VERIFICATION TEST PLAN

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

A black background with white text

Description automatically generated

Project Name: Verification of AXI4 Lite Protocol  
Members: Nithesh Kamireddy

Vyshnavi Julakanti

Satyam Sharma

Sarath Chandra Jampani

Date: 5/22/2025

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? 5](#_Toc157438335)

[3.1.3 Are the interfaces and specifications clearly defined at the level you are verifying. List them. 5](#_Toc157438336)

[4 Required Tools 5](#_Toc157438337)

[4.1 List of required software and hardware toolsets needed. 5](#_Toc157438338)

[4.2 Directory structure of your runs, what computer resources you will be using. 5](#_Toc157438339)

[5 Risks and Dependencies 5](#_Toc157438340)

[5.1 List all the critical threats or any known risks. List contingency and mitigation plans. 5](#_Toc157438341)

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

[6.1 Functions from specification and implementation 6](#_Toc157438343)

[6.1.1 List of functions that will be verified. Description of each function 6](#_Toc157438344)

[6.1.2 List of functions that will not be verified. Description of each function and why it will not be verified. 6](#_Toc157438345)

[6.1.3 List of critical functions and non-critical functions for tapeout 6](#_Toc157438346)

[7 Tests and Methods 7](#_Toc157438347)

[7.1.1 Testing methods to be used: Black/White/Gray Box. 7](#_Toc157438348)

[7.1.2 State the PROs and CONs for each and why you selected the method for this DUV. 7](#_Toc157438349)

[7.1.3 Testbench Architecture; Component used (list and describe Drivers, Monitors, scoreboards, checkers etc.) 8](#_Toc157438350)

[7.1.4 Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.) Describe why you chose the strategy. 9](#_Toc157438351)

[7.1.5 What is your driving methodology? 10](#_Toc157438352)

[7.1.6 What will be your checking methodology? 10](#_Toc157438353)

[7.1.7 Testcase Scenarios (Matrix) 11](#_Toc157438354)

[8 Coverage Requirements 12](#_Toc157438355)

[8.1.2 Assertions 13](#_Toc157438356)

[9 Resources requirements 14](#_Toc157438357)

[9.1 Team members and who is doing what and expertise 14](#_Toc157438358)

[10 Schedule 14](#_Toc157438359)

[10.1 Create a table with plan of completion. You can use the milestones as a guide to fill this 14](#_Toc157438360)

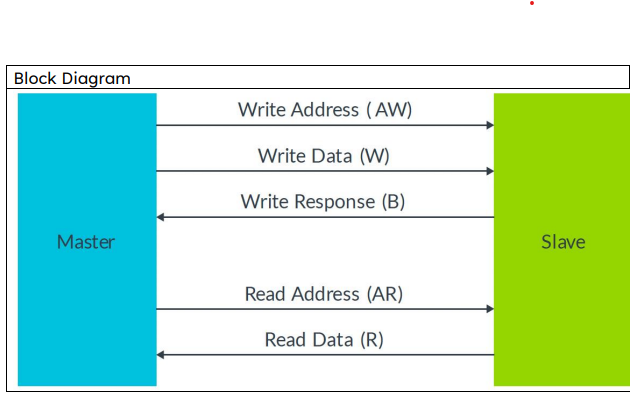
[11 References Uses / Citations/Acknowledgements 15](#_Toc157438361)

# Introduction:

## Objective of the verification plan

To functionally verify an AXI4-Lite slave interface using a class-based SystemVerilog testbench. The focus is on validating protocol handshakes, memory-mapped register access, concurrent channel behavior, and byte-level write correctness via WSTRB.

## Top Level block diagram



## Specifications for the design

* **AXI4-Lite compliant (AW, W, B, AR, R channels)**
* **32-bit data path**
* **4 registers, addressable via 2-bit MSB of address**
* **Byte-wise writes via 4-bit WSTRB**
* **Active-low reset (ARESETn), synchronous ACLK**
* **Memory-mapped access**

# Verification Requirements

## Verification Levels

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

* RTL module level – to validate the complete slave protocol behavior.

