Portland State
UNIVERSITY

IMPLEMENTATION AND VERIFICATION OF ASYNCHRONOUS FIFO USING BOTH CLASS-BASED AND UVM METHODOLOGIES

VERIFICATION PLAN

Fundamentals of Pre-Si Validation

ECE593

Manoj Bontalakoti

Balasainath Reddy Yelampalle

Nikhilesh Devarasetty

Tanvitha Koganti

TABLE OF CONTENTS

1. Objective of Verification Plan

The purpose of this verification plan is to confirm that the asynchronous FIFO (First-In-First-Out) design functions correctly according to its specifications. A robust asynchronous FIFO is crucial for ensuring reliable operation across different clock domains. This verification plan aims to ensure the design meets all requirements and handles corner cases effectively. The primary objectives include:

1.1 Functional Correctness

1. **Data Consistency:** Verify that the data read from the FIFO matches exactly with the data written, maintaining the correct order.
2. **Pointer Accuracy:** Ensure that the read and write pointers are correctly updated and maintained across different clock domains to avoid synchronization issues.
3. **Full and Empty Indicators:** Check that the full and empty flags are correctly set and reflect the actual status of the FIFO in various scenarios.

1.2 Clock Domain Crossing (CDC) Verification

1. **Metastability Handling:** Confirm that the FIFO design effectively manages metastability issues during signal transitions between clock domains.
2. **Data Coherence:** Ensure data integrity during transfer between write and read clock domains.
3. **Stable Convergence:** Test that CDC signals resolve metastable states within a defined number of clock cycles, ensuring stable operation.

1.3 Overflow and Underflow Conditions

1. **Overflow Management:** Validate the FIFO's behavior when writing to a full FIFO, ensuring data is not corrupted or lost.
2. **Underflow Management:** Evaluate the FIFO's response during read attempts from an empty FIFO, ensuring it behaves as expected without crashes or data corruption.

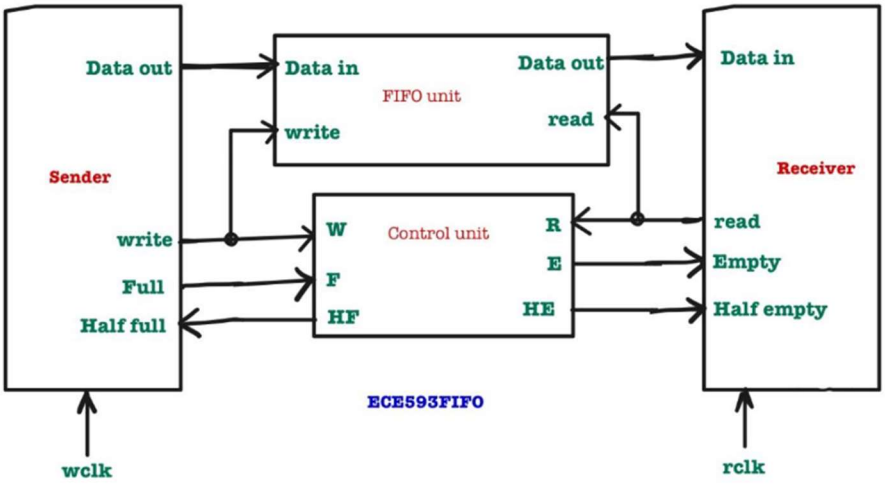
1.4 Boundary Conditions

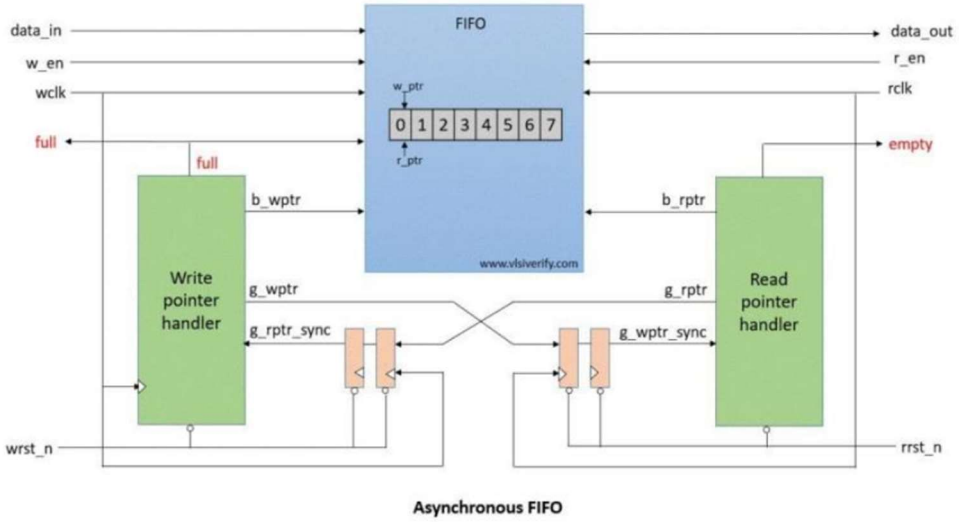
1. **Initialization and Reset:** Verify that the FIFO initializes and resets properly under all conditions.
2. **Corner Case Examination:** Assess the FIFO's performance under boundary conditions, including near-empty, near-full states, and transitions between full and not-full states.

