International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

# Design and Verification of Single-Master and Single-Slave AXI System using System Verilog

Tiksha Singh
Dept. of Electronics and Communication Engineering
SRM University, Kattankulathur
Chennai, India
ts2110@srmist.edu.in

Praful Babu T
Dept. of Electronics and Communication Engineering SRM University,
Kattankulathur Chennai, India
pt1711@srmist.edu.in

Ashvin Tiwari
Dept. of Electronics and Communication Engineering
SRM University, Kattankulathur
Chennai, India
at6899@srmist.edu.in

AVM Manikandan
Dept. of Electronics and Communication Engineering SRM University,
Kattankulathur Chennai, India
manikanm@srmist.edu.in

Corresponding Authors: AVM Manikandan
manikanm@srmist.edu.in

***Abstract*— The growing sophistication of SoC architectures has generated a solid need of fast and dependable on chip communication and AXI is one of the most utilized bus protocols in contemporary digital frameworks. AXI also supports parallel channel access, efficient burst transfers, independent reads and writes, which gives much greater flexibility and throughput to systems. The paper outlines the development and a test of an AXI slave interface in System Verilog. The slave design proposed has the ability to support all five AXI channels, that is, write address, write data, write response, read address and read data together with Fixed, INCR, and WRAP bursts. This is a 32-bit wide and 128 depth memory model that is integrated to test read and write transactions with various burst configurations. Finite state machines are used to control the architecture with valid and ready handshaking, tracking of burst length, address progression and response generation. To verify, a structured UVM based testbench is created that is comprised of generator, driver, monitor, agent, and scoreboard components to impose constrained random and directed stimulus. Correct burst handling, accurate response signalling, and transfer of accurate data are confirmed by simulation results. The verification logs show that it operates with no errors in various situations which proves the stability and compliance to the protocol of the AXI slave interface.**

**Keywords—AXI, AMBA, SoC, SystemVerilog, UVM, FSM, Burst Transfer, AXI Slave Interface, Functional Verification, Memory Interface.**

## I. Introduction

The high rate of development of SoC architectures has tremendously generated the need of effective and dependable on chip communication [3][8]. The current digital systems combine the processors, memory units and several peripheral blocks into one device, and this necessitates a fast interconnect protocol to handle the communication of data between the master and the slave units. One of the most popular protocols that have been adopted by most systems of this type is AXI as it has the capacity to support high bandwidth communication, independent read and write channels and burst based transactions which enhance overall throughput [3][6]. The proposed system is aimed at the implementation of an AXI slave interface that can support all five channels of AXI, that is, write address, write data, write response, read address, and read data. The protocol enables address, control and data signals to be executed in parallel thus minimizing latency and improving performance. Moreover, AXI also implements Fixed, INCR, and WRAP burst schemes, which allows effective memory access patterns based on the needs of the applications [7]. Nevertheless, proper implementation is not sufficient to ensure sound functioning. In the study, there is a formal verification strategy to make sure that the slave interface is a strict adherence to AXI protocol specifications in various transaction conditions. Effective handling of valid and ready hand shaking, burst length monitoring, address progression and response cueing is vital to avoid functional errors in complex SoC environments [8].

The current project concentrates on the design of AXI slave interface using SystemVerilog and testing its functionality by a UVM based verification environment [10]. The architecture proposed incorporates finite state machines to manage protocol behavior and a special model of memory to verify the operation of read and write across different burst configurations.

## II. Background and Related Work

The more recent developments in System on Chip architectures have greatly added weight to the need to utilize efficient interconnect and communication protocols like AXI, especially in high performance and real time applications.

According to Sharma et al. [1], a hardware optimized Neural Tree classifier was introduced on a Zynq based SoC platform with AXI protocol being closed loop neuromodulation. They had a system with 95.7 percent sensitivity and 94.3 percent specificity using only 0.59 kB on chip memory. It required 174 µW of power in 65 nm CMOS and 0.16 mm 2 of area, illustrating the cost-effectiveness of AXI made SoC integration in biomedical devices.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

In a study by Rudramuniyappa et al. [2], optimized FPGA algorithms to process 5G NR PDCCH were proposed based on AXI stream interfaces. Its implementation demonstrated 1.5 µs latency, 1.2 Gbps throughput, and 10 Gbps/watt power efficiency and hardware utilization of just 1.98 percent. This paper is a demonstration of the significance of high throughput AXI based interconnect in real time communication systems.

Fischer et al. [3] proposed FlooNoC, the AXI4 compliant network on chip that has wide physical connections and multistream transactions. This design had achieved a 645-Gb/s/link bandwidth with 0.15 pJ/byte/hop energy efficiency in 12 nm technology. It was three times more energy efficient and more than twice the bandwidth than state of the art NoCs, and proved the scalability of AXI based interconnects in large scale SoCs.

Heo et al. [4] have discussed FFT accelerator architectures over SoC based platforms with AXI interfaces. Their research indicated that the AXI data width increased by 20-26 percent the processing speed, and tuning of the burst length gave an added 4-5 percent. The systolic array architecture minimized the latency by 15.61 percent over the memory based designs, which demonstrated the effect of AXI configuration on the performance of a system.

