iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** VI **Month of publication:** June 2024

DOI: https://doi.org/10.22214/ijraset.2024.63078

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue VI June 2024- Available at www.ijraset.com

Design of a Bus Communication Architecture using AXI Protocol based SoC Systems

Bandan Kumar Bhoi[1], Aditya Kumar Hota[2]
Department of Electronics, Telecommunication Engineering Veer Surendra Sai University of Technology, Burla

Abstract: ***The main objective of this project is to develop and implement a bus monitor that will allow system-on-a-chip (SoC) systems that employ the Advanced EXtensible Interface (AXI) protocol to have their performance monitored. Modern SoC architectures commonly employ the AXI protocol, which offers high-performance and effective communication between master and slave devices, to connect components. The bus monitor's real-time collection and analysis of AXI bus transactions allows for thorough performance analysis and SoC system debugging. Understanding transaction-level behaviour, spotting bottlenecks, and evaluating the overall effectiveness of the system are important goals. Verilog HDL will be used to implement the suggested bus monitor, which will then be incorporated into an FPGA-based development platform. To record transaction data, including read and write operations, addressing information, data transfers, and other metadata, it will interface with the AXI connection. A user-friendly interface will be used to process and display the collected data in order to provide a thorough performance analysis. With the creation of a specialised bus monitor for AXI-based SoC systems, this project hopes to enhance performance for a range of uses, such as networking, digital signal processing, embedded systems, and digital signal processing. The final product will be an invaluable tool for researchers, software engineers, and system designers working on intricate SoC designs that make use of the AXI protocol.***

I. INTRODUCTION

These days, a lot of SOCs use the advanced extensible interface, or axi, protocol. The Advanced Microcontroller Bus Architecture (AMBA) specification includes Axi. This AMBA AXI protocol is intended for high-performance, high-frequency system designs and has a number of characteristics that make it suitable for high-speed submicron connections.

Experts in the field created the excellent methodology known as VIP to cut down on time spent in test bench settings. Vip is a built-in architecture that is incorporated in SoCs and is used for on-chip transaction verification. This work builds a test bench for the purpose of verifying axi memory transactions and data transfer between locations throughout the read and write phases in both the same and separate addresses.

Here, the primary goal of the verification environment—which was created using the Verilog system—is to confirm the aforementioned memory transfers. The generator, mailbox, monitor, interface, coverage, bus functional model (BFM), and monitor are the primary VIP components. Each module in this case is made in accordance with its intended use [1].

The main objective of the generator is to generate different test cases for different applications. One kind of conduit for communication that allows for message exchange is a mailbox. Here is where the bfm and generator exchange information. Transactions are received from the generator and sent to the axi interface by Bfm, which is an essential part of the verification environment. The axi interface's functions include connecting the slave and master and assisting with handshaking during dialogue. We can only retrieve the outputs, which contain all of the transaction signals, by using the Axis interface. To ensure that transactions are authentic, Axi Monitor is employed in compliance with the handshaking protocol. Here, the monitor is meant to be used for transactions using memory. Only when all of the different modules are instantiated in the top module and appropriate synchronization is maintained between them will we obtain precise handshaking. The memory transactions in this study are

A. Test Case 1:Write and Read from the Same Location

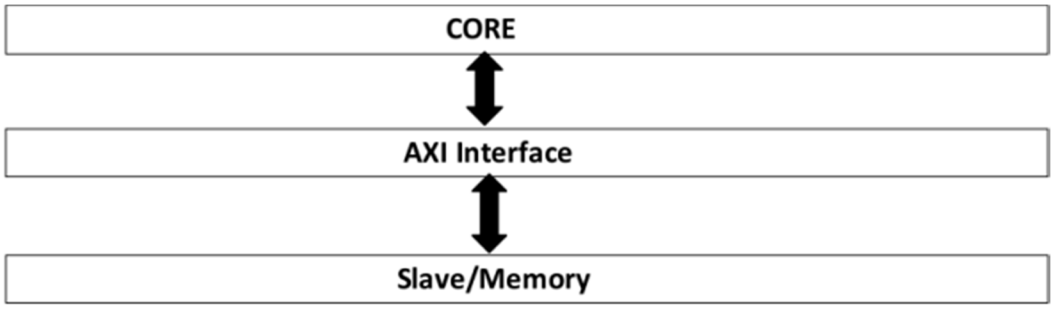
Test case 1: In this instance, the primary focus was on write and read transactions that took place at the same address. Axia protocol, on the other hand, divides write data operations into numerous phases and write address operations into a single phase [2]. The write address signal, or AWADDR, is a 32-bit signal that designates the specific address where numerous pieces of data need to be written. Similarly, read data operations happen in numerous phases, and read address operations happen in a single step. ARADDR stands for read address signal, a 32-bit signal. The signals in this test case must have the same for both ARADDR and AWADDR, which also happen at the same place.

©IJRASET: All Rights are Reserved

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue VI June 2024- Available at www.ijraset.com

B. Test case 2:write and read from the different location

Test case 2: Testcase 2 differs greatly from Testcase 1. In this instance, read and write take place at distinct places. The fact that the AWADDR and ARADDR signals in this test instance originate from different addresses suggests that the write and read signals ought to originate from separate addresses. In this work, we are practically testing the aforesaid test scenarios with the aid of collected waveforms.



[Fig 1.1 block diagram of AXI]

Fig. 1 AXI Block diagram

II. RELATED WORK

Research and development is now underway in the domain of bus monitor design for performance analysis of AXI protocol- based System-on-Chip (SoC) devices. AXI-based interconnects in SoC designs have been the subject of several related publications that have investigated related ideas and approaches with the goal of improving the effectiveness and precision of such analyses.