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

* Full controllability via constrained-random drivers; observability via monitors and reference memory model.

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

* Clearly defined: AXI4-Lite signals – AWADDR, WDATA, WSTRB, ARADDR, RDATA, and all valid/ready signals.

# Required Tools

## List of required software and hardware toolsets needed.

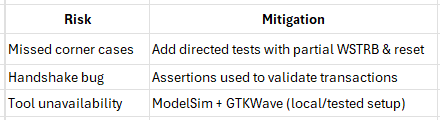
* **Simulator:** QuestaSim
* **Language:** SystemVerilog (OOP-based TB)
* **Viewer:** GTKWave
* **Version Control:** GitHub

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

* /rtl – RTL design
* /tb – testbench code
* /logs – simulation logs
* /scripts – run/compile scripts

# Risks and Dependencies

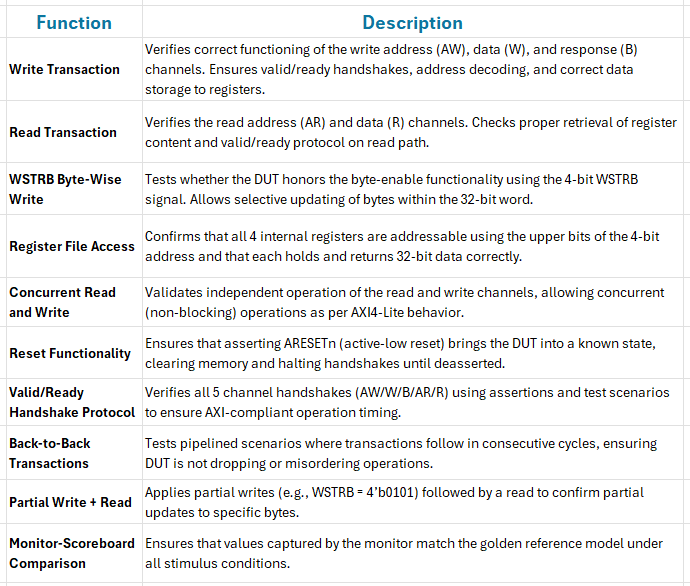
## List all the critical threats or any known risks. List contingency and mitigation plans.



# 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.

* Protocol errors or BRESP codes (out of scope for AXI-Lite)
* Unaligned access (not specified in DUT)

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

* Critical: All five protocol channels, memory behavior
* Non-Critical: BRESP, debug features (not present)

# Tests and Methods

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

* **White-box** (RTL access + internal observation)
* White-box testing was chosen because the DUT is custom RTL (developed by the same team), allowing internal signal observation and control.