1.5 Scalability and Parameterization

1. **Configurability Testing:** If the FIFO is parameterized (e.g., depth, width), ensure it operates correctly across all intended configurations.

1.6 Top Level Block Diagram





Asynchronous FIFO

2. Specifications for the Design

The asynchronous FIFO design features 350 entries, each capable of holding 64-bit wide data, facilitating seamless data transfer between two distinct clock domains while ensuring data integrity and synchronization despite differing clock rates. Robust clock domain crossing (CDC) techniques are incorporated to address metastability issues, ensuring uninterrupted data transmission. The FIFO's 350-entry depth provides substantial buffering to handle bursts of data traffic, reducing the risk of overflow in high-throughput scenarios. The 64-bit data width supports a wide range of data types, from simple control signals to complex byte-oriented data structures. Integrated full and empty flags manage the FIFO’s status to prevent underflows and overflows, while reset and startup methods ensure consistent operation under various scenarios. The design emphasizes high performance and low power consumption, making it suitable for applications requiring asynchronous communication between different clock domains. Additionally, it is optimized for scalability and flexible integration into various systems, ensuring robust and reliable performance.

3. Verification Requirements

3.1 Verification Levels

3.1.1 Hierarchy Level

Verification at the unit level focuses primarily on the block or module level, particularly targeting the complex Asynchronous FIFO. This approach is essential as the Asynchronous FIFO is a critical component with unique functionalities, requiring thorough and independent verification.

3.1.2 Controllability and Observability

Controllability at the Asynchronous FIFO level is moderate, involving the manipulation of control inputs such as write enable, write data, and read enable to simulate various scenarios. Observability is reasonably good, with effective assessment facilitated by monitoring status outputs, flags, and read data to understand the internal state of the FIFO during verification.

3.1.3 Interfaces and Specifications

Interfaces:

- **Write Interface:** Includes signals for write enable and write data.
- **Read Interface:** Comprises signals for read enable and read data.
- **Clock and Reset:** Involves the clock and reset signals specific to the FIFO.
- **Status Flags:** Indicators such as empty and full.

Specifications:

- **Write Operation:** Clearly defines the expected behavior when a write operation is initiated, ensuring accurate data storage.
- **Read Operation:** Outlines the anticipated behavior when a read operation is initiated, ensuring correct data retrieval.
- **Flag Conditions:** Explicitly defines the conditions under which status flags (empty, full) are asserted or de-asserted.
- **Asynchronous Aspect:** Specifies and verifies the asynchronous nature of the FIFO, ensuring proper functionality in asynchronous environments.

4. Required Tools

4.1 Software

- **Questa-Sim**
- **EDA Playground**

4.2 Hardware

- **Laptop/Desktop:** A computer with a stable internet connection.

4.3 Directory Structure and Resources

- **Software:** We will use Questa Sim and the associated accelerated UVM packages required for FIFO verification.
- **Computer Resources:** Ensure a well-organized directory structure for the runs.

5. Functions to be verified

5.1 Functions from Specification and Implementation

Write Operation

- **Description:** The process of storing data into the FIFO.
- **Verification:** Validate the accurate storage of data within the FIFO under various conditions, accounting for different scenarios involving write enable and write data.

Read Operation

- **Description:** The process of retrieving data from the FIFO.
- **Verification:** Ensure the retrieval of correct data during read operations, considering different read enable conditions.

Status Flags

- **Description:** Indicators showing whether the FIFO is empty or full.
- **Verification:** Verify the correct assertion or de-assertion of flags based on the FIFO's status, ensuring effective flow control.

Reset

- **Description:** Functionality of the reset signal.
- **Verification:** Confirm that resetting the FIFO results in a known initial state, ensuring proper operation.

6. Tests and Methods

6.1 Testing Methods

We employed the Gray Box testing strategy, which is a hybrid approach combining elements of both white box and black box testing. In Gray Box testing, the tester has partial knowledge of the internal operations of the system being tested.