The following are some significant findings from previous studies:

1) *Bus Functional Models (BFMs):* To simulate AXI transactions and confirm system behaviour, Bus Functional Models (BFMs) have been used in numerous prior research. BFMs are transaction-level models that can produce AXI traffic for testing and validation. They are commonly implemented in Verilog or System Verilog. These models play a key role in confirming functional accuracy and conformance with the AXI protocol.
2) *Performance Profiling Tools:* A number of tools have been created to profile AXI-based SoC systems' performance. These tools provide insights into latency, throughput, and other important parameters while capturing AXI transactions during runtime. Instruments for performance profiling assist in locating bottlenecks and improving system setups [3]
3) *FPGA-based Monitoring Solutions:* AXI protocol analysis-specific FPGA-based monitoring solutions have been the focus of some research. In order to record and analyse AXI transactions in real-time and offer comprehensive insight into system-level interactions, these solutions frequently include the creation of bespoke hardware designs
4) *Integration with System-Level Simulation Environments:* System C or System Verilog are two examples of system-level simulation environments that may be integrated with AXI bus monitors to provide a comprehensive analysis of SoC performance. Co-simulation of hardware and software components is made possible by this integration, allowing for a thorough examination of AXI-based communication.
5) *Open-Source Initiatives:* To meet the requirement for AXI performance analysis and monitoring, several open-source initiatives have been developed. These projects frequently include reusable parts, like software libraries or Verilog modules, to make it easier to create unique monitoring systems for AXI-based SoCs design.

By leveraging insights from these related works, the design of a bus monitor for performance analysis of AXI protocol-based SoC systems can benefit from established methodologies and techniques. The goal is to contribute to the advancement of tools and methodologies for optimizing the performance and efficiency of complex SoC architectures utilizing the AXI protocol.Top of Form
An open standard protocol called Advanced Microcontroller Bus Architecture (AMBA) is used for on-chip interconnects, which are specifications for connecting and managing functional blocks within a system-on-chip (SoC). Small-scale SoCs can readily use the AMBA bus. Because of its efficiency, the AMBA bus has thus served as the SOC market's representative. The AMBA specification defines three different buses: Advanced Peripheral Bus (APB) Advanced High performance Bus (AHB) Advanced extensible Interface Bus (AXI)

©IJRASET: All Rights are Reserved

The AMBA standard details all of the signals, transfer methods, structural layout, and other bus protocol details for the APB, AHB, and AXI buses. The AMBA APB is secondhand to communicate with any peripheral that has little bandwidth and doesn't require the high performance of a pipelined bus interface.

APB peripheral device that meet the specifications listed below can be easily incorporated into any design process:

a) Devices with low-speed peripheral buses • A non-multiplexed, synchronous bus 8, 16, 32-bit data bus, 32-bit address bus, single master (bridge)

b) Non-pipelined with the following specifications, AMBA AHB is a new bus level that operates above APB and incorporates the capabilities needed for high-performance, high clock frequency systems.

c) Split transactions; burst transfers • Greater data bus configurations (64/128 bits);

d) Single clock edge operation;

e) Single cycle bus master handover; AXI enhances the functionality of the AHB bus and supports the next generation of high act SoC architectures. The areas of the AXI bus protocol are backward compatibility with AMBA AHB and APB inter- faces, litheness in summit the boundary and act requirements of a wide range of components, and support for high frequency operation without the need for complex bridges. The features of the AXI protocol are as follows: • Support for non-aligned addresses, Distinct control, and data phases and the capacity to issue several pending addresses.

III. BUS MONITOR

Creating a specialised module that watches and records AXI transactions taking place within the SoC is necessary when designing a bus monitor for performance monitoring of AXI protocol-based System-on-Chip (SoC) systems. This bus monitor is essential for examining the AXI interconnect's efficiency, throughput, and latency since it offers insightful data that can be used to optimise the system and improve performance. The following are important factors and parts that are usually included in the bus monitor design for AXI protocol analysis [4]:

1) *Transaction Observation:* AXI transactions that take place on the connection must be seen by the bus monitor. This entails recording data, address, response status, data, transaction type (read or write), and other pertinent information.

2) *Interface:* The AXI interconnect in the SoC is directly interfaced with by the bus monitor. Depending on the design needs, it may connect to the interconnect via AXI protocol interfaces (AXI4, AXI4-Lite, etc.).

3) *Transaction Decoding and Analysis:* The bus monitor decodes and examines AXI transactions to derive important performance indicators. This include figuring out any bottlenecks as well as latency, or the amount of time it takes for a transaction to finish [5].

4) *Data Storage and Logging:* The bus monitor logs transaction data to an external storage medium or keeps it in memory to enable thorough performance analysis. For the purpose of post-analysis and system behaviour visualisation, this data logging is essential.

5) *Integration with Monitoring Tools:* To offer real-time insights into system behaviour, the bus monitor may integrate with analysis frameworks or performance monitoring tools. The performance of the AXI interconnect may be continuously observed and assessed thanks to this integration.

6) *Hardware Implementation:* The bus monitor can be implemented using either software (running on the SoC or included in firmware) or hardware (FPGA-based), depending on the needs and application. For precise monitoring, hardware implementations provide direct access to AXI signals.

7) *Validation and Verification:* To guarantee correct transaction capture and analysis, the bus monitor is put through validation and verification procedures prior to deployment. Simulation environments and test benches are used to verify the functionality in different scenarios [6].

8) *Customisation and Adaptability:* A well-thought-out bus monitor has to be both flexible and able to accommodate various system architectures and AXI configurations. Its adaptability makes it possible for it to be easily included into a variety of SoC architectures.

