*BY : SARAGADAM DILEEP KUMAR*

*BPRN 16 MAVEN SILICON*

*dileepkumarsaragadam@gmail.com*

AXI 3.0 Single Master-Slave VIP Project Report

## 1. INTRODUCTION

The Advanced eXtensible Interface (AXI) protocol is part of the ARM AMBA family of protocols. AXI 3.0 supports high-performance, high-frequency system designs and is widely used in SoC designs for its flexibility and efficiency. This project aims to create a verification environment using System Verilog and UVM for a simple AXI 3.0-based communication between a single master and a single slave. A Verification IP (VIP) is developed to emulate the AXI protocol and validate data transactions.

SoC verification is inherently complex because it involves integrating multiple Intellectual Properties (IPs). To address this complexity and reduce verification time, reusable Verification IPs (VIPs) are developed. A reusable VIP allows the same verification environment to be applied across multiple SoCs, significantly accelerating the overall verification process.

In this project, we aim to develop a Verification IP for the AMBA AXI3 protocol using the Universal Verification Methodology (UVM). Unlike traditional environments that include RTL designs of the master or slave, this setup uses a back-to-back VIP approach, where one agent in the testbench acts as the AXI master and the other as the AXI slave. No actual RTL module is present; instead, both ends of the communication are modelled using UVM components.

The AXI3 protocol supports five independent channels for read, write, and response operations. It allows multiple outstanding transactions and supports out-of-order responses—features that are rigorously tested in our verification environment. The test plan includes various test case scenarios designed to achieve high functional coverage, ensuring the VIP thoroughly exercises all key features and corner cases of the AXI3 protocol.

## 2. PROJECT OBJECTIVES

1. To implement AXI 3.0 protocol in a test environment with one master and one slave.
2. To develop AXI-compliant master and slave modules.
3. To create a UVM-based testbench for verification.
4. To implement a scoreboard to check data integrity.
5. To capture waveforms for visual verification.

## 3. AXI FEATURES

The key features of the AXI protocol are:

* Separate address/control and data phases
* Support for unaligned data transfers using byte strobes
* Burst-based transactions with only start address issued
* Separate read and write data channels to enable low-cost direct memory access
* Ability to issue multiple outstanding addresses
* Out-of-order transaction completion
* Easy addition of register stages to provide timing closure

**Key points about the AMBA protocol include:**

* Flexibility to suit various SoC architectures with differing size, power, and performance requirements.
* Ensures compatibility across IPs from different providers, facilitating integration and reuse and reducing time to market.
* AXI provides high-performance communication between Master and Slave components.
* Supports multiple master-slave configurations.
* Each channel has specific signals for operations; for instance, the write address channel carries control and write data information.

AXI Protocols drives the following channels:

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

## 4. SIGNALS DESCRIPTION

1. **Write Address Channel (AW)**

Used to send address and control information for write transactions.

| **Signal** | **Width** | **Direction** | **Description** |
| --- | --- | --- | --- |
| AWID | 4bits | Master → Slave | Transaction ID for matching with response. |
| AWADDR | 32 / 64 bits | Master → Slave | Address of the first write transfer in the burst. |
| AWLEN | 4 bits | Master → Slave | Burst length – number of data transfers. |
| AWSIZE | 3 bits | Master → Slave | Size of each transfer (e.g., 2^x bytes). |
| AWBURST | 2 bits | Master → Slave | Burst type: FIXED, INCR, WRAP. |
| AWVALID | 1 bit | Master → Slave | Indicates address and control info is valid. |
| AWREADY | 1 bit | Slave → Master | Indicates slave is ready to accept address. |

**2. Write Data Channel (W)**

Used to send the actual data to be written.

| **Signal** | **Width** | **Direction** | **Description** |
| --- | --- | --- | --- |
| WID | 4bits | Master → Slave | Transaction ID |
| WDATA | 32 / 64 / 128 bits | Master → Slave | Write data. |
| WSTRB | (Width/8) bits | Master → Slave | Write strobe – byte-level control. |
| WLAST | 1 bit | Master → Slave | Indicates the last transfer in the burst. |
| WVALID | 1 bit | Master → Slave | Write data is valid. |
| WREADY | 1 bit | Slave → Master | Slave is ready to accept write data. |

**3. Write Response Channel (B)**

Used by the slave to respond after write operations.