Yadav et al. [5] have used AXI interconnects and provided a hardware software co design frame work of medical image processing on an UltraScale MPSoC platform. The design recorded 298.22 frames per second in malaria detection, and 205.87 frames per second in pneumonia detection, and consumed only 14.62 mJ per image of energy. The FPGA implementation was found to be 8.3 times faster and 4.3 times more energy efficient than embedded implementation.

Cuccagna et al. [6] conducted an experimental latency and throughput analysis of AXI DMA based PS to PL communication in edge computing systems. Their study showed that system performance can become either protocol limited or hardware limited depending on configuration. The work provided practical design guidelines for low latency SoC communication.

Jiang et al. [7] proposed AXI InterconnectRT, a real time AXI interconnect architecture that eliminates FIFO based priority inversion. Their compositional scheduling approach improves predictability and scalability in heterogeneous SoCs, addressing timing violations caused by interconnect contention.

Benz et al. [8] introduced AXI REALM, a lightweight real time extension for AXI4 interconnects in mixed criticality systems. The proposed architecture achieved up to 68.2 percent of isolated performance under contention and reduced subordinate access latency by up to 24 times through burst fragmentation and bandwidth regulation, with less than 2 percent area overhead.

Another high performance on chip bus that was proposed by Yang and Andrian [9] is MSBUS and was compared to AXI. Experimental results indicated that MSBUS consumed 63 percent of AXI transfer time consumption, 2.3 times greater wire efficiency and 1.6 times greater valid data bandwidth in block transfer mode and used half the energy. This is a comparison of the performance trade offs in AXI based systems.

Plasencia Balabarca et al. [10] came up with a flexible UVM based verification framework that could be used across AXI, AHB, Avalon and Wishbone interfaces. Their effort showed the need to have scalable and reusable verification methodologies when it comes to multiple bus protocols found in complex SoC settings.

## III. AXI Protocol Overview

AXI is designed in 5 autonomous communication channels that run simultaneously to facilitate effective master and slave communication. This channel separation enables address, data and response information to be simultaneously copied and thus enhances throughput and reduces latency [3][8]. The five channels are, Write Address, Write Data, Write Response, Read Address, and Read Data. Every channel employs valid and prepared handshaking signals to have synchronized communication [8]. The transfer of data only takes place when both the valid and the ready is claimed, which ensures controlled and reliable transactions. In the AXI slave interface, all five channels are implemented in the proposed system. The write related channels handle transfer between the master and slave and the read related channels handle the transfer between the slave and the master.

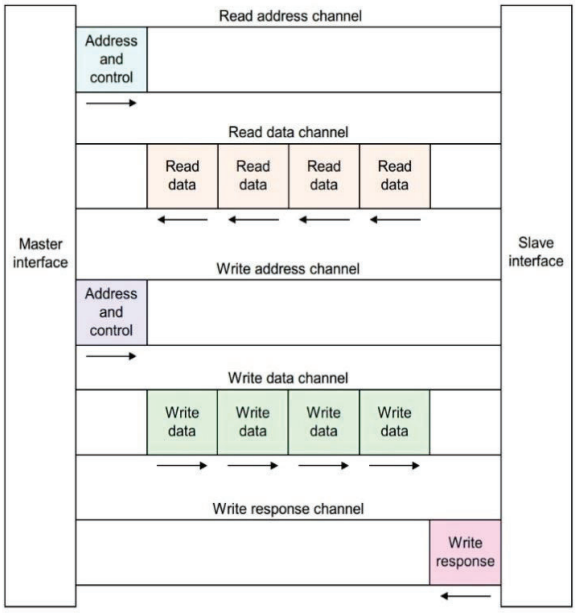


Fig 1: AXI Architechture

### A. *Write Address Channel:*

The Write Address Channel is tasked with sending the starting address and control information between the master and the slave prior to the start of data transfer. This channel determines the place where the data should be written and the particular attributes of the transaction [3]. Notable signals are awvalid, awready, awid, awlen, awsize, awaddr, and awburst.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

The awvalid signal means that the master is supplying a valid address and control information. The slave claims already when it is ready to receive such information. The awid signal is used to uniquely identify the transaction whereas awlen is used to define the burst length. The number of actual data transfers in a burst is awlen + 1. awsize signal is used to set the number of bytes to be transferred and awburst to set the type of burst which can be Fixed, INCR or WRAP. The awaddr signal gives the address at which the write operation will start. The proposed architecture decodes these signals to correctly initiate the write transaction.

### *B. Write Data Channel*

Write Data Channel The data sent by the master to the slave is sent by the Write Data Channel once the address phase has been completed [3]. The key indicators within this channel are wvalid, wready, wid, wdata, wstrb and wlast. The wvalid signal means that there is valid write data on the bus. The slave will claim to be ready when it is ready to take the data. Wid signal is the transaction identifier. The wdata signal is used to transmit the 32 bit data value to be stored in memory. The wstrb signal identifies the lanes of data that have valid data, and allows selective memory updates. The wlast signal represents the last transfer in a burst transaction. These signals are adopted in the proposed system to store data in the right place in memory and to control completion of bursts.

### *C. Write Response Channel*

The Write Response Channel enables the slave to report to the master on the status of the completed write transaction. This guarantees compliance to protocol and reliability [8]. The main indications are bvalid, bready, bid, and bresp. The slave bvalid when there is a response. The master says he is ready to receive the response. Bid signal is used to determine the transaction. The bresp signal gives the status of the transaction like OKAY, SLVERR or DeCerr. The proposed system will produce suitable responses codes in case of successful access of memory or failure conditions like access to an invalid address range.