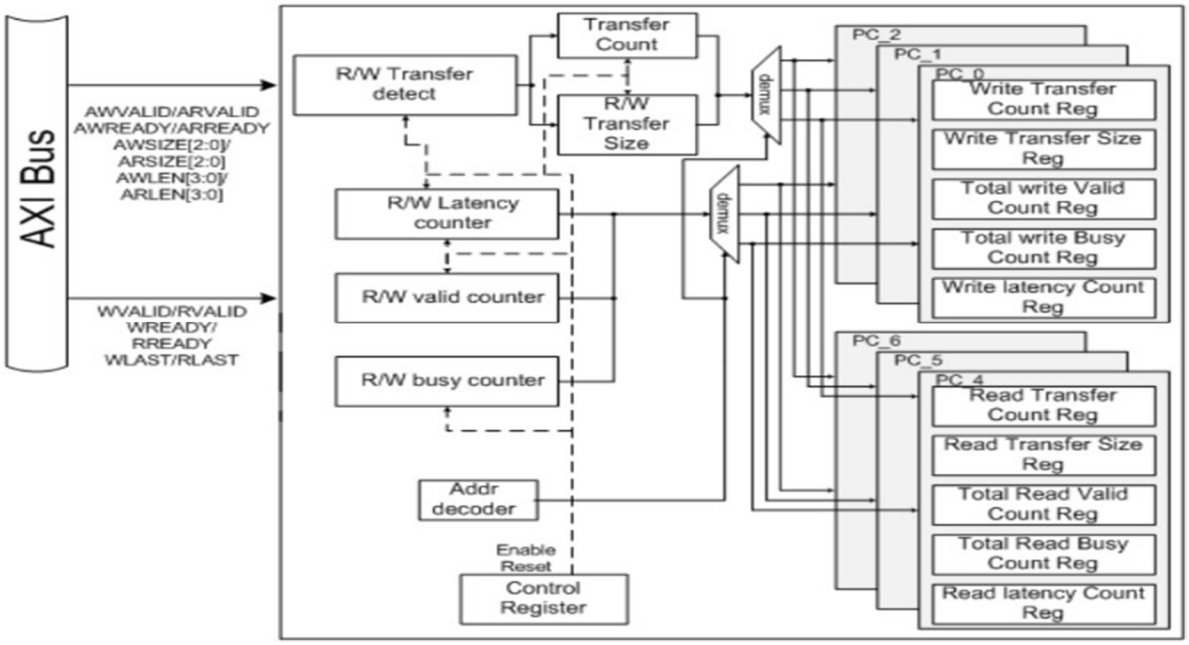


Fig. 2

IV. ARCHITECTURE OFAXI PROTOCOL

A. Principal Characteristics

• Phases for data and address/control are distinct. • 4 bit byte strobe data transmissions that are not aligned. • Parallelism is achieved by having separate read and write data channels. • Possession of several outstanding addresses. • Completing transactions out of order. • The interconnect's simple register stage adding process makes timing closure possible. • It makes a distinction between instructions and data. • Low power operation support signals.

Fig. 3 Architecture of AXI

V. SYSTEM ARCHITECTURE

The following blocks are part of the suggested architecture:
Master. Slave. Interconnect.
According to the protocol, five separate Channels must be used. Write Address Channel
Write Data Channel Write Response Channel Read Address Channel Read Data Channel
The following blocks make up the interconnect:
Master Controller Arbiter
Decoder

Slave Controller as shown in Fig

1) The interconnect have two arbiters: one for the read group and one for the write group of channels. Each arbiter grants access to a single master at a time who has requested access to the write or read groups of channels. Because of this, two different masters can transfer data simultaneously, one of them reading the data while the other writes it[7].
2) The address from which the decoder derives the slave address is AWADDR [31:0], ARADDR [31:0], which correspond to the address of the memory location for the Write or Read operation.4. Each intermediary grants access to one master at a time who has requested it. Consequently, data can only be transferred simultaneously by two different masters, one for the read operation and one for the write action.
3) The Master Controller complexes the output bus signals from the authorized master before sending them to the Slave Controller and other blocks. Correspondingly, it demultiplexes the signals from the Slave Controller onto the authorized master's input bus signals using the arbiter grant control signal.
4) In a similar manner, the Master Controller's signals are demultiplexed against the input bus signals of the chosen slave via the interpreter control signals, and the Slave Controller multiplexes the output bus signals from the chosen slave through the decoder control signals to the Master Director & further blocks.

This model included five decoders, one per each channel. Multiple outstanding transactions are made possible by this.

TABLE I
CHANNEL WIDTH IN AXI

Channel widths	
Channel	Width (bits)
Write Adress Channel	56
Write Data Channel	43
Write Response Channel	8
Read Adress Channel	56
Read Data Channel	41

Write channel signal(write address)

VI. CHANNEL SIGNALS

Table II
Write Channel Signal

Write address[AW]
AWREADY
AWADDR[32]
AWSIZE [3]
AWBURST [2]
AWCACHE [4]
AWPROT [3]
AWID[x:0]
AWLEN [4] AWLEN [8]
AWLOCK [2]
AWQOS [4]
AWREGION [4]
AWUSER[x:0]

Following are (write data)
The followings are Write response

VII. READ CHANNEL SIGNALS

Read Address
Read Data
The followings are Read Data

TABLE III
WRITE DATA SIGNALS

Write Data [W]
WVALID
WREADY
WLAST
WDATA[x:0]
WSTRB[x:0]
WID[x:0]
WUSER[x:0]