| **Signal** | **Width** | **Direction** | **Description** |
| --- | --- | --- | --- |
| BID | 4 bits | Slave→ Master | Transaction ID (from AWID). |
| BRESP | 2 bits | Slave→ Master | Write response: OKAY |
| BVALID | 1 bit | Slave→ Master | Indicates response is valid. |
| BREADY | 1 bit | Master→ Slave | Master is ready to receive response. |

**4. Read Address Channel (AR)**

Used to send address and control information for read transactions.

| **Signal** | **Width** | **Direction** | **Description** |
| --- | --- | --- | --- |
| ARID | 4 bits | Master → Slave | Transaction ID for matching with read data. |
| ARADDR | 32 / 64 bits | Master → Slave | Address of the first read transfer. |
| ARLEN | 4 bits | Master → Slave | Burst length. |
| ARSIZE | 3 bits | Master → Slave | Size of each transfer. |
| ARBURST | 2 bits | Master → Slave | Burst type. |
| ARVALID | 1 bit | Master → Slave | Read address/control info valid. |
| ARREADY | 1 bit | Slave → Master | Slave is ready to accept address. |

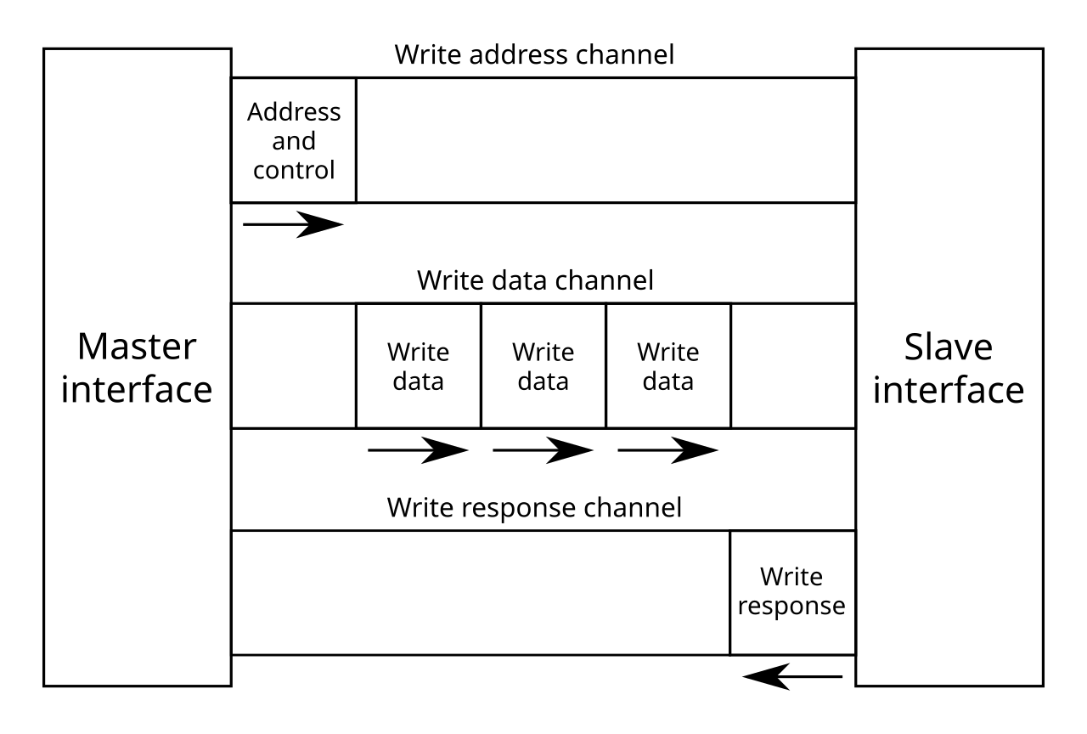
**5. Read Data Channel (R)**

Used by the slave to return read data.

| **Signal** | **Width** | **Direction** | **Description** |
| --- | --- | --- | --- |
| RID | User-defined | Slave → Master | Transaction ID (from ARID). |
| RDATA | 32 / 64 / 128 bits | Slave → Master | Read data. |
| RRESP | 2 bits | Slave → Master | Response type: OKAY, EXOKAY, etc. |
| RLAST | 1 bit | Slave → Master | Last transfer in a read burst. |
| RVALID | 1 bit | Slave → Master | Read data is valid. |
| RREADY | 1 bit | Master → Slave | Master is ready to receive data. |

## 5. AXI ARCHITECTURE

### 5.1 AXI WRITE CHANNEL ARCHITECTURE



**Write Transaction Sequence**

1. **Write Address Phase (AW Channel):**  
   The Master begins a write transaction by transmitting the write address and control signals via the **AW channel**.