### *D. Read Address Channel*

The Read Address Channel is used by the master to request data from a specific memory location in the slave. This channel includes signals such as arvalid, arready, arid, araddr, arlen, arsize, and airburst[3]. The arvalid signal indicates that a valid read address and control information are available. The slave asserts arready when it accepts the request. The arid identifies the read transaction. The araddr signal specifies the starting address for the read operation. The arlen and arsize signals define the burst length and transfer size respectively. The arburst signal determines how the address progresses during burst transactions. The proposed architecture captures these parameters to initiate correct read operations.

### *E. Read Data Channel*

The Read Data Channel sends the data requested by the slave to the master. The significant pointers are rvalid, rready, rid, rdata, rresp, and rlast[3][8]. The slave claims rvalid when there is read data. The master declares ready in the condition when it is ready to receive the information. Rid signal is equal to read transaction identifier. The rdata signal contains the 32 bit data value which is read out of memory. The rresp signal is used to show the read operation status and rlast is used to indicate the last transfer in a burst. The suggested system will guarantee that the data is returned properly based on the burst configuration and proper response codes are produced based on valid or illegal access.

The proposed AXI slave interface provides reliable communication, good burst management and compliance with protocol specifications in SoC environments through the implementation of these five channels.

## IV. Burst Transfer Mechanism

One of the most significant features of AXI that enhance efficiency in the access of memories and minimise control overheads is burst transfer. AXI enables the transfer of multiple data beats with a single address phase, as opposed to one data beat being transferred with a different address each time [7]. Awlen and arlen are the number of transfers in a burst, respectively, on write and read operations. Data beats are the real number of beats plus one of burst length. Awsize or arsize defines the size of each data beat (transfer size). The proposed system accommodates all three types of AXI bursts namely Fixed, INCR and WRAP and encodes the logic of address progression as specified in the protocol [3].

### *A. Fixed Burst*

The address in Fixed burst mode is the same one to all the data transfers during the burst [3]. In case the burst length is eight transfers, the address used is the same on all eight data beats. This implies that the data is continually read or written at the same memory address. This type of behavior can be applied in those applications where a particular register or peripheral address has to be updated at a constant time.

A system like this is proposed with the data location in memory decided by the wstrb signal, which selects the valid bit lanes in the wdata, a 32 bit wdata. Because the memory is addressable as 8 bit locations, a lane can be considered a memory location. As an example, when only the least significant byte is valid, all data is written to a single address. When there are many lanes in which data is valid, data is written in sequential addresses starting at the same starting address. Nevertheless, the base address remains the same in the next burst beat in Fixed mode.

### *B. Increment Burst*

The address in INCR burst mode is incremented with each data transfer [3][7]. This mode is usually employed when accessing sequential memory addresses like array or buffers. The increment value is determined by the amount of valid lanes in each transfer. The wstrb signal is used to show the number of valid bytes in the 32-bit data bus and the address is correspondingly incremented.

As an example, when two lanes of the transfer are valid and the next address is updated by adding two to its value. This guarantees that the successive data beats will be stored in successive memory addresses without writing over the old data. The architecture proposed computes the subsequent address following any transfer and updates the memory. This is ensured to ensure that memory space is utilised effectively and that sequential data placement is done correctly in case of burst transactions.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

### C. *Wrap Burst*

Wrap burst mode permits the address to be moved in a sequential manner within a specified region of memory and automatically wraps back to a lower address when the upper limit of that region is met [3]. This mode is especially helpful in applications (like circular buffers or streaming data systems) that need continuous access in a fixed block of memory. The slave in the proposed system measures the burst length and transfer size to decide the address range of the burst. The address grows in a normal manner until it meets the maximum limit of the designated area during the transaction. When this limit is met the address circles back to the initial limit of that region and repeats the burst operation. This guarantees that every data transfer is contained within the assigned memory block, and has sequential access behavior.

## V. Proposed AXI Memory Slave Architecture

The proposed system has a full implementation of AXI slave interface, which is meant to facilitate the read and write based on the specifications of AXI protocol [3][8]. The architecture incorporates a memory block and several finite state machines to deal with channel level control, handshaking, burst control, and response generation. Individual control of each channel in the AXI is done to maintain correct synchronization between the master and slave. The study incorporates structured state based control logic to ensure credibility in data transfer in a variety of transaction conditions.

### A. *Memory Organization*

The architecture proposed has an internal memory block that is 32 bit wide and 128 deep. Memories are addressable at 8 bit level and this provides access to each memory at the level of individual bytes. This configuration enables the selective storage of valid byte lanes, depending on the wstrb signal, during write transactions.

The memory is envisaged as an array structure which can store 32 bit data values and still have access to them at a byte level. In write operations, valid data bytes are written to sequential memory addresses as per the transfer size. In read operations, the stored data previously stored at the given address is read and sent back using the read data channel. This architecture allows effective bursting and malleable access to memory in the slave interface [7].

### B. *Write Address FSM*

Write Address Channel includes a finite state machine which is used to control the acceptance of address and control information by the master [3]. The FSM consists of three main states: idle, start, and ready. The idle state is when the system is waiting to get the reset deactivated and get ready to accept a new transaction. When the reset is released, the FSM switches to the start state which is monitoring the awvalid signal.