TABLE IV
WRITE RESPONSE SIGNALS

Write Response(B)
BVALID
BREADY
BRESP [2]
BID[x:0]
BUSER[x:0]

TABLE V
READ CHANNEL SIGNALS

Read Address[AR]
ARVALID
ARREADY ARSIZE
[3] ARADDER[32]
ARBURST [2]
ARCACHE [4]
ARPROT [3]
ARID[x:0]
ARLEN [4] ARLEN [8]
ARLOCK [2]
ARQOS [4]
ARREGION [4]
ARQOS [4]
ARUSER[x:0]
ARLOCK [2]
ARLOCK
ARREGION [4]
ARUSER[x:0]

©IJRASET: All Rights are Reserved

TABLE VI
READ DATA SIGNALS

Read Data[R[
RVALID
READY
RLAST
RDATA[x:0]
RRESP [2]
RID [x:0]
RUSER[x:0]

VIII. AXI PROTOCOL SPECIFICATION

Many master and slave devices coupled by the linking protocol make up a typical system-on-chip design. For the following interfaces, the AXI protocol offers a single interface definition: • Between an interconnect and a master• Between a subordinate and the • Between a slave and his owner.

IX. BURST OPERATION IN AXI

Burst-based transaction functionality is provided by the AXI protocol. Address and control information are sent over the address channel describing different aspects of the data to be transmitted, and they are part of every transaction. Data is transferred between the master and slave via a write data channel. The additional write response channel in the AXI protocol is used by the slave to acknowledge write transactions by sending a response to the master. The AXI protocol makes it possible to: • handle many open transactions; • issue address information prior to the actual data transfer. • assistance with transaction completion that occurs out of order [7]. Three types of transaction bursts are defined by the AXI protocol and are explained in: • Static burst;

- Gradual burst;
- Wrapping burst

TABLE VII
CONTROL INFORMATION SIGNALS

CONTROL IN-FORMATION SIGNALS	
ARLEN [3:0]	
AWLEN [3:0]	1,2 ,3,4,5,6,78,9
ARSIZE [2:0]	10,
AWSIZE [2:0]	1,2 ,4,8,16,32,64,128
AWBURST [1:0]	By ers ers XED,
ARBURST [1:0]	FI INCR,WRAP, Reserved

X. PROPOSED WORKS

The purpose of this study is to verify the design using System Verilog once Verilog has been used to establish communication between a slave and a master. Design of AXI Protocol: With a clock cycle duration of 10ns and an operational frequency of 100MHz, the AMBA AXI4 slave may send up to 256 data transfers in a single burst. The AMBA AXI4 system component consists of a master and a slave, as shown in Fig. 1. There are five channels that connect the AXI slave and master: write address, write data, read data, read address, and write response.

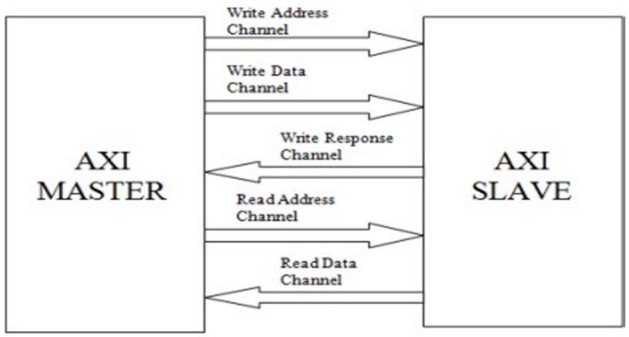


Fig -1: Block Diagram of a system
Fig. 4 AXI System block diagram

XI. BLOCK DIAGRAM OF AXI SYSTEM

Every transfer in the AXI protocol uses the handshake procedure. The same VALID/READY handshake is used by every channel to transfer control and data. Both of the master and slave can regulate the speed at which data and control data travel thanks to this two-way flow control system. It will help to communicate the data or control information the source generates the VALID signal. To signify that it has accepted the data or control data , the destination creates the READY signal. Only when both the READY and VALID signals are HIGH does transfer take place. On both the master and slave interfaces, there cannot be any combinatorial routes connecting the input and output signals.

A. Address Write Channel (AW Channel)

AXI_MASTER energies the write grasp signals while ARESET n is HIGH; otherwise, it initiatives all signals as 0. The address write command signals AWID, AWADDR, AWBURST, AWLEN, AWSIZE, AWCACHE, AWLOCK, and AWPROT are driven by the AXI_MASTER. When AWVALID is set to HIGH, the driven signals are legitimate. The AXI_MASTER doesn't driven by the AWVALID signal as LOW until the AWREADY signal, which is driven by the DESTINATION_SLAVE and indicates that it has received the address write instruction signals. If AWREADY is LOW, AXI_MASTER values stay the same. Fig. 2 displays the address write command signals state diagram [8].

XII. WRITE DATA CHANNEL (W CHANNEL)

Following the transmission of the write address grasp signals, the AXI MASTER drives this Write Data signals. Only when ARESET n is HIGH does it drive these signals; in other cases, it drives all signals to zero. When the WREADY signal is received, AXI MASTER drives the WDATA signal by WVALID set to HIGH and maintains that value. WREADY drives the subsequent WDATA if it is HIGH. The AWLEN No. of data is driven by AXI MASTER. It pushes the WLAST as HIGH while driving the most recent data. The WRITE DATA channels' state diagram is displayed in Fig. 3.