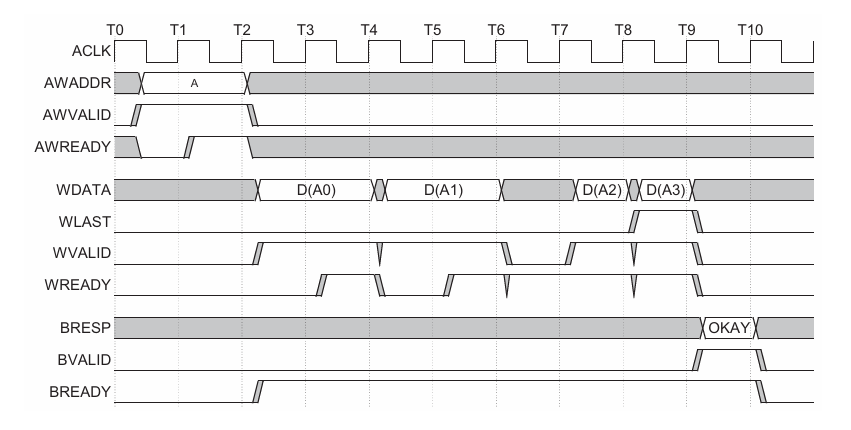
* It asserts AWVALID and drives the address-related signals: AWADDR, AWLEN, AWBURST, and AWSIZE.
* When the Slave asserts AWREADY, the address handshake is completed.

1. **Write Data Phase (W Channel):**  
   After the address phase, the Master sends the actual write data on the **W channel**.

* It asserts WVALID along with WDATA, WSTRB, and WLAST.
* In burst mode, multiple data beats may be transferred.
* The Slave acknowledges each beat by asserting WREADY.

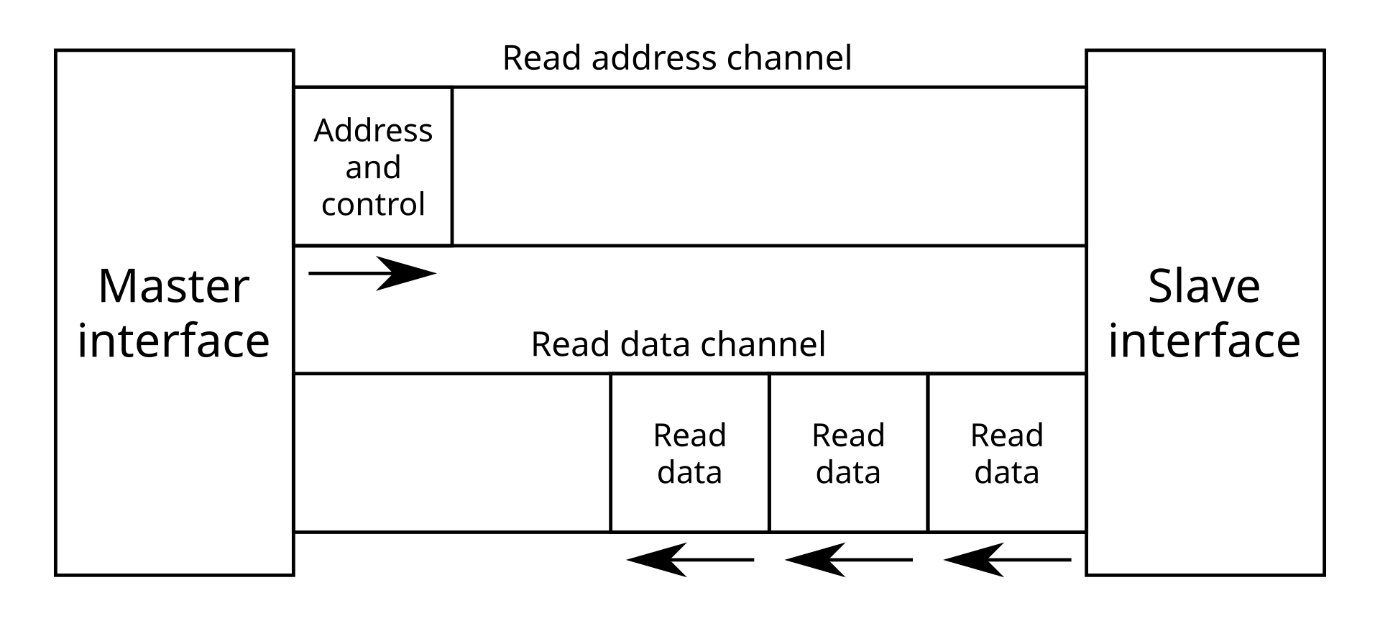
1. **Write Response Phase (B Channel):**  
   Once all data beats are transferred, the Slave responds using the **B channel**.