In the case of awvalid, the address parameters, burst parameters including awlen, awsize and awburst are recorded. The FSM then shifts to the ready state and awready is asserted to mean that the slave is ready to receive the address. Following successful handshaking between awvalid and awready the FSM goes back to idle state to wait till the next interaction.

This controlled transition ensures synchronized address acceptance and proper initialization of write operations.

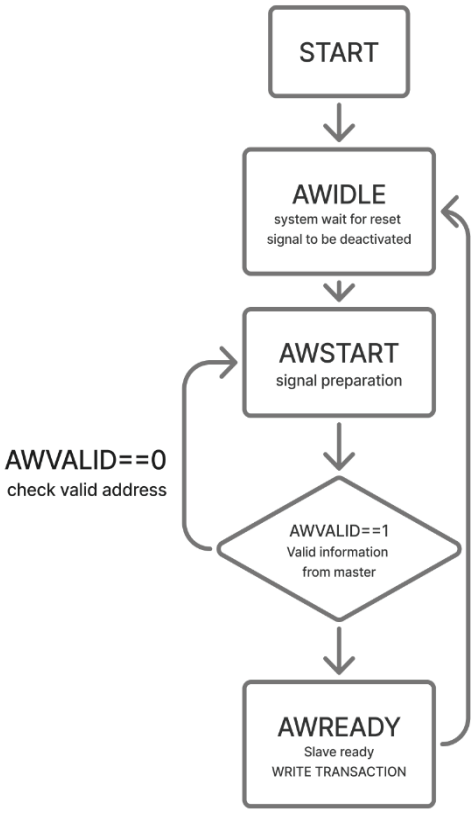


Fig. 3-1. Write Address Channel FSM – The diagram illustrates the state transitions controlling write address acceptance and handshaking between master and slave.

### C. *Write Data FSM*

A write data channel is controlled by a Finite State Machine (FSM) whose responsibility is to receive write data and then store that data in memory [3][7]. There are multiple states in the FSM including the idle, start, address decode, ready, and valid continuation states. During the idle state, all internal counters and control signals are initialized to a known value. Once this is done, the FSM moves into the start state where it waits for an assertion of wvalid (this signals to the FSM that valid data has been received). Once valid data has been detected by the FSM, it will move into the address decode state. Here, the next address (the location in memory where the write data will be stored) will be determined based upon the burst type and transfer size.

After the address has been computed, the FSM will transition into the ready state and assert wready, which signals to the external write data source that the data has been accepted for storage. If the FSM detects that the signal wlast has been asserted (indicating that this is the final data beat of the burst), all internal counters would reset and it would return to the idle state. Should wlast not be asserted, the FSM would continue to process any additional data beats.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

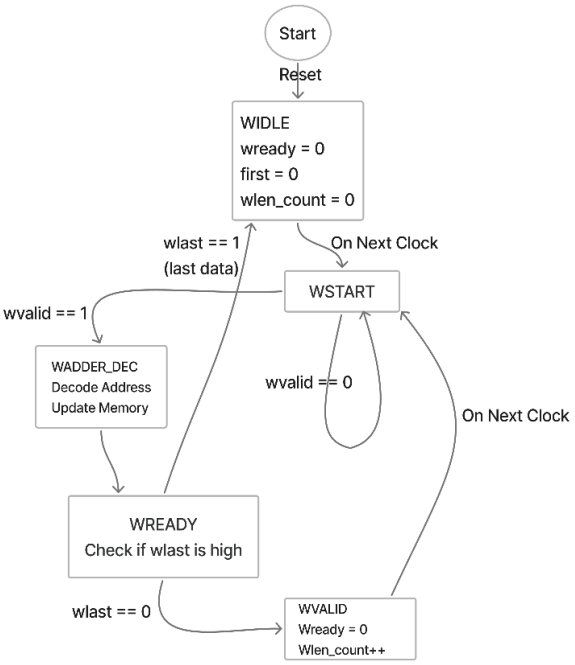


Fig. 3-2. Write Data Channel FSM – The diagram represents the state flow for receiving write data, decoding burst behavior, and updating memory.

*D. Write Response FSM*

The Write Response Channel sends feedback to the master after each 'write burst' is finished. The FSM that controls this channel assures every write transaction gets an appropriate acknowledgement (response) [8]. Once the write data burst has finished, the FSM goes from its idle state to a response preparation state. Here, the bresp signal is generated as the response indicating whether the write operation was completed successfully or not. When the slave asserts bvalid, it indicates that the response has been produced. After the master has asserted bready, it has successfully acknowledged receipt of the response and the FSM will revert back to its idle state. This allows reliable confirmation of successful write operations and full compliance with the protocol.

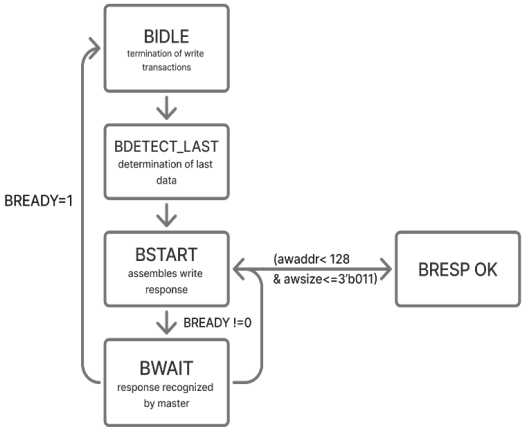


Fig. 3-3. Write Response Channel FSM – The diagram shows the response generation and acknowledgment process following write completion.