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

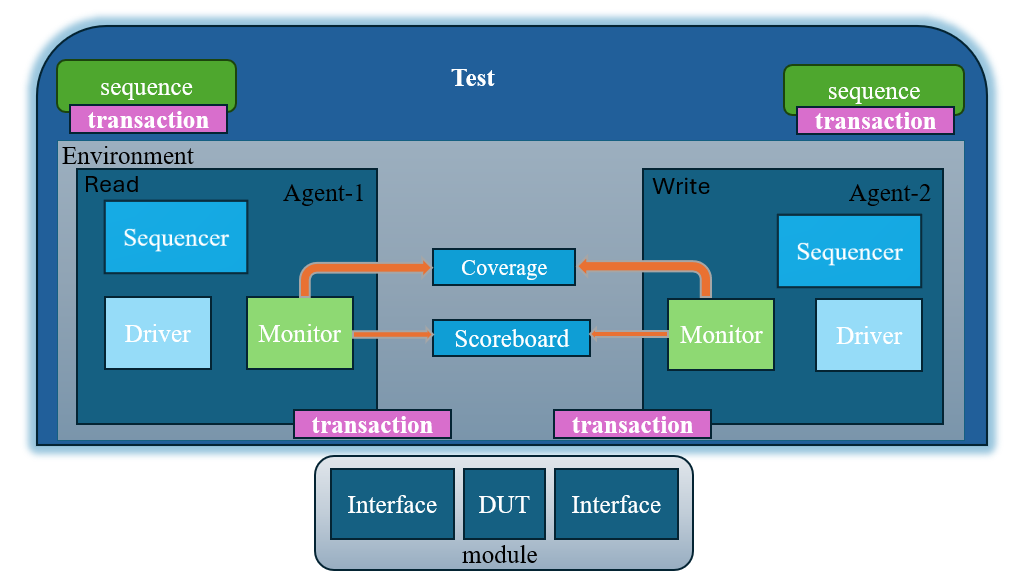
**Pros**

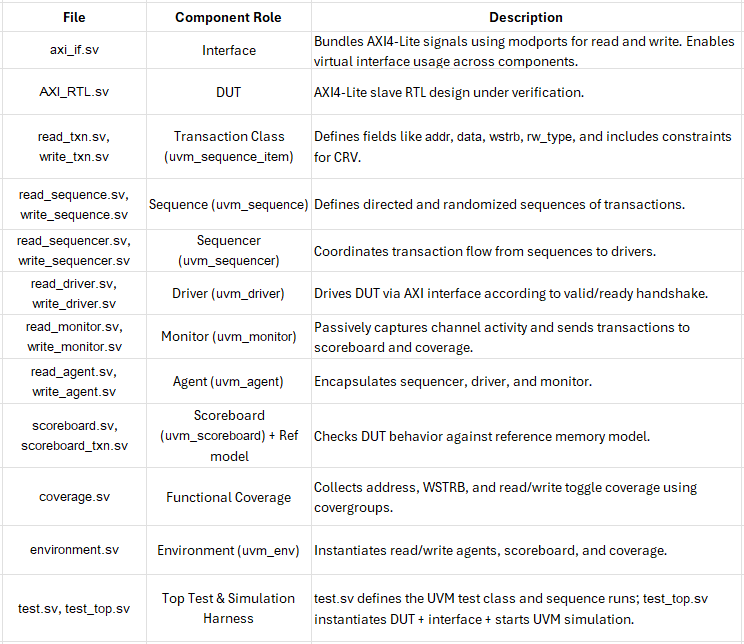
* **Precise debugging:** Internal signals such as register contents and handshake states can be directly monitored, aiding rapid identification of issues.
* **Efficient development:** Design bugs, especially those related to protocol handshakes or register access, can be isolated quickly without reliance on indirect observation.
* **Strong scoreboard alignment:** Full visibility into the design allows tight synchronization between expected behavior (reference model) and observed outputs.

**Cons**

* **Requires detailed RTL understanding:** Verification engineers must be familiar with internal register mapping, reset behavior, and signal timing.
* **High testbench coupling:** Modifications to the design may necessitate updates in drivers, monitors, or scoreboard components due to tight integration.
* **Limited abstraction:** In contrast to reusable UVM Verification IP (VIP), this environment is tailored specifically to the current design, reducing portability and reuse for other DUTs.
* Selected white-box testing because we have a clear understanding of how the AXI4-Lite slave module works internally. This approach lets us directly observe key internal signals like register states and handshake behavior, which makes debugging much easier and faster. It also helped us build a reliable testbench using both directed and constrained-random tests. With most of the checking handled through a functional scoreboard, and a few targeted assertions for protocol handshakes, white-box testing gave us the visibility and control we needed to fully verify the design without adding unnecessary complexity.

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





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

**Dynamic Simulation using UVM**

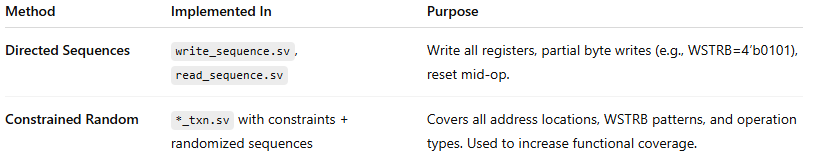
#### **Justification:**

* Enables concurrent testing of AXI read/write channels.
* Supports both directed and randomized stimulus using sequences.
* Allows modular component reuse and structured messaging via uvm\_report\_server.
* Functional coverage, scoreboard checking, and simulation control are tightly integrated into the environment.
* Provides waveform generation and debug visibility through QuestaSim and GTKWave.