* It provides the status of the write operation via BRESP and asserts BVALID.
* The Master acknowledges the response by asserting BREADY



The write transaction begins when the Master transmits the address and associated control information over the **Write Address (AW) channel**. Following this, the Master sends the actual data items sequentially over the **Write Data (W) channel**. When the final data item of the burst is transmitted, the Master asserts the WLAST signal to indicate the end of the data phase. Once the Slave has successfully received and accepted all data beats, it generates a **Write Response (B) channel** transaction, driving BVALID along with the response code (BRESP) to inform the Master that the write transaction has been completed.

### 5.2 AXI READ CHANNEL ARCHITECTURE



**Read Transaction Sequence**

1. **Read Address Phase (AR Channel):**  
   The Master initiates a read transaction by transmitting the read address and control information on the **AR channel**.

* It asserts ARVALID and provides signals such as ARADDR and ARID.
* The Slave completes the handshake by asserting ARREADY.

1. **Read Data Phase (R Channel):**  
   The Slave returns the requested data on the **R channel**.

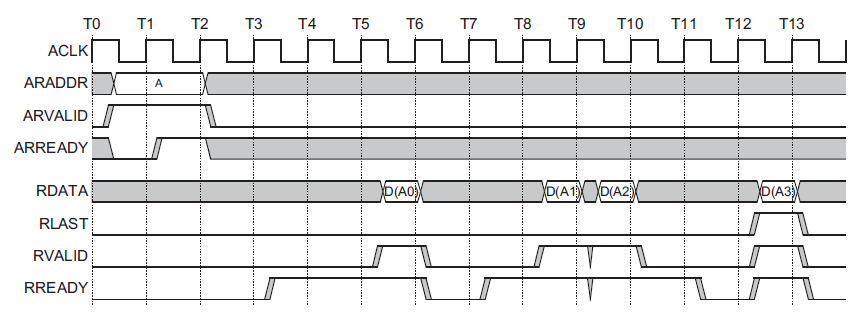
* It asserts RVALID along with RDATA, RRESP, and RLAST (for the final beat in a burst).
* The Master acknowledges each data beat by asserting RREADY.

After the address appears on the address bus, the data transfer occurs on the read data

channel. The slave keeps the VALID signal LOW until the read data is available. For

the final data transfer of the burst, the slave asserts the RLAST signal to show that the

last data item is being transferred.



AXI has five independent channels, each with a set of information signals and VALID and READY signals that provide a two-way handshake mechanism.

The master initiates the VALID signal to indicate when valid address, data, and control information are available in the channel.

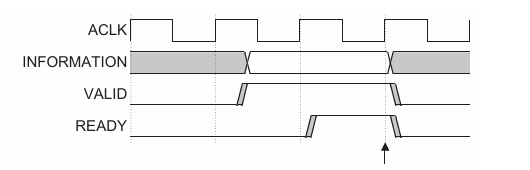
AXI includes a WLAST signal to indicate the last data item in a write transaction, RLAST signal to indicate the final data item in a read transaction.

## 6. DIFFERENT HAND-SHAKING TYPES

All five AXI channels use a common **VALID/READY handshake** for data and control transfers. The **source** asserts VALID when data is available, and the **destination** asserts READY when it can accept the data. A transfer occurs only when both signals are **HIGH**. To maintain signal integrity, no **combinatorial paths** should exist between inputs and outputs on either master or slave interfaces.

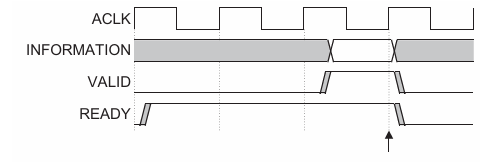
**Three Cases of Hand Shaking Mechanism**

1. Valid Before Ready Handshake

The source presents the data or control information and drives the VALID signal HIGH. The data or control information from the source remains stable until the destination drives the READY signal HIGH, indicating that it accepts the data or control information. The arrow shows when the transfer occurs.

