condition when fifo is having some data in it and check the fifo is empty or not. |
| 1.1.2 Write_test | Write operation on fifo and check fifo is full or not |
| 1.1.3 Read_test | Read operation on fifo and check fifo is empty or not |
| 1.1.4 Write_Read_test | First we are writing to the fifo then reading the data that we have written |
| 1.1.5 Write_error_test | Writing to fifo even the fifo is full then full signal will be high in the next cycle |
| 1.1.6 Read_error_test | Reading to fifo even the fifo is empty then empty signal will be high in the next cycle |

- Complex Tests

| Test Name / Number | Test Description/ Features |
|---|---|
| 1.2.1 Concurrent_Write_read_test | Concurrent writes and reads will happen Conditions: fifo_full/fifo_empty/always_full/always empty etc. |
| 1.2.2 | |

- Any special or corner cases testcases

| Test Name / Number | Test Description |
|---|---|
| 1.4.1 Write_error_test | Writing to fifo even the fifo is full then full signal is signal will be high in the next cycle |
| 1.4.2 Read_error_test | Reading to fifo even the fifo is empty then empty signal will be high in the next cycle |

### 6.5 Bug injection Scenario:

Injected a bug scenario, by manipulating the write pointer and writing to same location without incrementing it.

# 7 Coverage Requirements

### 7.1 Code Coverage Goals**:**

A statistic called code coverage is used to evaluate how much of the Design Under Verification (DUV) source code has been run through simulation or testing. It facilitates the identification of code sections that have or have not been used. In order to guarantee that the entire design has been extensively tested, the objective is to obtain complete code coverage.

Statement Coverage:

Objective: Attain complete statement coverage.

Reasoning: To help indicate possible code paths that have not been examined, Reasoning seeks to validate that each single line of code in the DUV has been run at least once during testing.

Branch Coverage:

Reaching 100% branch coverage is the aim.

Reasoning: To give a more thorough understanding of the decision-making process by confirming that both branches of all choices in the code have been taken during testing.

Path Coverage : high path coverage is the aim.

Reasoning: To make sure that during testing, every route through the DUV's control flow has been taken in order to identify any potential problems with the order in which the operations are performed.

Coverage of Function and Procedure:

Objective: Complete coverage of all operations and processes.

Reasoning: To make sure that every process and function—including boundary cases and various usage scenarios—is adequately tested.

**Code coverage of 97.67% was achieved.**

### 7.2 Functional Coverage Goals:

A metric called "functional coverage" is used to assess how thoroughly tests have been conducted in relation to functional requirements or specifications. It assists in guaranteeing that essential features and functionalities have undergone sufficient testing.

Fundamental Features:

Objective: Completely cover all fundamental FIFO operations.

Reasoning: To make sure that all essential asynchronous FIFO functions, such as the ability to write and read data, are thoroughly tested.

Clock Domain Crossings: The objective is to cover clock domain crossings to 100%.

Reasoning: Since the FIFO is asynchronous, it is essential to fully evaluate the interactions between various clock domains to guarantee correct synchronization and data integrity.

Functional coverage was not completely achieved due to the following reasons:

The cross coverage between wr_en and rd_en creates a new possible combination of wr_en and rd_en being high at same time. But we have designed the Transaction in such a way that it only enables read or write once but not both of them simultaneously

**Functional Coverage of 97.72% was achieved**

# 8 Challenges in Design & Verification

Half empty and Half full flags weren't covered in previous design. The design from CummingsSNUG2002SJ covers all operations in FIFO using gray pointers for read and write making it complex for implementing half empty and full flags. We have successfully implemented those flags in the new design.

Another significant challenge we faced in current design was integration and communication between various testbench components such as generator and driver, monitor and scoreboard, and interfacing all those under environment class.

Writing bins for the FIFO involved considering all corner cases for generating inputs, which weren't covered in previous milestone.

# 9 Resources requirements

Team members and who is doing what and expertise.

Sai Teja Gali–

- Design modules: FIFO memory and synchronizers, Design top module.
- Verification Modules: Driver, monitor, scoreboard and Integration of all modules

Abhishek Musku –

- Design modules: Write Pointer, Full logic & Half full logic, Sender and Receiver.
- Verification Modules: Agent, Environment, coverage and Integration of all modules.

Sandeep Goud –

- Design modules: Read Pointer, Empty logic & Half empty
- Verification Modules: Sequence_item, sequence, sequencer and Integration of all modules.

# 10 Schedule

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

| **Milestone** | **WORK** |
|---|---|
| 1 | Worked on complete design and traditional test bench. |
| 2 | Worked on class-based verification test bench including Generator, Driver, Monitor, Scoreboard |
| 3 | Worked on Test Bench Components, Generator, Driver, Monitor, Scoreboard, Coverage. |
| 4 | Worked on Full Functionality of FIFO, Exhaustive Testing and update the SV test bench to “UVM”. |
| 5 | Worked on completing the UVM architecture, UVM environment and UVM testbench<br>Created additional testcase scenarios and bug-injection scenarios.<br>Created a Bug injection scenario and verified it using the UVM testbench. |

# 11 Instructions to run the project

The project can be run by executing the run.do file in questasim.
Go to the location: ece593s24_team_7_Asyn_FIFO_M5-> M5 ->UVM-> run.do
Run the file in questasim by the following command: “do run.do”.

# 12 GitHub Link

https://github.com/saitejagali/team_7_Async_FIFO

# 13 References Uses / Citations/Acknowledgements


- http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf
- http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO2.pdf
- https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdf
- https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/
- https://www.youtube.com/watch?v=0LVHPRmi88c
- https://vlsiverify.com/uvm/
- https://verificationacademy.com/topics/uvm-universal-verification-methodology/