# Verification Test Plan

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

## Project Details

- **Project Name:** AXI4-Lite Slave Verification
- **Members:** Sarath Jampani, Venkata Vyshnavi Julakanti, Nithesh Kamireddy, Satyam Sharma
- **Date:** May 2, 2025
- **GitHub Repository:** github.com/nitheshkamireddy7/team_3_AXI4_lite_Verfication

# Contents

# 1 Introduction

## 1.1 Objective of the verification plan

To verify the functionality, correctness, and robustness of the AXI4-Lite slave interface through a class-based UVM-style verification environment.

## 1.2 Top Level block diagram

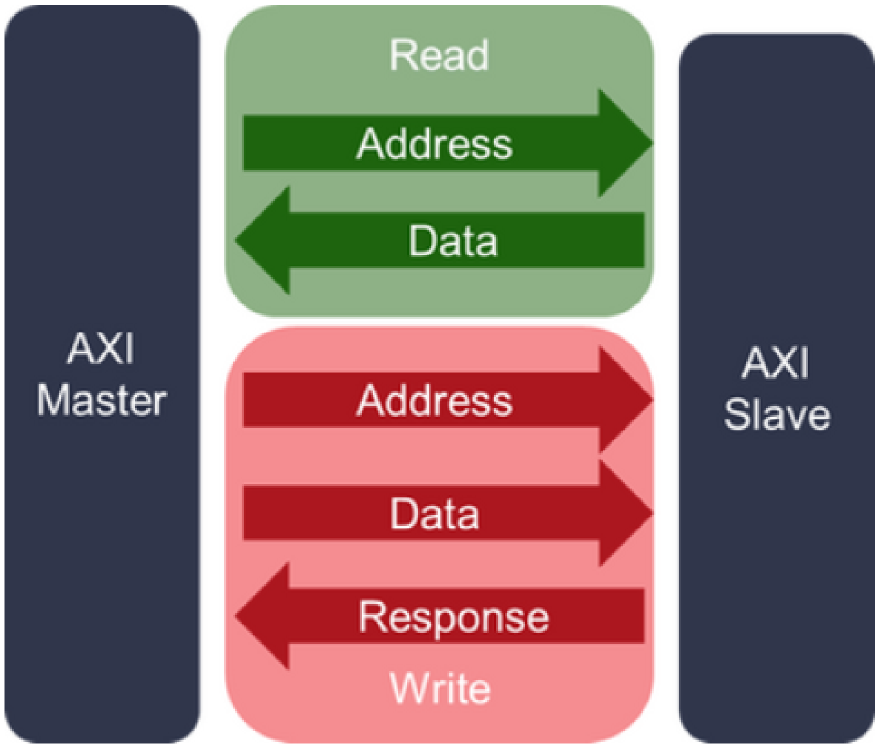

Figure 1: Top-Level Block Diagram of AXI4-Lite Verification

## 1.3 Specifications for the design

AXI4-Lite interface supporting read/write operations with address, data, and strobe signals; including register access and memory-mapped functionality.

# 2 Verification Requirements

## 2.1 Verification Levels

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

Top-level AXI4-Lite slave is verified to capture full system interactions and protocol compliance.

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

Controllability is high via driver; observability is enabled via monitor and scoreboard.

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

Yes. AXI4-Lite channel specifications:

- Write Address Channel (AW)
- Write Data Channel (W)
- Write Response Channel (B)
- Read Address Channel (AR)
- Read Data Channel (R)

# 3 Required Tools

## 3.1 List of required software and hardware toolsets needed

- QuestaSim Simulator
- SystemVerilog/UVM environment
- Linux Workstation

## 3.2 Directory structure of your runs and computer resources

- /rtl
- /tb
- /sim
- /log
- /scripts

## 4 Risks and Dependencies

### 4.1 List critical threats or known risks, and mitigation plans

- RTL updates may break tests *(Mitigation: version control and interface stability)*
- Schedule tight *(Mitigation: parallel development and test reuse)*

## 5 Functions to be Verified

### 5.1 Functions from specification and implementation

#### 5.1.1 List of functions that will be verified

- Read operation
- Write operation
- Concurrent read and write
- Strobe-based memory update
- Memory-mapped register access

#### 5.1.2 Functions that will not be verified (with justification)

- AXI4-Full or burst-based transactions (not in scope of AXI4-Lite)

#### 5.1.3 Critical vs Non-Critical Functions for Tapeout

- Critical: Basic Read/Write, Strobe Functionality, Protocol Timing
- Non-Critical: Corner cases like always full/always empty

## 6 Tests and Methods

### 6.1 Testing Methods: Black/White/Gray Box

White-box and gray-box testing.

### 6.2 Pros and Cons of Each Method and Why Chosen

- White-box: Higher coverage, good debug. **Chosen for internal register access.**
- Gray-box: Useful for integration and functional correctness.