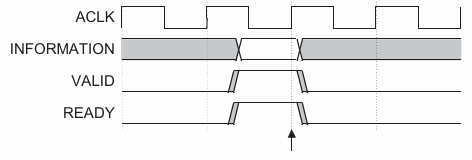
*VALID before READY handshake*

1. Ready Before Valid Handshake

The destination drives READY HIGH before the data or control information is valid. This indicates that the destination can accept the data or control information in a single cycle as soon as it becomes valid. The arrow shows when the transfer occurs.

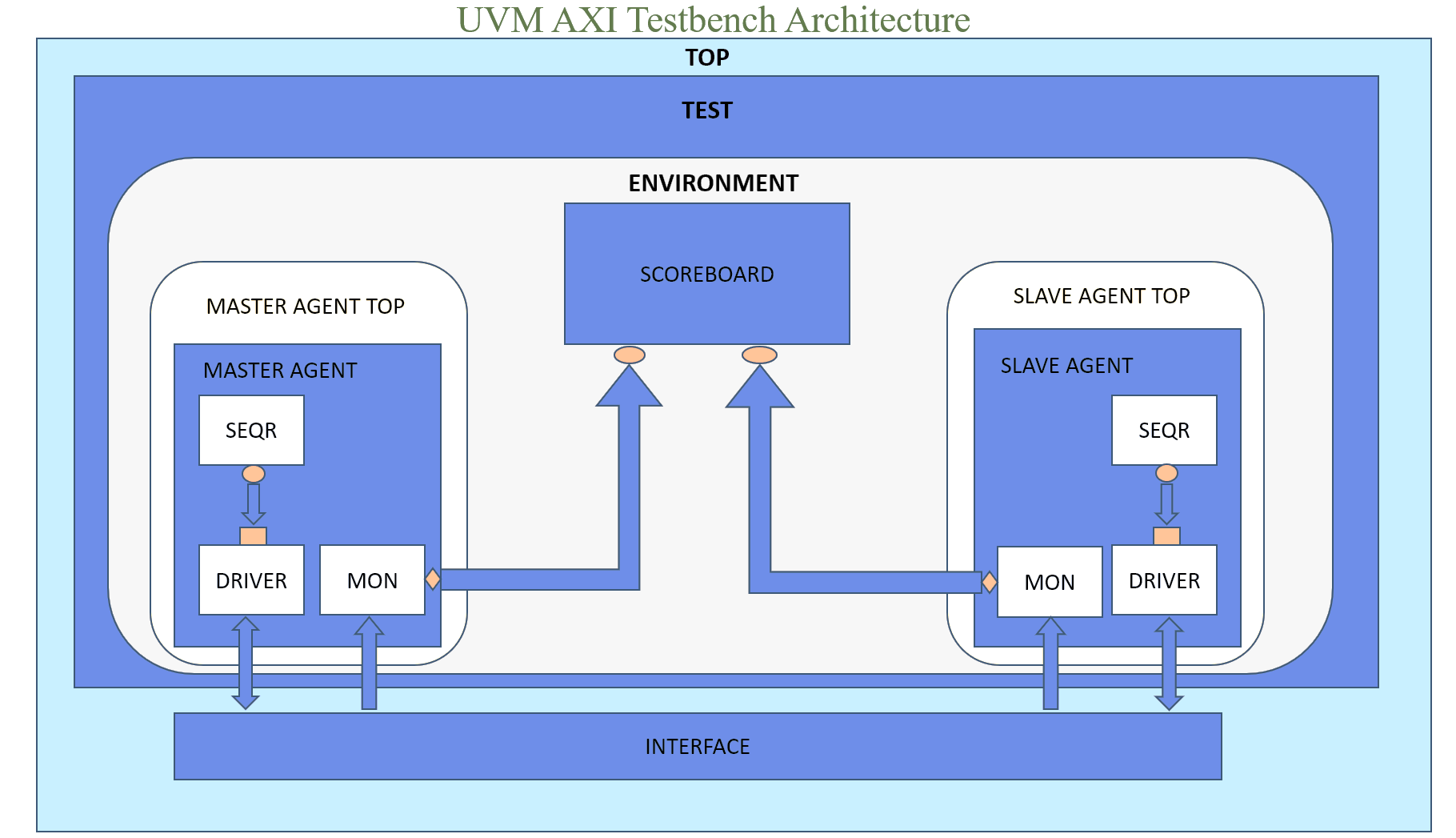
*READY before VALID handshake*

1. Valid With Ready Handshake

Both the source and destination happen to indicate in the same cycle that they can transfer the data or control information. In this case, the transfer occurs immediately. The arrow shows when the transfer occurs.

