VERIFICATION TEST PLAN

ECE-593: Fundamentals of Pre-Silicon Validation  
Maseeh College of Engineering and Computer Science  
Winter, 2025

A black background with white text

Description automatically generated

**Project Name:** Asynchronous FIFO Design and Verification using UVM  
**Members:** Uma Mahesh Bestha, Thanmai Neelampalli, Sathwik Bandi, Bhanu Sai Sidhardha Tummala  
**Date:** 01/25/2025

Table of Contents

[2 Introduction: 4](#_Toc157438328)

[2.1 Objective of the verification plan 4](#_Toc157438329)

[2.2 Top Level block diagram 4](#_Toc157438330)

[2.3 Specifications for the design 4](#_Toc157438331)

[3 Verification Requirements 4](#_Toc157438332)

[3.1 Verification Levels 4](#_Toc157438333)

[3.1.1 What hierarchy level are you verifying and why? 4](#_Toc157438334)

[3.1.2 How is the controllability and observability at the level you are verifying? 4](#_Toc157438335)

[3.1.3 Are the interfaces and specifications clearly defined at the level you are verifying. List them. 4](#_Toc157438336)

[4 Required Tools 4](#_Toc157438337)

[4.1 List of required software and hardware toolsets needed. 4](#_Toc157438338)

[4.2 Directory structure of your runs, what computer resources you will be using. 4](#_Toc157438339)

[5 Risks and Dependencies 4](#_Toc157438340)

[5.1 List all the critical threats or any known risks. List contingency and mitigation plans. 4](#_Toc157438341)

[6 Functions to be Verified. 4](#_Toc157438342)

[6.1 Functions from specification and implementation 4](#_Toc157438343)

[6.1.1 List of functions that will be verified. Description of each function 4](#_Toc157438344)

[6.1.2 List of functions that will not be verified. Description of each function and why it will not be verified. 4](#_Toc157438345)

[6.1.3 List of critical functions and non-critical functions for tapeout 4](#_Toc157438346)

[7 Tests and Methods 4](#_Toc157438347)

[7.1.1 Testing methods to be used: Black/White/Gray Box. 4](#_Toc157438348)

[7.1.2 State the PROs and CONs for each and why you selected the method for this DUV. 4](#_Toc157438349)

[7.1.3 Testbench Architecture; Component used (list and describe Drivers, Monitors, scoreboards, checkers etc.) 4](#_Toc157438350)

[7.1.4 Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.) Describe why you chose the strategy. 4](#_Toc157438351)

[7.1.5 What is your driving methodology? 4](#_Toc157438352)

[7.1.6 What will be your checking methodology? 4](#_Toc157438353)

[7.1.7 Testcase Scenarios (Matrix) 4](#_Toc157438354)

[8 Coverage Requirements 4](#_Toc157438355)

[8.1.2 Assertions 4](#_Toc157438356)

[9 Resources requirements 4](#_Toc157438357)

[9.1 Team members and who is doing what and expertise 4](#_Toc157438358)

[10 Schedule 4](#_Toc157438359)

[10.1 Create a table with plan of completion. You can use the milestones as a guide to fill this 4](#_Toc157438360)

[11 References Uses / Citations/Acknowledgements 4](#_Toc157438361)

# **Introduction****:**

## **Objective of the verification plan**

The objective of the verification plan for the Asynchronous FIFO design is to ensure that the implemented system meets its specified functional and performance requirements. This involves a comprehensive testing strategy that includes functional, performance, corner case, and stress testing to validate the design under various conditions. The plan aims to identify and rectify any discrepancies or issues in the design before deployment, thereby enhancing the reliability and integrity of data transfer across different clock domains. Ultimately, the verification process is critical for confirming that the asynchronous FIFO operates correctly and efficiently within the larger system architecture, as outlined in the document.

## **Top Level block diagram**

A diagram of a level block diagram

Description automatically generated

## **Specifications for the design**

# **Verification Requirements**

## **Verification Levels**

### **What hierarchy level are you verifying and why?**

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

### **Are the interfaces and specifications clearly defined at the level you are verifying. List them.**

# **Required Tools**

## **List of required software and hardware toolsets needed.**

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

# **Risks and Dependencies**

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

# **Functions to be Verified.**

## **Functions from specification and implementation**

### **List of functions that will be verified. Description of each function**

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

### **List of critical functions and non-critical functions for tapeout**

# **Tests and Methods**

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

**For verifying the Asynchronous FIFO, the following testing methodologies are considered:**

1. **Black Box Testing**
   * **Definition:** Treats the FIFO as a closed system where only inputs and outputs are considered. Internal logic is not examined**.**

**Application to FIFO Verification:**

* + - **Functional verification:** Checking if FIFO behaves as per specifications (e.g., read/write operations, full/empty conditions).
    - **Boundary Conditions:** Checking FIFO operation at near-empty/full states**.**
    - **: Stress Testing:** Applying extreme conditions to observe system response.