*E. Read Address FSM*

There is an FSM that is in charge of regulating the acceptance of read requests via the Read Address Channel [3]. The FSM will contain several states; for example, idle (waiting for a valid read request), start (taking note of the read request), and ready. The slave will be in the idle state until an arvalid signal is issued from the master. At that point, the FSM will store the read address and the associated burst information, and assert the aready signal to acknowledge the acceptance of the read request. Once the handshake is completed, the FSM will begin monitoring for the next read request transaction. The use of an FSM ensures that there is an orderly and synchronous relationship between the master & the slave while establishing a read request.

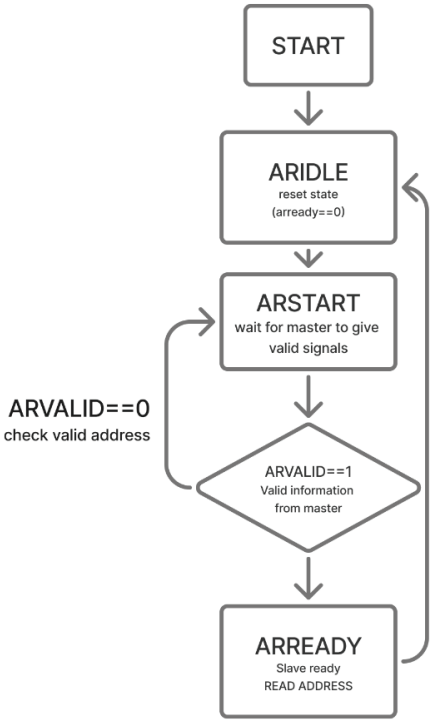


Fig. 3-4. Read Address Channel FSM – The diagram illustrates the state transitions for accepting read addresses and initiating read operations.

*F. Read Data FSM*

The Read Data Channel FSM retrieves the stored data stored in memory, and delivers it back to the master [3][7][8]. It goes through several states during operation including idle, validation, waiting for ready state, data transfer and error handling.

When the Read Data Channel FSM receives an associated read request that is valid, it verifies the address provided and transfer size. Upon verifying that the request is valid, it proceeds to fetch data from memory and places it onto the rdata signal. Then the slave asserts rvalid to indicate that it has valid data available, and will wait for the master to acknowledge receipt by asserting rready.

During burst transactions, the Read Data Channel FSM keeps track of how many data beats have been transferred and

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

asserts rlast upon completion of the last data beat transfer. If it receives an invalid address or experienced an error condition, the Read Data Channel FSM will generate the appropriate rresp code. After completing the burst transfer, the Read Data Channel FSM will return to idle state, waiting for another read request.

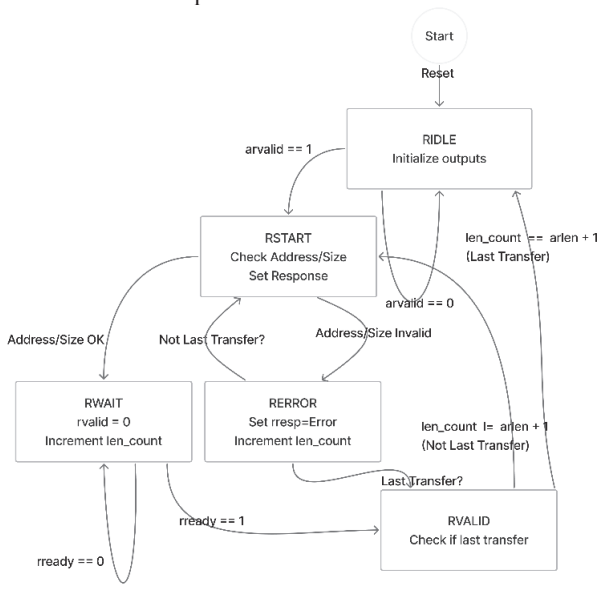


Fig. 3-5. Read Data Channel FSM – The diagram depicts the state transitions governing read data transmission, burst tracking, and response signaling.

## VI. Verification Architecture

The purpose of the verification strategy is to assess the operation of the AXI slave interface in relation to multiple transactions, i.e., single transfers and burst operations. The new proposed design incorporates constrained, random stimulus generation to test combinations of burst length, transfer size and burst type. Additionally, there are a number of directed tests employed within the verification environment used to probe specific edge scenarios - in other words, tests intended to verify a function by distinguishing if the test was successful or not, such as validating boundary conditions and ensuring invalid accesses don't occur.

The verification environment utilizes the wlast and rlast signals of the burst operations occurring within the verification environment to monitor the status of the handshaking mechanisms (i.e., valid and ready), the progression of addresses, the signaling of responses and burst completion.

Functional correctness is determined by using a dedicated mechanism to read the expected values and compare them to the actual output values [10]. The use of a structured verification strategy results in improved coverage and increased confidence in protocol compliance [10].
The verification environment continuously monitors the handshaking sequences of valid and ready handshakes across all five of the AXI channels to ensure proper compliance of the AXI protocol [8]. Each transaction is carefully evaluated by confirming that the handshaking sequence of the transaction between the master and slave is completed prior to any transfer of data. In addition, the sequence of the addresses is evaluated in accordance with burst type selected and validates that the sequence of the addresses are following the corresponding burst configuration during multi-beat transactions.