*VALID with READY handshake*

## 7. AXI VERIFICATION TB ARCHITECTURE



**UVM AXI Testbench Architecture Overview**

The diagram represents a **Universal Verification Methodology (UVM)**-based testbench developed for verifying the AXI3 protocol through **VIP (Verification IP)**. It demonstrates a **Back-to-Back** configuration where both the AXI **Master** and **Slave** are implemented as verification components (no RTL DUT).

**Top-Level Hierarchy**

* **TOP Module**:  
  This is the highest-level wrapper that instantiates the **TEST** class.
* **TEST**:  
  Contains the instantiation and configuration of the **ENVIRONMENT**. The TEST layer defines which sequences to run and initiates the simulation.

**Environment Layer**

The **Environment** encapsulates all the agents and verification components:

* **Master Agent Top**
* **Slave Agent Top**
* **Scoreboard**

**Master Agent Top**

This block contains the **Master Agent**, responsible for driving AXI transactions.

* **Sequencer (SEQR)**:  
  Controls the sequence flow and provides transaction items to the driver.
* **Driver**:  
  Receives transactions from the sequencer and drives them onto the interface as AXI signals (e.g., AWADDR, WDATA).
* **Monitor (MON)**:  
  Observes the interface, captures transaction activity, and converts it into higher-level transaction objects for checking or analysis.
* **Connection to Scoreboard**:  
  The monitor sends observed transactions to the **Scoreboard** for comparison and validation.

**Slave Agent Top**

This is symmetrical to the master agent but acts as a **responder** to transactions.

* **Sequencer (SEQR)**:  
  Generates response transactions (e.g., RDATA, BRESP) for the slave side.
* **Driver**:  
  Drives the slave-side AXI signals in response to master's requests.
* **Monitor (MON)**:  
  Captures incoming data to the slave and sends this to the **Scoreboard**.

**Scoreboard**

* The **Scoreboard** acts as the central checker block.
* It receives transactions from both the **Master Monitor** and **Slave Monitor**.
* It compares **expected vs. actual** results to ensure protocol correctness and data integrity.
* Useful for functional coverage and debugging.

**Interface**

* The **Interface** sits below both the agents and connects them logically.
* It defines all AXI signal declarations and ensures proper synchronization between master and slave drivers/monitors.

## 8. SCORE-BOARD DESCRIPTION

The AXI Scoreboard is a critical component in the functional verification environment, responsible for validating the correctness of AXI protocol transactions between the master and the slave. It monitors the communication happening across the interface and ensures that the data being sent matches the data being received. This process helps identify mismatches or protocol violations early in the verification phase.

**Functionality and Verification Process:**

* The scoreboard receives transactions from both the master and slave agents through monitored data channels.
* It compares the data, addresses, and control signals sent by the master against the responses received from the slave to check for consistency.
* For every transaction, the scoreboard tracks how many reads and writes were received and how many of them matched successfully.
* In case of a mismatch, the scoreboard logs an error for debugging.
* At the end of the simulation, it provides a summary of total transactions received and compared.

**Coverage Process:**

* Coverage groups are used to ensure that all variations and combinations of AXI signals are exercised during simulation.
* Coverage is collected for write and read addresses, burst types, sizes, lengths, data values, and response types.
* Cross-coverage is implemented to verify different combinations of control signals and data strobe patterns.
* For burst transfers, coverage is sampled for each data beat to ensure full observability of signal behavior.
* This detailed coverage helps in identifying untested scenarios and improving test completeness.

## Challenges Faced

* Timing Violations: Faced during simultaneous read/write operations.
* Handshaking Misalignment: Resolved using delays and signal coordination.
* VIP Debugging: Needed multiple iterations to perfect the monitor and scoreboard components.

## Objectives Met

* Developed AXI Master and Slave components as per AXI 3.0 protocol.
* Created UVM environment including driver, monitor, scoreboard, and sequencer.
* Verified read and write operations using multiple test scenarios.
* Debugged timing and handshake issues using waveforms.
* Learned how to implement UVM agents and components.
* Developed debugging skills using waveform analysis.

## Conclusion

This project successfully implemented an AXI 3.0-based communication system with one master and one slave using a UVM-based verification environment. It validated read/write operations, ensured data integrity through the scoreboard, and demonstrated the use of AXI VIP effectively.