1. **White Box Testing**
   * **Definition:** Examines the internal structure of the FIFO design, including RTL (Register Transfer Level) implementation**.**
   * **Application to FIFO Verification:**
     + Verifies Gray Code Counters for pointer updates.
     + Verifying FIFO control logic and state transitions.
     + Checking metastability mitigation in synchronizers.
2. **Gray Box Testing**
   * **Definition:** A mix of Black Box and White Box testing, where the test environment has partial knowledge of internal design.
   * **Application to FIFO Verification:**
     + Checking clock domain crossing (CDC) issues by monitoring synchronization stages.
     + Ensures FIFO pointers are synchronized correctly between read and write clocks.
     + Debugging unexpected behaviors in simulation while using partial design knowledge.

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Method** | **Pros** | **Cons** | **Chosen for FIFO Verification?** |
| Black Box | - Tests functional correctness - Simple to implement - Focuses on I/O behavior | - Cannot detect internal design flaws - Debugging failures is difficult | Yes, for functional validation |
| White Box | - Allows detection of design flaws in RTL - Verifies all internal states - Debugging is easier | - Time-consuming - Requires deep design knowledge | Yes, for control logic, synchronizers, and gray code verification |
| Gray Box | - Balances functional and structural testing - Detects hidden bugs related to CDC and pointer updates | - Requires a mix of testbench and RTL analysis | Yes, to verify CDC, pointer transitions, and deep debugging |

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

**Include general testbench architecture diagram and how it relates to your design**

The **testbench** for the Asynchronous FIFO will follow the **UVM (Universal Verification Methodology)** and include:

1. **Testbench Components:**
2. **Driver:**
   * Generates **random and directed transactions** (write/read operations).
   * Mimics actual FIFO usage scenarios..
3. **Monitor:**
   * Observes transactions on both input and output interfaces.
   * Captures write/read operations and compares against expected behavior.
4. **Scoreboard:**
   * Maintains **expected FIFO state** for comparison.
   * Flags mismatches in expected vs. actual output data.
5. **Checker:**

* Implements **assertions** to verify FIFO properties (e.g., FIFO follows First-In-First-Out behavior).
* Ensures compliance with **full and empty conditions**.

1. **Sequence & Sequencer:**
   * Generates various test cases, including random and directed tests.
   * Handles **corner case testing** (e.g., simultaneous read/write, overflow, underflow).
2. **Environment (env):**
   * Integrates all UVM components into a structured verification setup.
   * Manages multiple test scenarios and test case execution.

**Testbench Architecture**

A screenshot of a computer

Description automatically generated

### **Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.) Describe why you chose the strategy.**

For the **Asynchronous FIFO Verification**, the chosen **verification strategy** is **Dynamic Simulation using SystemVerilog UVM**. This approach ensures that the design functions correctly across various conditions, particularly handling **Clock Domain Crossing (CDC), read/write synchronization, and metastability issues**.

**Selected Verification Strategy: Dynamic Simulation**

**Definition:**  
Dynamic simulation involves applying real-time stimuli to the Design Under Test (**DUT**) and observing its behavior over time using **SystemVerilog and UVM**.

**Why Dynamic Simulation for FIFO?**

* **Ensures Functional Correctness**
  + Verifies FIFO **read/write operations, full/empty conditions, and pointer synchronization**.
* **Detects Timing and Synchronization Issues**
  + Identifies issues caused by **clock domain crossings (CDC)**.
* **Tests Stress and Corner Cases**
  + Simulates extreme FIFO conditions like **continuous write without read, simultaneous read/write operations, and underflow/overflow scenarios**.
* **Supports Random and Directed Testing**
  + Uses **random stimulus generation** to cover unexpected scenarios while allowing specific directed tests for critical cases.
* **Debugging with Waveforms and Logs**
  + Allows **visual debugging using waveform analysis (VCD, FSDB)** and transaction logs

**Key Testing Approaches Under Dynamic Simulation**

1. **Transaction-Level Modeling (TLM) in UVM**
   * Abstracts away low-level details, making verification efficient.
2. **Clock Domain Crossing (CDC) Verification**
   * Ensures that FIFO pointers are synchronized correctly between **write clock and read clock** domains.
3. **Coverage-Driven Verification (CDV)**
   * Uses functional and code coverage to **measure verification completeness**.
4. **Assertion-Based Verification (ABV)**
   * Uses **SystemVerilog Assertions (SVA)** to check FIFO properties (e.g., FIFO is never read when empty).

### **What is your driving methodology?**

**My Driving Methodology for FIFO Verification**

**Constrained Random Stimulus Generation**  
I am using constrained random stimulus to generate **write/read operations, data sequences, and clock variations** dynamically. This helps in covering a wide range of FIFO conditions without manually specifying each scenario.

**Directed Test Cases**  
To ensure coverage of all critical scenarios, I am also writing directed test cases for:

* **FIFO Full and Empty conditions**
* **Simultaneous Read and Write operations**
* **Boundary conditions like FIFO overflow/underflow**
* **Clock domain crossing synchronization**

**UVM Sequences & Sequencer-Based Driving**  
I am implementing UVM sequences that interact with the **UVM driver** to generate different test scenarios efficiently. The **UVM sequencer** controls transaction flow, ensuring dynamic verification.