### What is your driving methodology?

* All stimulus is generated through UVM sequences and driven to the DUT via sequencers and drivers.
* Read Agent Flow:
* read\_sequence.sv creates read-only transactions.
* read\_sequencer.sv issues them to the read\_driver.sv.
* Read driver asserts ARVALID/ARADDR and waits for handshake.
* Write Agent Flow:
* write\_sequence.sv creates transactions with addr, data, and wstrb.
* write\_driver.sv drives AWVALID/WVALID and waits for response via BVALID.

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



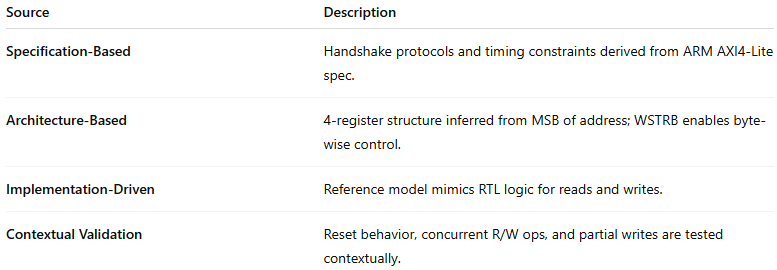
### What will be your checking methodology?

* The DUT output is checked via a scoreboard that compares monitor-observed results to expected values generated by a reference memory model.

**Data Flow:**

* Monitors capture AXI channel activity and forward it to scoreboard.sv.
* scoreboard\_txn.sv helps organize expected vs actual result comparison.
* Ref model updates memory on write and returns expected value on read.

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



### Testcase Scenarios (Matrix)

#### Basic Tests

|  |  |
| --- | --- |
| Test Name / Number | Test Description/ Features |
| 1.1.1 | Perform a basic read from a single register. Validates ARVALID/ARREADY and correct RDATA output. |
| 1.1.2 | Write to a register with WSTRB = 4’b1111 (all bytes valid). Ensures full-word write is stored correctly. |
| 1.1.3 | Apply reset during operation. Check that DUT clears internal state and halts valid transactions. |
| 1.1.4 | Write and read to/from all 4 registers (address range 0x0 to 0xC). Ensures full address decoding is functional. |

#### Complex Tests

|  |  |
| --- | --- |
| Test Name / Number | Test Description/ Features |
| 1.2.1 | Concurrent read and write operations. Verifies independent channel operation and valid/ready handshakes on both sides. Checks for coherency when accessing the same register. |
| 1.2.2 | Perform a partial write using WSTRB = 4’b0101 followed by an immediate read. Verifies byte-level write masking and correct data merging in the target register. |

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

|  |  |
| --- | --- |
| Test Name / Number | Test Description/Features |
| 1.3.1 | Sequence of read/write operations with randomized values. Confirms overall DUT stability and correctness under repeated simulation runs. |
| 1.3.2 | Perform reset between multiple back-to-back transactions. Verifies consistent DUT recovery and output under reset stress. |

#### Any special or corner cases testcases

|  |  |
| --- | --- |
| Test Name / Number | Test Description |
| 1.4.1 | Special case: Write with WSTRB = 4’b0001 (LSB byte only), then read back. Confirms that only byte[7:0] is updated and others remain unchanged. |
| 1.4.2 | Inject backpressure by holding READY low for multiple cycles. Tests DUT’s protocol tolerance, especially during handshake delays. |

# Coverage Requirements

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

The goal is to achieve:

* **100% code coverage** for the DUT (AXI\_RTL.sv) including:
  + All if/else, case, and FSM transitions
  + Conditions around WSTRB, read/write paths, and reset logic
* **95%+ functional coverage** through covergroups focused on:
  + All valid register addresses (4 total)
  + All 4-bit WSTRB byte combinations
  + Operation type: read, write
  + Read-after-write scenarios
  + Reset followed by operation
  + Channel activity coverage (valid/ready assertions per channel)

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

* Functional coverage is implemented using a dedicated coverage.sv file. Covergroups are sampled inside monitors and optionally inside drivers for constrained-random behavior. Coverage reports are extracted post-simulation using tool-generated output.