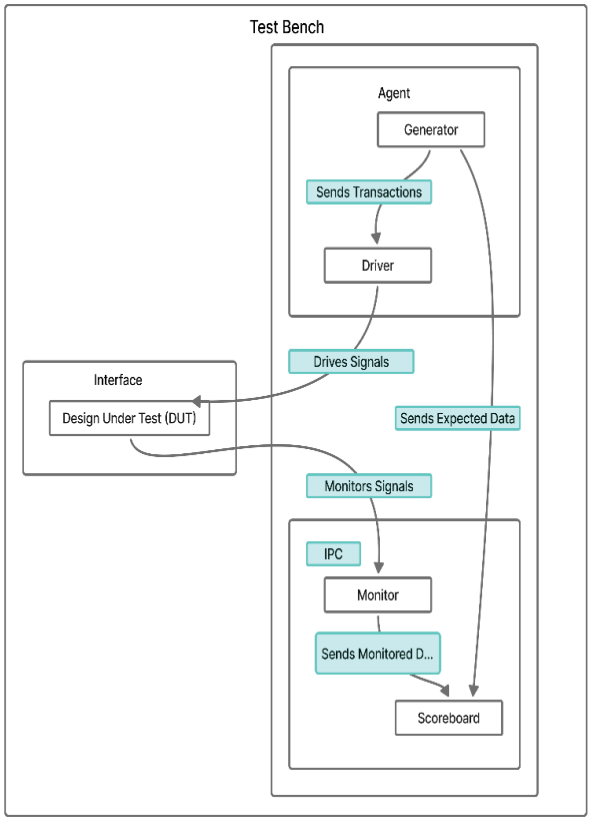


Fig 4-1. Verification Environment

### *A. Testbench Top*

The Testbench top acts as the highest level in the verification framework. This module is where both the AXI slave interface (which represents the actual design being tested) and the verification environment are instantiated. The AXI slave interface and the verification blocks are connected by using a virtual interface and hence are able to communicate with each other without direct physical wire connections. The Testbench top allows for coordinating simulation execution and ensuring verification and DUT (Design Under Test) can properly communicate with each other throughout the execution of the simulation.

### *B. Environment*

The Environment is the top level component of all major verification components; It is an integrated component containing the Agent and Scoreboard and therefore also acts as a controller for managing the verification flow as well [10]. The proposed system has utilized the Environment to provide an interface for sync'ing stimulus generation with the monitoring of DUT activity and the validation of DUT output results. In addition, the environment allows for future AXI

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

based designs since all verification components are able to be arranged according to a single, modular verification hierarchy, allowing for improve scalability and re-use.

### C. Agent

The agent is a reusable verification component that drives and monitors a single AXI interface. It serves as an intermediary between higher-level transaction generation and lower-level signal interaction. It contains subcomponents named the generator, driver and monitor. In this case, the agent is used to apply transactions to the DUT (design under test) and collect corresponding outputs that can be validated with expectations found in the scoreboard. This modular design allows for increased clarity and maintainability of the overall verification framework. [10].

### D. Generator

The generator creates transaction level stimulus for the AXI slave interface. The transaction includes read and write operations with varying burst lengths, transfer sizes and burst types. The research includes generating both constrained random and directed sequences in order to achieve a wide breadth of functional coverage. Once generated, the transactions are sent to the driver for application against the DUT. The expected result of each transaction is then sent to the scoreboard for comparison purposes.

### E. Driver

The Driver converts transaction level activity into the signal level activity on the AXI interface. It propagates signals like awvalid, wvalid, arvalid and control parameters related to protocol timing considerations. The Driver provides correct timing with the system clock, and valid and ready handshaking. The Driver allows pin level interaction with the AXI slave interface by correctly translating transactions into pin level stimulus.

### F. Monitor

The Monitor is a passive element that monitors the output signals of DUT without affecting the system behavior. It records signal activity on channels like write response and read data and transforms into transaction level objects. Observed transactions are sent to the Scoreboard to compare them with what is expected. Monitor is very important when validating the actual behavior of DUT in simulation.

### G. Interface

The Interface is a description of the physical signal interface between the DUT and the verification environment. It bundles signals of AXI channels into a structured bundle, enabling simplified access by components based on classes. The Interface is used by the Driver to drive the input signals and by the Monitor to view the output signals. Such separation enhances understandability and de-couples verification blocks and hardware description code.

### H. Scoreboard

The Scoreboard is the major checking element in the verification environment. The Generator transmits expected transactions to it and the Monitor transmits actual transactions to it. The Scoreboard compares the two streams of data to identify inconsistencies in data, burst processing, or response indication. The suggested system incorporates the use of the Scoreboard to monitor the statistics of transactions, such as successful transfers and the errors identified. Any difference between projected and actual performance is communicated, and any design problems are detected early.

## VII. Simulation Results

The AXI slave interface simulation was done to confirm the functional behavior of the system in various transaction cases. Simulation checks proper handshaking, burst handling, address progression and response signaling in both write and read operations. The outcomes prove that the developed FSMs work as planned and adhere to the requirements of AXI protocols strictly.