**Transaction-Level Modeling (TLM)**  
I am using TLM for efficient data handling, reducing low-level signal manipulations, and improving simulation performance. This ensures modular and scalable verification.

#### **List the test generation methods (Directed test, constrained random)**

**Test Generation Methods Used**

**Directed Testing**  
I am creating specific test cases to validate key FIFO functionalities and corner cases, including:

* **Basic Read/Write operations** – Ensuring data integrity.
* **Full & Empty Conditions** – Checking proper handling of FIFO full and empty scenarios.
* **Boundary Conditions** – Testing FIFO behavior when it is **almost full or almost empty**.
* **Clock Domain Crossing (CDC) Scenarios** – Ensuring correct synchronization between write and read domains.

**Constrained Random Testing**  
I am generating randomized stimuli for:

* **Write/Read sequences** with varying data lengths and patterns.
* **Clock frequency variations** between write and read domains.
* **Random bursts of reads and writes** to simulate unpredictable traffic.

**Regression Testing**

* Ensuring that all test cases pass consistently after any design changes.
* Running multiple tests with different seeds to improve coverage.

### **What will be your checking methodology?**

**My Checking Methodology for FIFO Verification**

**Scoreboard-Based Checking**

* I am using a **UVM scoreboard** to compare **expected vs. actual FIFO output**.
* The scoreboard tracks all transactions and flags mismatches when data integrity is violated.
* It helps ensure FIFO follows the **First-In-First-Out** principle.

**Assertions-Based Verification (ABV)**

* I am implementing **SystemVerilog Assertions (SVA)** to verify:
  + **FIFO does not read when empty.**
  + **FIFO does not write when full.**
  + **Correct pointer increments and wraparounds.**
  + **Clock Domain Crossing (CDC) synchronization correctness.**

**Reference Model Comparison**

* I am using a **golden reference model** to validate FIFO behavior against a known correct model.
* The testbench fetches outputs from both the **FIFO DUT and the reference model**, comparing results to detect functional mismatches.

**Functional Coverage Analysis**

* I am tracking **functional coverage metrics** to ensure that:
  + All **FIFO states (full, empty, normal, almost full, almost empty)** are tested.
  + All read/write sequences and burst operations are exercised.

#### **From specification, from implementation, from context, from architecture etc**

**Sources for My Checking Methodology**

**From Specification**

* I am validating FIFO behavior against the **design specification document** to ensure:
  + FIFO adheres to **First-In-First-Out principles**.
  + Proper handling of **full and empty conditions**.
  + Correct implementation of **synchronization across clock domains**.
  + Proper **reset behavior** ensuring FIFO starts in a known state.

**From Implementation (RTL Design)**

* I am directly checking FIFO behavior using the **RTL (Register Transfer Level) implementation** to confirm that:
  + FIFO control signals behave as expected.
  + Read/write operations correctly modify internal registers and pointers.
  + **Gray code pointer synchronization** functions correctly to prevent metastability.

**From Architecture Context**

* I am ensuring that the FIFO design integrates correctly within a larger **system-on-chip (SoC) environment**, checking:
  + Compatibility with **producer and consumer modules**.
  + Data integrity when transferring between **different clock domains**.
  + Timing constraints and interaction with **external memory subsystems**.

**From Verification Environment (Testbench & UVM Components)**

* Using the **UVM scoreboard, reference model, and functional coverage metrics**, I am verifying:
  + Data integrity across FIFO transactions.
  + Detection of incorrect behaviors using **assertions (SystemVerilog SVA)**.
  + Coverage-driven validation ensuring all functional scenarios are exercised.

### **Testcase Scenarios (Matrix)**

#### **Basic Tests**

|  |  |
| --- | --- |
| **Test Name / Number** | **Test Description/ Features** |
| **1.1.1 Basic Write Operation** | **Writes a set of data into FIFO and checks if it is correctly stored.** |
| **1.1.2 Basic Read Operation** | **Reads data from FIFO and verifies correctness.** |

#### **Complex Tests**

|  |  |
| --- | --- |
| **Test Name / Number** | **Test Description/ Features** |
| **1.2.1** | **Concurrent events (R+W)**  **Conditions: fifo\_full/fifo\_empty/always\_full/always empty etc.** |
| **1.2.2** |  |

#### **Regression Tests (Must pass every time)**

|  |  |
| --- | --- |
| **Test Name / Number** | **Test Description/Features** |
| **1.3.1** | **Tests that should always pass** |
| **1.3.2** |  |

#### **Any special or corner cases testcases**

|  |  |
| --- | --- |
| **Test Name / Number** | **Test Description** |
| **1.4.1** | **Special Case testing tests and conditions** |
| **1.4.2** | **Bug injection and testing scenario** |

# **Coverage Requirements**

#### **Describe Code and Functional Coverage goals for the DUV**

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

### **Assertions**

#### **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.**

# **Resources requirements**

## **Team members and who is doing what and expertise.**

# **Schedule**

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

# **References Uses / Citations/Acknowledgements**