**Covergroups and Coverpoints:**

|  |  |  |  |
| --- | --- | --- | --- |
| Covergroup Name | Coverpoints | Bins | Description |
| cg\_address | addr | 4 bins for 0x0, 0x4, 0x8, 0xC | Ensures all 4 register addresses are accessed |
| cg\_wstrb | wstrb | 16 bins (all possible 4-bit combinations) | Checks full and partial byte-wise writes |
| cg\_op\_type | rw\_type | 2 bins: read, write | Confirms both operation types are exercised |
| cg\_reset\_sequence | reset followed by op | 1 bin: reset → write/read | Confirms proper DUT behavior after reset |
| cg\_interleaved | concurrent read/write trigger | 1 bin: active overlap | Ensures both agents operate simultaneously at least once |

**How Coverage is Sampled:**

* write\_monitor samples WSTRB and address
* read\_monitor samples address and operation type
* Reset-based covergroup sampled in test sequences

### 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.

* To enhance protocol validation, a **minimal set of AXI4-Lite protocol assertions is planned to be implemented** as part of the verification environment. These assertions will be embedded within the interface (axi\_if.sv) or integrated into the monitor components for runtime protocol compliance checking.

**Purpose of Assertions:**

* **Catch protocol violations early** during simulation
* **Validate handshake behavior** between valid and ready signals
* **Improve observability** without relying solely on waveform inspection
* **Complement functional coverage** by enforcing timing relationships

**Planned Assertions:**

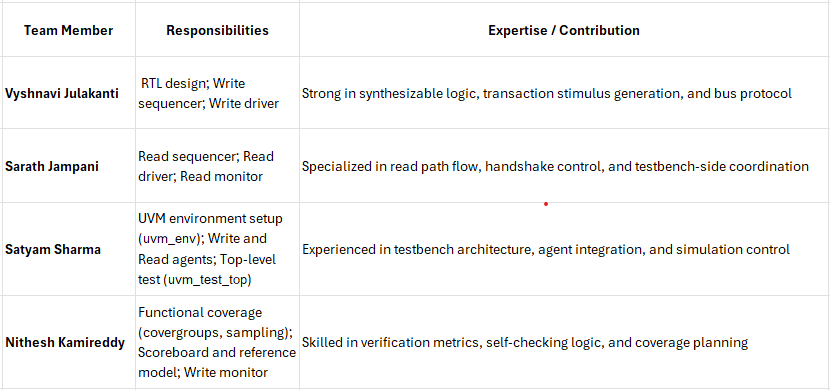
|  |  |  |
| --- | --- | --- |
| **Assertion Name** | **Expression** | **Purpose** |
| assert\_aw\_handshake | `AWVALID | |  | | --- | |  |  |  | | --- | | -> ##1 AWREADY` | |
| assert\_w\_handshake | `WVALID | -> ##1 WREADY` |
| assert\_ar\_handshake | |  | | --- | |  |  |  | | --- | | `ARVALID | | -> ##1 ARREADY` |
| assert\_r\_handshake | |  | | --- | |  |  |  | | --- | | `RVALID | | -> ##1 RREADY` |

**Impact on Verification Quality:**

* These assertions will help enforce **protocol correctness**, especially under random and interleaved traffic. While not directly contributing to coverage metrics, they serve as critical checks to prevent handshake mismatches or timing bugs that might otherwise go undetected.