### A. Write Channel Result

The write operation starts with assertion of awvalid and the address and burst parameters. When a valid request is detected, the slave claims awready to accept the address phase. The awlen, awsize, awburst parameters are captured parameters which define the burst behavior and the number of transfers.

Under the data stage, wvalid is issued by the master and the slave acknowledges wready to receive every beat of data. The signal wdata conveys the 32 bit data value, and the wstrb signal specifies which lanes of the data to be stored in memory are valid. The address moves in sequence in INCR mode following every accepted beat of the data. The last transfer is signalled by the wlast signal which shows that the burst is complete.

Once the last data beat has been received, the Write Response FSM produces a response with the bresp signal. The slave uses bvalid to show that it has a response ready and the master uses bready to acknowledge it. Value of the response will indicate that the write operation has been successfully completed.

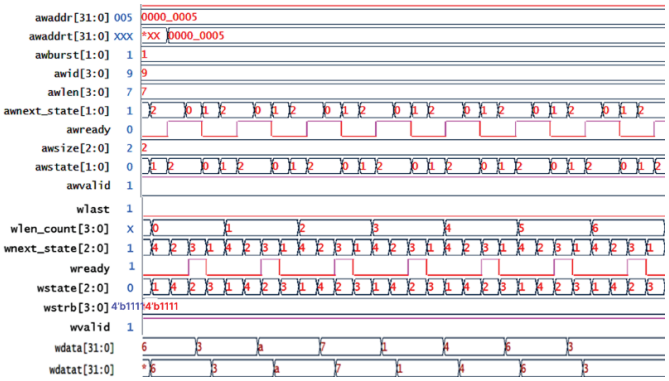


Fig. 5-1. Write Address and Write Data Channel Results - The waveform shows proper acceptance of address, burst development, as well as data transfer via valid and ready handshaking.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026


Fig. 5-2. Write Response Channel Results - The waveform indicates that response is generated successfully and this response is acknowledged upon completion of the write burst.

The state changes of the Write Address, Write Data and Write Response FSMs occur in the correct order without any protocol violations showing that the write channel is implemented correctly.

### *B. Read Channel Result*

Read operation starts when the master sends arvalid, starting address and burst parameters. The slave replies by claiming aready, which means that the read request was successfully accepted. The burst length and burst type determine the number of data beats returned and the direction the address follows.

In the data stage, the slave reads stored values in memory and puts them on the rdata signal. The slave asserts to rvalid to show that valid data is present. The master claims to be ready to take every beat of data. In the burst mode, address increments sequentially in INCR mode, and the len_count mechanism makes sure that the number of transfers is duly tracked. The last beat of the data is asserted as the rlast signal to indicate burst completion.

The rresp signal is in the OKAY state when there is a valid access. The Read Data FSM generates the corresponding response code in case of address error or invalid address.

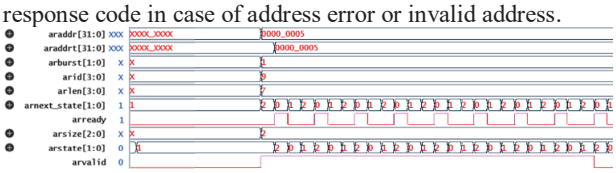


Fig. 5-3. Read Address Channel Results - The waveform shows that there is correct read request acceptance and burst parameter decoding.

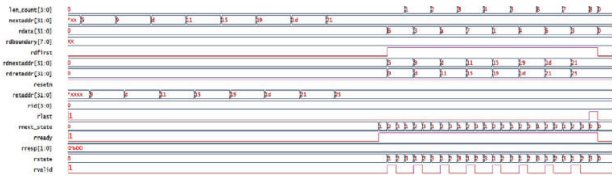

Fig. 5-4. Read Data Channel Results The waveform indicates a series of data transfer, proper burst following, and appropriate rlast assertio.

The simulation confirms correct synchronization between master and slave during read operations and accurate memory retrieval behavior.

### *C. UVM Verification Result*

The UVM based verification environment generates simulation logs that show transaction level activity as it runs. Informational messages verify the start of write and read operations with given burst length and transfer size parameters.

According to the logs, write response and read response values are error-free between the simulation. The Scoreboard shows that there is no mismatch between the expected and observed transactions and that the proposed AXI slave interface is functionally correct. Constrained random stimulus enables the exploration of various burst configurations and address combinations and enhances the verification confidence.

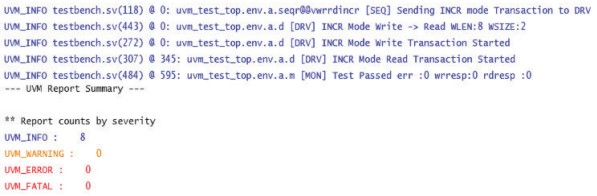


Fig. 5-5. UVM verification results

Functional coverage analysis indicates that different burst types, transfer sizes, and response conditions were exercised during simulation. The absence of reported protocol violations and the successful completion of transaction

## VIII. Performance Analysis

The proposed system strictly follows AXI valid and ready handshaking rules across all five channels. The UVM based verification environment was used to validate multiple burst configurations and transaction scenarios. The following verification metrics were achieved during simulation:

Table 1: Verification Metrics during Simulation

| Parameter | Results |
|---|---|
| Write transaction success rate | 100% |
| Read transaction success rate | 100% |
| Burst type coverage | Fixed, INCR, WRAP fully exercised |
| Handshake compliance | 100% |