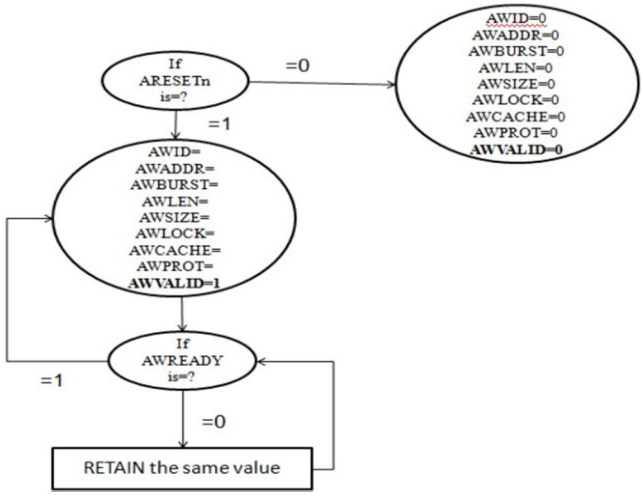


Fig. 5 AW Channel signals

XIII. WRITE RESPONSE CHANNEL (B CHANNEL)

Only when ARESETn is HIGH the DESTINATION_SLAVE drive Write Response signals; in other cases, it energies all signals as 0. Awaiting the WLAST signal is DESTINATION_SLAVE. It energies these response signals, with BVALID set to HIGH, upon receiving the WLAST signal. Till it receives the AXI MASTER's BREADY signal, it remains at that value. When ACLK's next positive edge approaches, all signals are driven to zero if BREADY is HIGH; otherwise, its value remains unchanged.

XIV. ADDRESS READ CHANNEL (AR CHANNEL

Only when ARESET n is HIGH does AXI_MASTER drive the command signals; in other cases, this determines all signals as 0. The AXI_MASTER drives the following address read command signals: ARID, ARADDR, ARBURST, ARLEN, ARSIZE, ARCACHE, ARLOCK, and ARPROT. When ARVALID is set to HIGH, it means that the driven signals are legitimate. Until the AXI_MASTER receives the ARREADY signal—which is obsessed through the SOURCE_SLAVE—as confirmation that it has received the address read instruction signals, it doesn't drive the ARVALID signal as LOW. The values of AXI_MASTER remain unchanged if ARREADY is LOW.

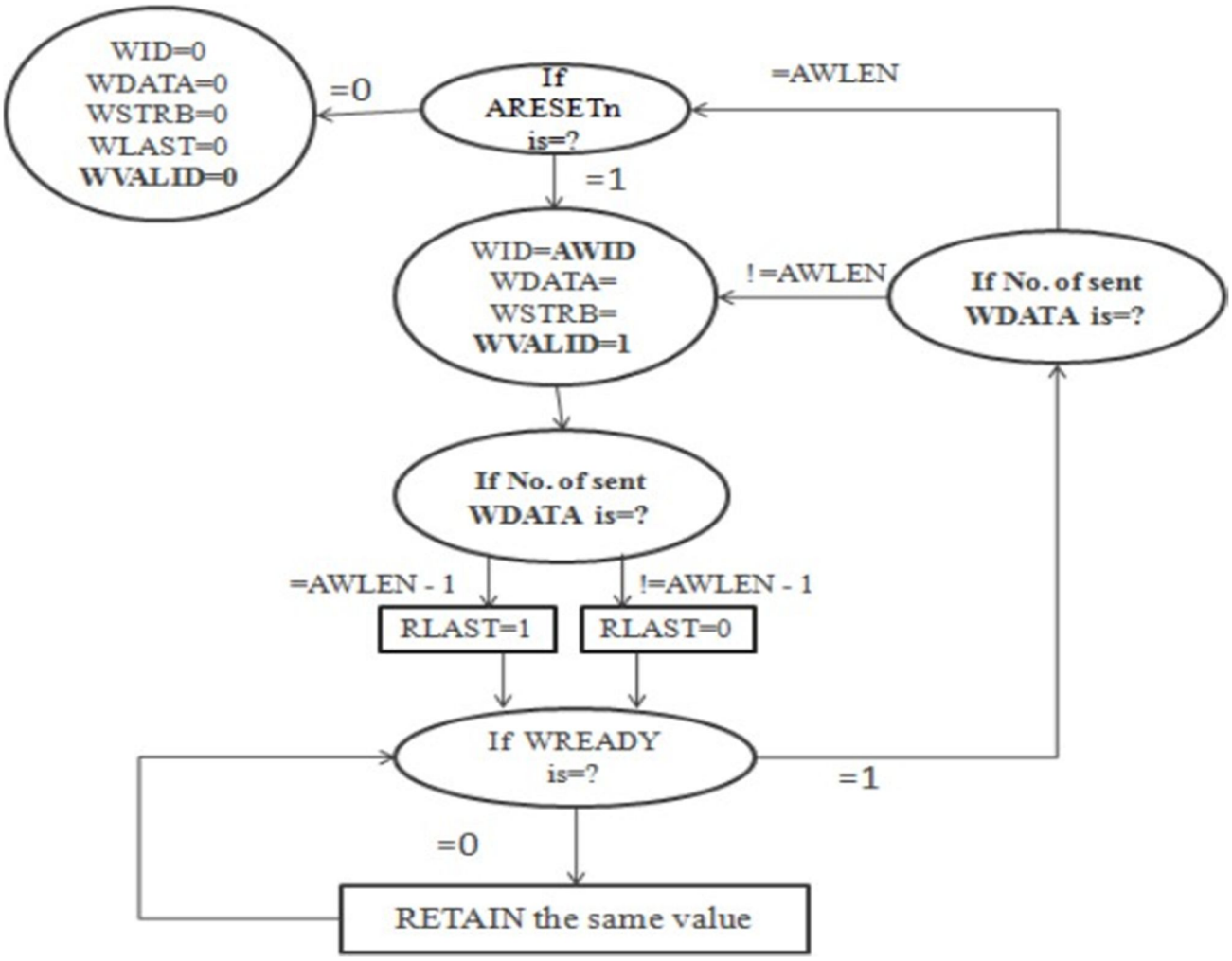


Fig. 6 W channel

A. Read Data Channel (R Channel)

The SOURCE_SLAVE initiatives this Read Data signals after receipt the read command signals. This signals are individual driven when ARESETn is HIGH; or else, all signals are driven as zero. SOURCE_SLAVE drives the RDATA signal with RVALID set to HIGH until it gets the RREADY signal. When RREADY is HIGH, the next RDATA is driven. SOURCE_SLAVE is the data type that powers ARLEN No. Even though lashing the last data, it drives the RLAST as HIGH [9].