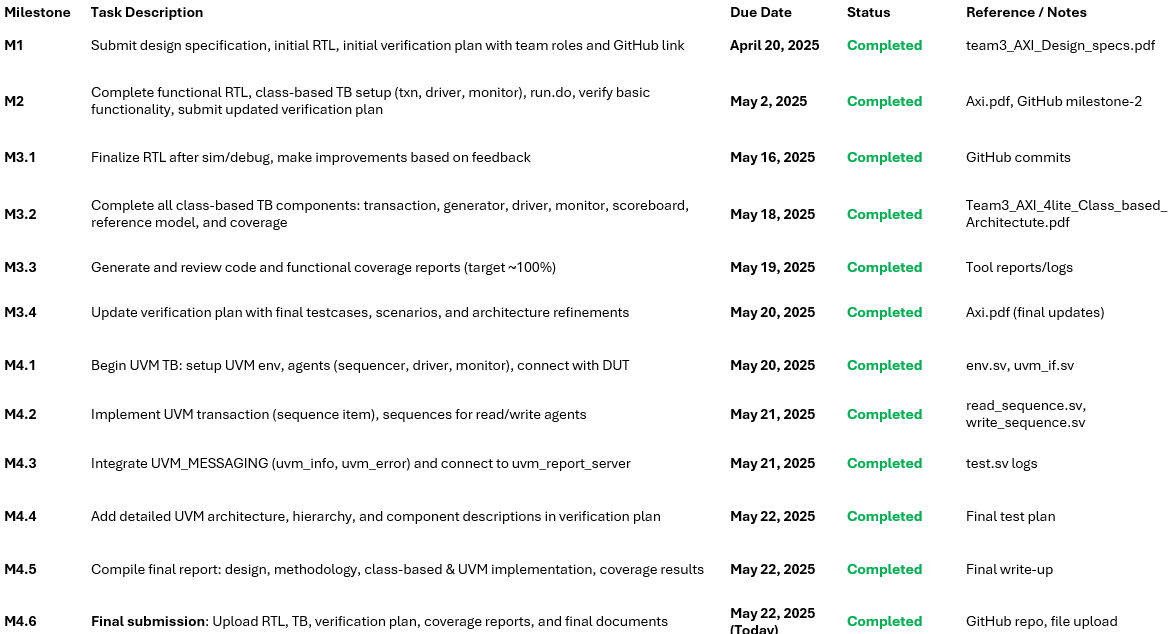
# Resources requirements

## 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

* **ARM AMBA AXI4-Lite Specification**  
  <https://developer.arm.com/documentation/ihi0022/latest/>
* **SystemVerilog IEEE 1800-2017 Standard**
* **UVM Class Reference Manual (UVM 1.2)**
* **ECE-593 Course Materials**  
  Portland State University, Spring 2025 — Instructor: Prof. Venkatesh Patil
* Melikyan, V., Harutyunyan, S., Kirakosyan, A., & Kaplanyan, T. (2021).  
  *UVM Verification IP for AXI.*  
  In **2021 IEEE East-West Design & Test Symposium (EWDTS)**, Batumi, Georgia, pp. 1–4.  
  DOI: 10.1109/EWDTS52692.2021.9580997  
  **Keywords:** AXI, UVM, Verification, Functional Coverage, SystemVerilog, IP Design
* Sangani, H., & Mehta, U. (2022).  
  *UVM-Based Verification of Read and Write Transactions in AXI4-Lite Protocol.*  
  In **2022 IEEE Region 10 Symposium (TENSYMP)**, Mumbai, India, pp. 1–5.  
  DOI: 10.1109/TENSYMP54529.2022.9864552  
  **Keywords:** AXI4-Lite, UVM, Read/Write Protocol, SoC Verification, Simulation, SystemVerilog

**GitHub Repository**:  
<https://github.com/nitheshkamireddy7/team_3_AXI4_lite_Verfication>

**Contributors Acknowledged**:  
All team members equally contributed to implementation, integration, and documentation across various verification phases.