## 6.3 Testbench Architecture

- Components: Drivers (Write: Sarath), Monitors, Scoreboards (Nithesh), Checkers
- Read Agent: Satyam, Interface/RTL: Vyshnavi

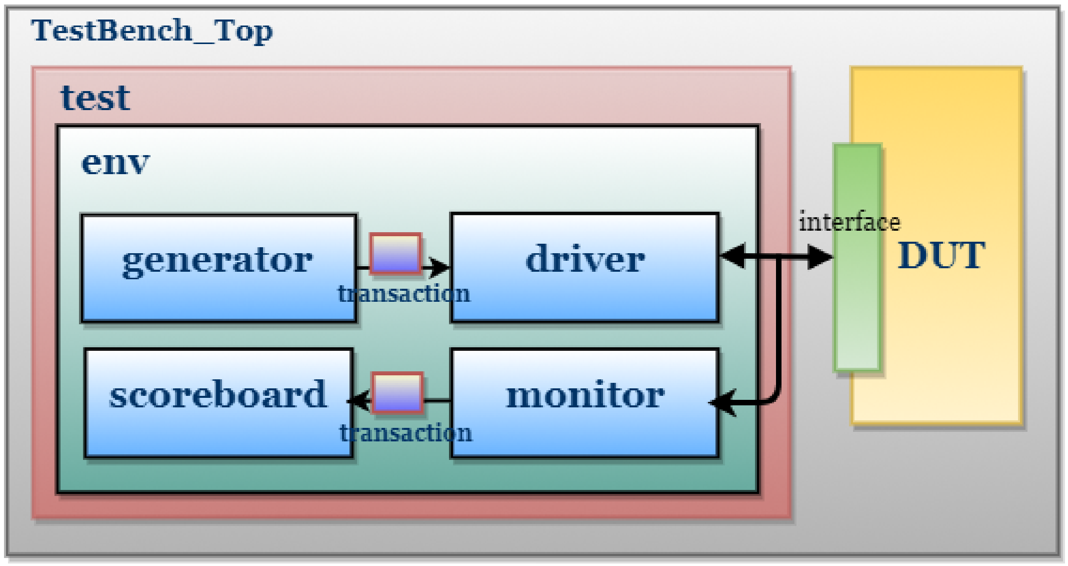

Figure 2: Testbench Architecture for AXI4-Lite Verification

## 6.4 Verification Strategy

Dynamic Simulation using QuestaSim

## 6.5 Driving Methodology

- Directed and Constrained Random test generation

## 6.6 Checking Methodology

Checkers compare DUT outputs against predicted results using scoreboard and functional coverage.

## 6.7 Testcase Scenarios (Matrix)

**Basic Tests**

| Test Number | Test Description / Features |
|---|---|
| 1.1.1 | Check basic read operation |

| | |
|---|---|
| 1.1.2 | Check basic write operation |
| 1.1.3 | Memory access at valid addresses |
| 1.1.4 | Single strobe write functionality |

**Complex Tests**

| Test Number | Test Description / Features |
|---|---|
| 1.2.1 | Concurrent read/write under FIFO full/empty/always full/always empty |
| 1.2.2 | Different write strobes to validate selective byte update in memory |
| 1.2.3 | Overlapping accesses to same memory locations |

**Regression Tests**

| Test Number | Test Description / Features |
|---|---|
| 1.3.1 | Re-run full test suite for regressions |
| 1.3.2 | Random tests with seeds for consistency |

**Corner Case Tests**

| Test Number | Test Description |
|---|---|
| 1.4.1 | Read/write at boundary addresses (e.g., 0x00, 0xFF) |
| 1.4.2 | Bug injection and strobe/memory mismatch scenarios |

## 7 Coverage Requirements

### 7.1 Code and Functional Coverage Goals

- Code Coverage: 95%
- Functional Coverage: 90%

### 7.2 Covergroups and Coverpoints

- Covergroup: Access type (read/write)
- Coverpoints: address range, strobe patterns, read-after-write
- Bins: Write full/half/byte access; Read from empty/full; Invalid address access

### 7.3 Assertions

- Write handshake assertion
- Read valid-ready protocol check
- Strobe update correctness

## 8 Resources Requirements

### 8.1 Team Members, Responsibilities, and Expertise

- **Sarath Jampani** – Write Agent
- **Satyam Sharma** – Read Agent
- **Nithesh Kamireddy** – Environment, Scoreboard, Testbench Integration
- **Venkata Vyshnavi Julakanti** – Interface, RTL, Assertions

## 9 Schedule

### 9.1 Project Completion Plan

| **Milestone** | **Description** | **Target Date** |
|---|---|---|
| Design Read | Understand RTL design | 04/25/2025 |
| Environment Setup | Class-based architecture development | 04/27/2025 |
| Test Development | Regression test creation and execution | 04/29/2025 |
| Coverage Closure | Achieve coverage goals through new tests and analysis | 05/01/2025 |
| Final Report | Document results and finalize report | 05/02/2025 |

## 10 References / Citations / Acknowledgements

- AXI4-Lite Protocol Specification: `https://developer.arm.com/documentation`
- GitHub Project Repo: `https://github.com/nitheshkamireddy7/team_3_AXI4_lite_Verfication`
- Mentor QuestaSim Documentation: `https://eda.sw.siemens.com/en-US/ic/questa/`
- ChatGPT by OpenAI: Used for planning, structuring, and refining verification documentation `https://chat.openai.com`