B. AXI Transaction

AXI supports a wide range of transactions. Type of burst: Three types of fixed increment and wrap are combined: with fixed, the address is the same for each transfer. Every transfer in an incrementing transfer is the sum of all previous transfers. The exchange measure is necessary for the estimation of augmentation. Wrap is similar to an addition of previous exchanges added to each exchange address of burst. When it gets there, though, it wraps the address around the edge. In queue and out of queue To get the addresses of transfers during a burst, use the following equations [10]:

Start Address = ADDR Number Bytes = 2SIZE Burst Length = LEN + 1 Aligned Address = (INT(Start Address / Number Bytes)) x Number Bytes.

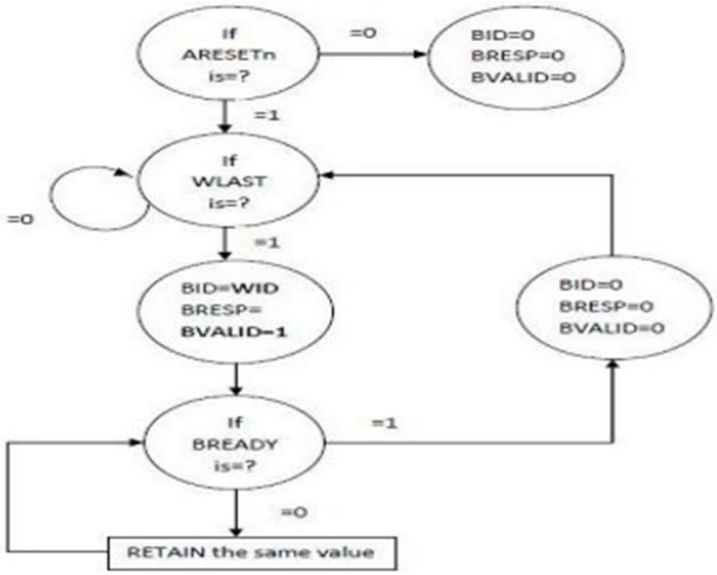


Fig. 7 B Channel

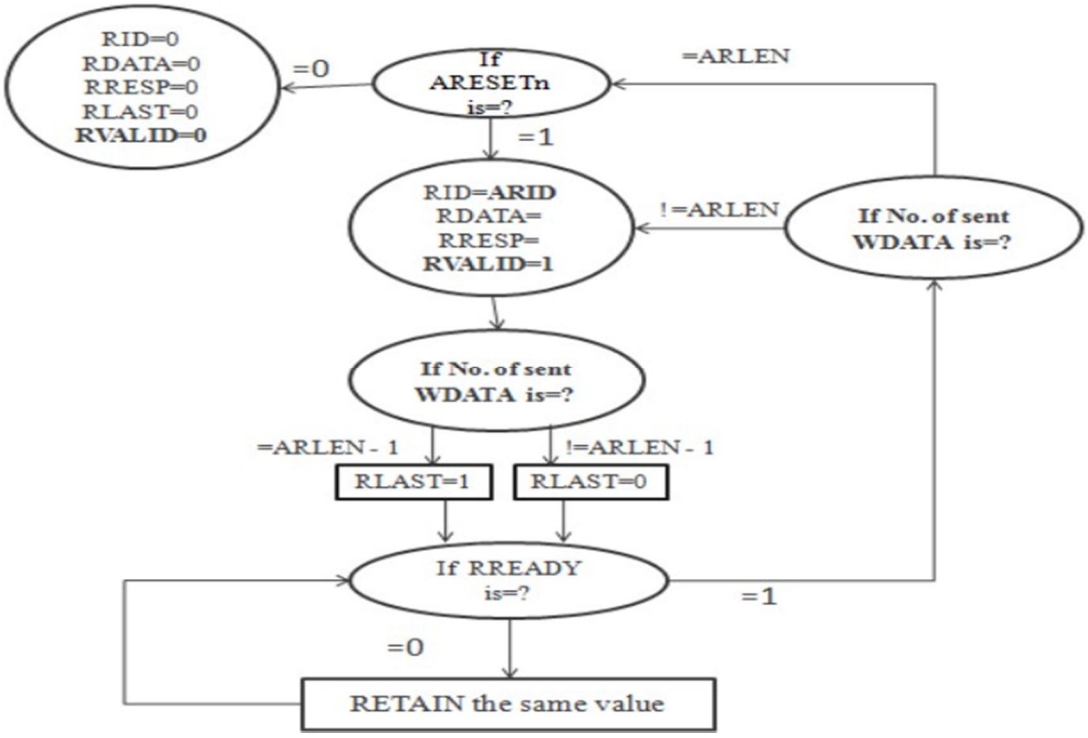


Fig. 8 R Channel

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue VI June 2024- Available at www.ijraset.com

C. The AXI Protocol Verification Environment :

System Verilog is used to provides the authentication environment for the AXI bus, which is displayed below. This environ- ment's hierarchical layer structure makes it easier to maintain and repurpose with various designs undergoing verification [11].