6.2 Pros and Cons of Gray Box Testing and Justification for Selection

Drawbacks:

- **Restricted Internal Knowledge:** Testers may not have full access to the internal architecture or source code, making it challenging to thoroughly understand and test every component.
- **Reliance on Documentation:** Gray box testing depends on accurate and readily available system documentation, which can be problematic if documentation is inadequate.
- **Potential Bias:** Incomplete knowledge can lead to unintentional assumptions or biases in test scenarios, affecting the quality of testing.
- **Complex Test Design:** Creating effective gray box tests can be difficult due to the need to balance understanding internal structures with simulating real-world usage, adding complexity to test design.

Advantages:

- **Combines White and Black Box Testing:** Gray box testing integrates the strengths of both white box and black box testing, addressing both structural and functional issues.
- **Increased Test Coverage:** Testers can create test cases that cover a broader range of conditions compared to black box testing alone, helping to identify issues with both functionality and code execution.
- **Realistic Testing Scenarios:** By simulating actual usage scenarios, gray box testing can reveal flaws not easily visible in black box testing, especially with some understanding of the internal workings.

Justification: The decision to use gray box testing for the Device Under Validation (DUV) depends on factors such as the nature of the DUV, testing objectives, and access to internal data. Gray box testing is advantageous if it helps uncover potential issues aligned with testing objectives, and partial knowledge of internal operations is beneficial. By integrating the benefits of both white box and black box testing, gray box testing enables a more comprehensive and effective testing strategy.

6.3 Testbench Environment

Testbench Architecture: The top-level testbench includes the FIFO Device Under Verification (DUV) and other necessary modules, organizing the simulation process efficiently. Configuration parameters and options for the testbench environment are outlined in the Testbench Configuration section.

Interface DUV: The FIFO Interface, consisting of input and output ports, facilitates communication with the FIFO DUV.

Sequencer for Testing: The Test Sequencer generates read and write operations, edge cases, corner scenarios, and various stimuli and sequences to thoroughly test the FIFO.

Operators:

- **Write Driver:** Manages write operations to the FIFO input ports, generating the necessary data and control signals.
- **Read Driver:** Handles reading data from the FIFO output ports, ensuring accurate data retrieval and processing.

Relationship to the Design: The asynchronous FIFO design and the testbench architecture interact through the generation of test sequences, stimulus driving, input and output monitoring, timing and data integrity checks, and collection of coverage metrics. Each element of the testbench architecture plays a crucial role in ensuring thorough verification of the FIFO design, covering functional accuracy, efficiency, resilience, and compliance with specifications. Comprehensive testing and analysis facilitated by the testbench identify and resolve potential issues and corner cases, ensuring the reliability and efficiency of the asynchronous FIFO across different clock domains.

6.4 Verification Strategies

The chosen verification method is dynamic simulation, which offers several benefits:

- **Real-world Simulation:** Dynamic simulation allows for examining the asynchronous FIFO design under various operating scenarios, including clock domain interactions and real-time data flow.
- **Reproduction of Complex Scenarios:** This method is excellent for simulating complex interactions typical in asynchronous FIFO designs, ensuring accurate representation of operating conditions and evaluating the design's response to changing stimuli.
- **Comprehensive Functional Verification:** Dynamic simulation enables thorough functional verification by testing a range of conditions, including different clock frequencies, variable input data rates, and corner cases.

6.5 Driving Techniques

Test Generation Techniques:

- **Directed Test:** Focused tests are essential for targeting specific features and known edge cases in the asynchronous FIFO design. These tests validate features, boundary conditions, and critical paths to ensure a systematic examination of the design space.
- **Constrained Random:** This technique complements directed testing by providing broader design space coverage. It generates random stimuli within defined constraints, helping to identify unexpected responses and corner cases that might be missed in directed testing.

6.6 Test Case Scenarios

Basic Tests:

- **Reset_test:** Tests the reset condition when the FIFO contains data and checks if the FIFO is empty.
- **Write_test:** Tests the write operation on the FIFO and checks if the FIFO becomes full.
- **Read_test:** Tests the read operation on the FIFO and checks if the FIFO becomes empty.
- **Write_Read_test:** Tests writing to the FIFO followed by reading the data that was written.
- **Write_error_test:** Tests writing to the FIFO when it is full, expecting the write_error signal to be high in the next cycle.
- **Read_error_test:** Tests reading from the FIFO when it is empty, expecting the read_error signal to be high in the next cycle.