To better understand the advantages of the proposed system, a comparison is made between a basic non burst memory interface and the implemented AXI slave interface. The comparison highlights improvements in address overhead, channel independence, burst capability, and verification robustness. Unlike a simple memory interface where each transfer requires a separate address phase, the proposed AXI architecture reduces control overhead through burst based communication and parallel channel operation. The following table summarizes the comparative analysis:

Table 2: Comparative Study

| Parameter | Non-Burst Interface | Proposed AXI |
|---|---|---|
| Address phase per transfer | Required every time [5][7] | Required once per burst |
| Address overhead in 8 beat transfer | 8 address phases[7] | 1 address phase |

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14, Issue 06, ACSCON - 2026

| | | |
|---|---|---|
| Address overhead reduction | 0%[4] | 87.5% |
| Burst support | Not supported[4] | Fixed, INCR, WRAP supported |
| Channel independence | Single combined channel[4] | Separate read and write channels |
| Functional correctness | Scenario dependent[1] | 100% in tested scenarios |

The Comparative Study demonstrates that the proposed AXI slave interface achieves improved transaction efficiency, strict adherence to AXI handshake protocols, and reliable functional verification [6][8]. The architecture is designed to handle burst based read and write transactions efficiently while maintaining synchronization between the master and slave components [3][7].

The modular finite state machine control structure enables clear separation of channel operations such as address reception, data transfer, and response generation, which improves system organization and simplifies debugging. In addition, the verification process confirms correct timing behavior and signal coordination across all AXI channels [10]. As a result, the proposed design provides enhanced scalability, better reliability, and improved communication performance when integrated within System on Chip based architectures [1][9].

## CONCLUSION

The proposed system presents the design and functional verification of an AXI slave interface using SystemVerilog and a structured UVM based verification framework. The architecture supports all five AXI channels and incorporates Fixed, INCR and WRAP burst modes to enable efficient memory transactions. A 32 bit wide and 128 depth memory model is used to validate read and write operations across multiple burst configurations. The design includes FSM based control logic for all channels, ensuring proper valid and ready handshaking, accurate burst tracking, and correct response generation. Simulation results confirm protocol compliance under various transaction scenarios, with constrained random and directed testing ensuring robustness. The Scoreboard based verification confirms that all transactions are executed without mismatch, demonstrating correctness and reliability of the proposed AXI slave interface.

The current implementation focuses on a single master and single slave configuration; however, the modular nature of the AXI protocol enables straightforward scalability to multi master and multi slave systems using AXI interconnects or crossbar switches with arbitration mechanisms such as round robin or fixed priority. Future work includes extending the design to support AXI4 Lite for low complexity control applications and AXI Stream for high speed data transfer scenarios. Additionally, incorporation of transaction identifiers can enable out of order execution and improve throughput in complex systems. Further enhancements include the integration of formal functional coverage and code coverage metrics to quantitatively evaluate verification completeness.

## IX. REFERENCES

[1] A. P. Sharma *et al.*, "Hardware optimization and implementation of a 16-channel neural tree classifier for on-chip closed-loop neuromodulation," *IEEE Trans. Biomed. Circuits Syst.*, vol. 19, no. 2, pp. 244–257, Apr. 2025.

[2] H. Rudramuniyappa *et al.*, "Optimized algorithms for FPGA implementation of PDCCH chain for 5G-NR base station," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 72, no. 7, pp. 3474–3487, Jul. 2025.

[3] T. Fischer *et al.*, "FlooNoC: A 645-Gb/s/link 0.15-pJ/B/hop open-source NoC with wide physical links and end-to-end AXI4 parallel multistream support," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 33, no. 4, pp. 1094–1107, Apr. 2025.

[4] J. Heo *et al.*, "Design space exploration of FFT architectures and numerical formats for SoC-based FMCW radar signal processing," *IEEE Access*, vol. 13, pp. 217364–217375, 2025.

[5] A. Yadav *et al.*, "FPGA-based medical image processing using hardware-software co-design approach," *IEEE Trans. Biomed. Circuits Syst.*, vol. 20, no. 1, pp. 57–68, Feb. 2026

[6] S. Cuccagna *et al.*, "Dynamic deployment of real time services in edge computing systems," *IEEE Open J. Commun. Soc.*, vol. 7, pp. 1163–1185, 2026.

[7] Z. Jiang *et al.*, "AXI-ICRT: Towards a real-time AXI-interconnect for highly integrated SoCs," *IEEE Trans. Comput.*, vol. 72, no. 3, pp. 786–801, Mar. 2023.

[8] T. Benz *et al.*, "AXI-REALM: Safe, modular and lightweight traffic monitoring and regulation for heterogeneous mixed-criticality systems," *IEEE Trans. Comput.*, vol. 74, no. 9, pp. 3072–3088, Sept. 2025.

[9] X. Yang *et al.*, "A high-performance on-chip bus (MSBUS) design and verification," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 7, pp. 1350–1354, Jul. 2015.

[10] F. Plasencia-Balabarca *et al.*, "A flexible UVM-based verification framework reusable with Avalon, AHB, AXI and Wishbone bus interfaces for an AES encryption module," in *Proc. IEEE Int. Conf.*, 2019, pp. 1–4.