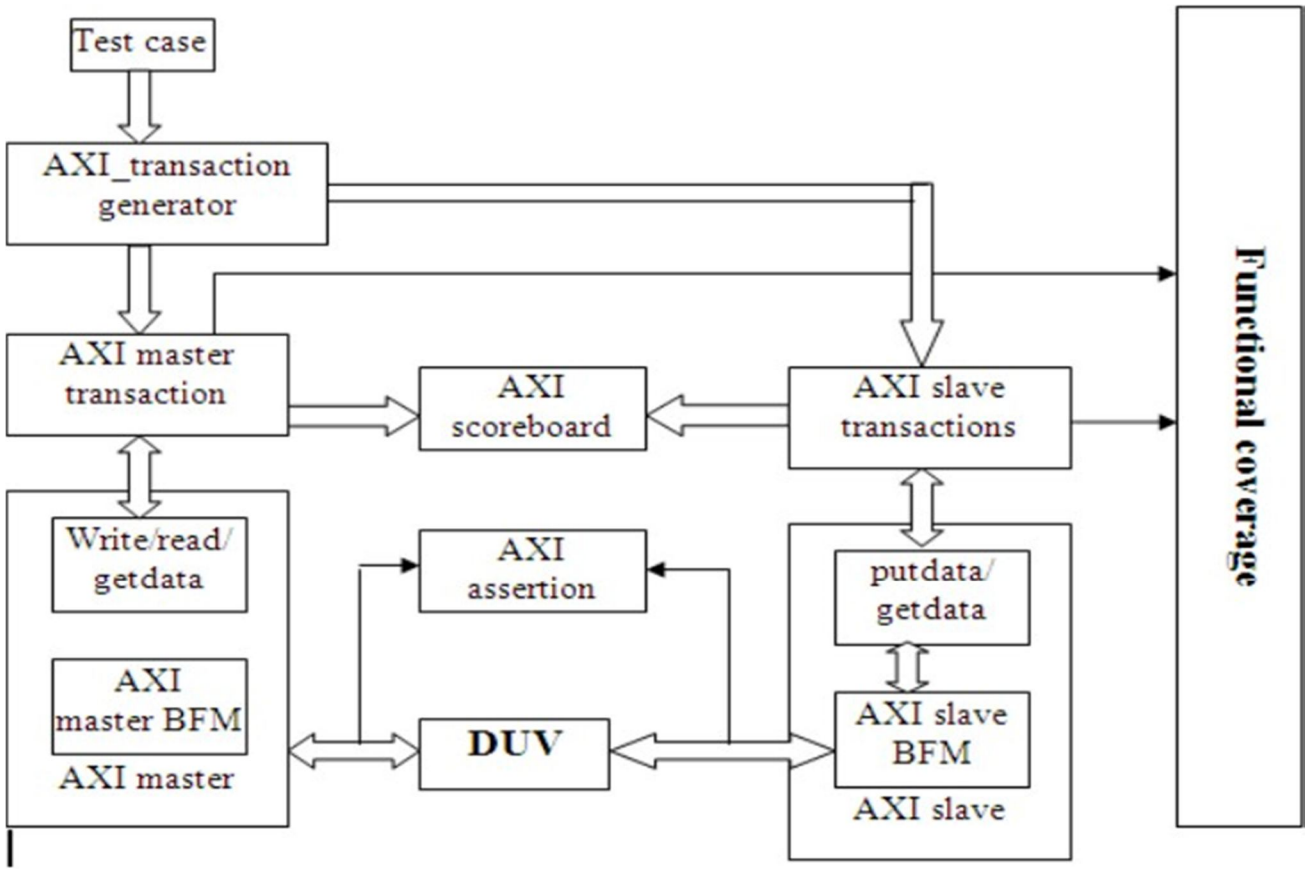


Fig. 9 AXI Protocol verification environment

D. Test Case

Test Example :

The list of test cases is included in the „Test case". Every test case is linked to "sequences" that are written for various scenarios, such as

single_write_operation [12]. write-followed-read-operation, multiple write,

single read, multiple reads

To confirm the design for a specific scenario, any test case is linked to the Verification Environment [6].

Transaction Generator (AXI) The "sequence item" is another name for the transaction generator. A class called Se- quence_item has all port signals as one of its properties. The "rand" keyword is used to declare each of these signals, allowing this class to allot the random rate to each signal after using the randomize method. The DUV is later assigned these generated input values [13].

XV. AXI MASTER TRANSACTION

It consists of the signals that the master drives. There is an instance of the AXI_transaction_generator in this class. The values produced by the AXI_transaction_generator may be overridden by the master transaction. The values produced by the AXI transaction generator are sent to the DUV assuming that no signals have been overridden.

XVI. AXI_SLAVE_TRANSACTION

AXI_Master_Transaction-like functionality is included, with the exception that signals driven from the slave are also in- cluded.

©IJRASET: All Rights are Reserved

XVII. AXI_SCOREBOARD

The AXI_scoreboard additionally stores the standards produced in the AXI_Master_Transaction and AXI_Slave_Transaction. These signals can subsequently be used to compare the expected and actual outputs.
Useful CoverageThis class contains a collection of several coverage scenarios that are used to determine the extent to which this design is protected through verification. The classes AXI Master Transaction and AXI Slave Transaction will raise this reporting.