Complex Tests:

- **Concurrent_Write_read_test:** Tests concurrent writes and reads under various conditions, such as FIFO being full, empty, always full, or always empty.
- **Test Full:** Writes data to all available locations in the FIFO until it is full, verifying the full flag and the FIFO's behavior when further writes are attempted.
- **Test Empty:** Reads data from the FIFO until it is empty, verifying the empty flag and the FIFO's behavior when further reads are attempted.
- **Test Full Error:** Verifies the FIFO's behavior when attempting to write data while it is already full.
- **Test Empty Error:** Verifies the FIFO's behavior when attempting to read data while it is empty.
- **Test Concurrent Write-Read:** Tests concurrent writing and reading, verifying the FIFO's correct operation under simultaneous read and write conditions.

Functional Verification: Validates basic FIFO functionality under normal operating conditions. Tests various data patterns (e.g., sequential, random) to ensure correct data storage and retrieval. Verifies that the FIFO correctly handles data arrival and departure in the correct order (FIFO property). Checks for proper handling of overflow and underflow conditions. Tests asynchronous operation to ensure proper communication between write and read ports. Verifies that the FIFO operates correctly under different clock domains or asynchronous interfaces.

Regression Testing: Develops a comprehensive suite of regression tests to verify ongoing correctness as the design evolves. Automates test execution and result analysis to streamline the verification process.

7. Coverage Requirements

7.1 Code Coverage

Code coverage is a critical metric that measures how much of the Design Under Verification (DUV) source code has been tested. The goal is to achieve comprehensive code coverage.

- **Statement Coverage:** Aim for complete statement coverage to ensure every line of code in the DUV is executed at least once during testing, identifying untested code paths.
- **Branch Coverage:** Achieve 100% branch coverage to confirm that both branches of all decision points in the code have been tested, providing a thorough understanding of decision-making processes.
- **Path Coverage:** Aim for high path coverage to ensure every route through the DUV's control flow has been traversed during testing, identifying potential issues with the order of operations.
- **Function and Procedure Coverage:** Ensure thorough testing of every process and function, covering boundary cases and various usage scenarios.

7.2 Functional Coverage Objectives

Functional coverage assesses how thoroughly tests align with functional requirements or specifications, ensuring essential features and functionalities undergo sufficient testing.

- **Fundamental Features:** Completely cover all fundamental FIFO operations (enqueue, dequeue, etc.) to thoroughly test essential asynchronous FIFO functions, including writing and reading data.
- **Clock Domain Crossings:** Achieve 100% coverage of clock domain crossings to evaluate interactions between different clock domains, ensuring correct synchronization and data integrity.
- **Managing Complete and Empty Conditions:** Achieve complete coverage for both empty and full FIFO circumstances to ensure proper signal and flag generation, confirming the FIFO operates appropriately when full or empty.
- **Error Circumstances:** Achieve complete coverage of exception handling and error circumstances to ensure proper handling and reporting by the FIFO design for issues like overflows, underflows, or unforeseen circumstances.
- **Performance Metrics:** Comply with performance-related criteria (e.g., latency, throughput) to ensure the FIFO meets performance standards outlined in the design, including data throughput rates and response times.

7.3 Assertions

Effective use of assertions enhances both the functional and coverage aspects of verifying the asynchronous FIFO design.

- **Functional Assertions:** Monitor specific features such as clock domain crossings, enqueue, and dequeue actions in real-time during simulation, ensuring the accuracy of basic operations and synchronization between clock domains. Immediate error detection helps swiftly identify and rectify issues.
- **Data Integrity Assertions:** Verify the correct handling of data during read and write operations to detect potential problems like data corruption or misalignment, ensuring accurate and continuous data transmission within the FIFO.
- **Status Assertions:** Confirm that the design appropriately generates flags when the FIFO is empty or full, enhancing reliability by identifying situations where the FIFO fails to report its status accurately.
- **Clock Domain Crossing Assertions:** Identify issues related to timing violations and metastability in asynchronous FIFO operations, ensuring proper synchronization and data transfer across varying clock frequencies.
- **Coverage-guided Verification:** Strategically place assertions to cover essential paths, trigger corner cases, and address boundary conditions, ensuring a comprehensive exploration of the design space and enhancing overall coverage metrics.
- **Specification-aligned Assertions:** Ensure the asynchronous FIFO complies with mandated specifications by acting as checks against the expected functionality, reinforcing that the design meets functional expectations.

8. Risks and Dependencies

8.1 Metastability and Clock Management

- **Risk:** Challenges related to metastability and the correct use of two clocks.

- **Mitigation Plan:** Implement simulation strategies to cover timing margins and mitigate risks.

9. References

9.1 References / Citations / Acknowledgements

Simulation and Synthesis Techniques for Asynchronous FIFO Design" by Clifford E. Cummings, Sunburst Design, Inc. Sunburst Design Paper