XVIII. AXI_MASTER

Its the important wedge of master part, including two subblocks like, Write/read/get data &AXI master BFM. Read/write/get data: The sequencer, driver, and monitor objects belong to the classes represented in this sub-block. The sequencer selects and inserts the desired sequence into the driver. In compliance with the protocol, it drives these signals. Monitor checks whether or not signals are changing in accordance with procedure [14].

XIX. AXI MASTER BFM

It is the class that contains the bus-related functions. Bus Function Modules is an acronym for BFM. Ultimately, the DUV receives the signals that the driver is driving.The functionality of AXI_Slave and AXI_Master is comparable.

XX. AXI_ASSERTIONS

The list of assertions written in accordance with the signal description is included. Assert statements are used to write these. Prior to being applied to the DUV, these claims are applied to the driver's driving signals [15].

XXI. SYSTEM VERILOG

Its known as Hardware Verification Language (HVL). The main purpose is verification. The test bench (TB) is first written in the Verilog language using household tasks and purposes [11]. However, the procedure remained extremely drawn out. It circumvents this laborious process. Concepts like the notion of OOPs, randomization, and constrained randomization are all included in System Verilog, an updated version of Verilog. These tools enable us to rapidly generate any potential set of inputs, which lets us validate the design step-by-step [16].

A. RTL Design

////////write address channel (aw)
logic awvalid; /// master sends new address

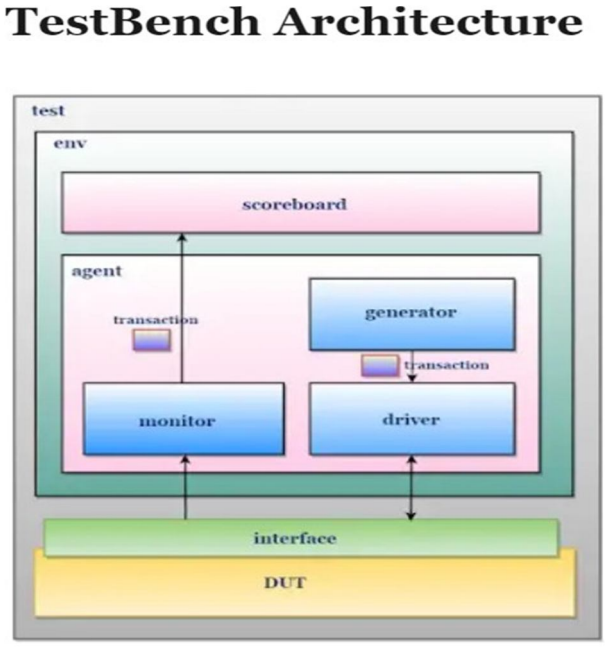


Fig. 10 system verilog architecture

logic awready; /// slave is ready to accept request
logic [3:0] awid; ////// exclusive ID for each transaction
logic [3:0] awlen; ////// burst length AXI3 : 1 to 16, AXI4 : 1 to 256 logic [2:0] awsize; ////unique transaction size : 1,2,4,8,16 ...128 bytes logic [31:0] awaddr; ////write adress of transaction
logic [1:0] awburst; ////burst type : fixed , INCR , WRAP
/////////write data channel (w)
logic wvalid; //// master sends new data
logic wready; //// slave is ready to accept new data logic [3:0] wid; /// exclusive id for transaction logic [31:0] wdata; //// data
logic [3:0] wstrb; //// lane having valid data logic wlast; //// last transfer in write burst
/////////write response channel (b)
logic bready; ///master is ready to accept response logic bvalid; //// slave has valid response
logic [3:0] bid; ////unique id for transaction logic [1:0] bresp; /// status of write transaction
//////////////read address channel (ar)
logic arvalid; /// master sends new address logic arready; /// slave ready to accept request
logic [3:0] arid; ////// exclusive ID for each transaction
logic [3:0] arlen; ////// burst length AXI3 : 1 to 16, AXI4 : 1 to 256 logic [2:0] arsize; ////unique transaction size : 1,2,4,8,16 ...128 bytes logic [31:0] araddr; ////write adress of transaction
logic [1:0] arburst; ////burst type : fixed , INCR , WRAP
////////// read data channel (r)
logic rvalid; //// master is sending new data
logic rready; //// slave is ready to accept new data logic [3:0] rid; /// unique id for transaction
logic [31:0] rdata; //// data
logic [3:0] rstrb; //// lane having valid data logic rlast; //// last transfer in write burst logic [1:0] rresp; ///status of read transfer

XXII. RTL DIAGRAM 4 1

(Initial RTL Fig 4.2) (Data types size)

A. *Comparison Between Others*

TABLE VIII
COMAPRISON BETWEEN AXI AND AHB

AXI	AHB
Advanced eXtensible Interface (AXI)	Advanced High-performance Bus (AHB)
Complete transfer are supported	Exclusive transfer are not supported
Axi has 1 read & write address cannel 1 read & write data channel& 1 write response channel	It has 1 read & write data channel,1 address channel
Extremely high outturn	Limited outturn
Advanced powerdispersion	Low power dispersion
Number of cables are more due to 5channels	Numbers of channels are less
Write strobes are supported	Write strobes are not shore up
Axi supports Ax USER bits i.e,it supports side -band signals	No essential supports for side -band signals

B. *Result Analysis*

This section presents the findings from the memory transaction verification for the AXI protocol, which is simulated with VIVADO and modeled in the Verilog system. By encoding AWSIZE, which is transmitted from master to slave via the address channel of the AXI bus, the total transfer size outputs are acquired. By looking at the values of AWVALID/AWREADY and ARVALID/ARREADY, the total transfer count provides the number of transactions. Total busy counts count the number of clock cycles on which the transactions are in the busy state and verify if the master and slave are enabled (Design of a Bus Monitor for

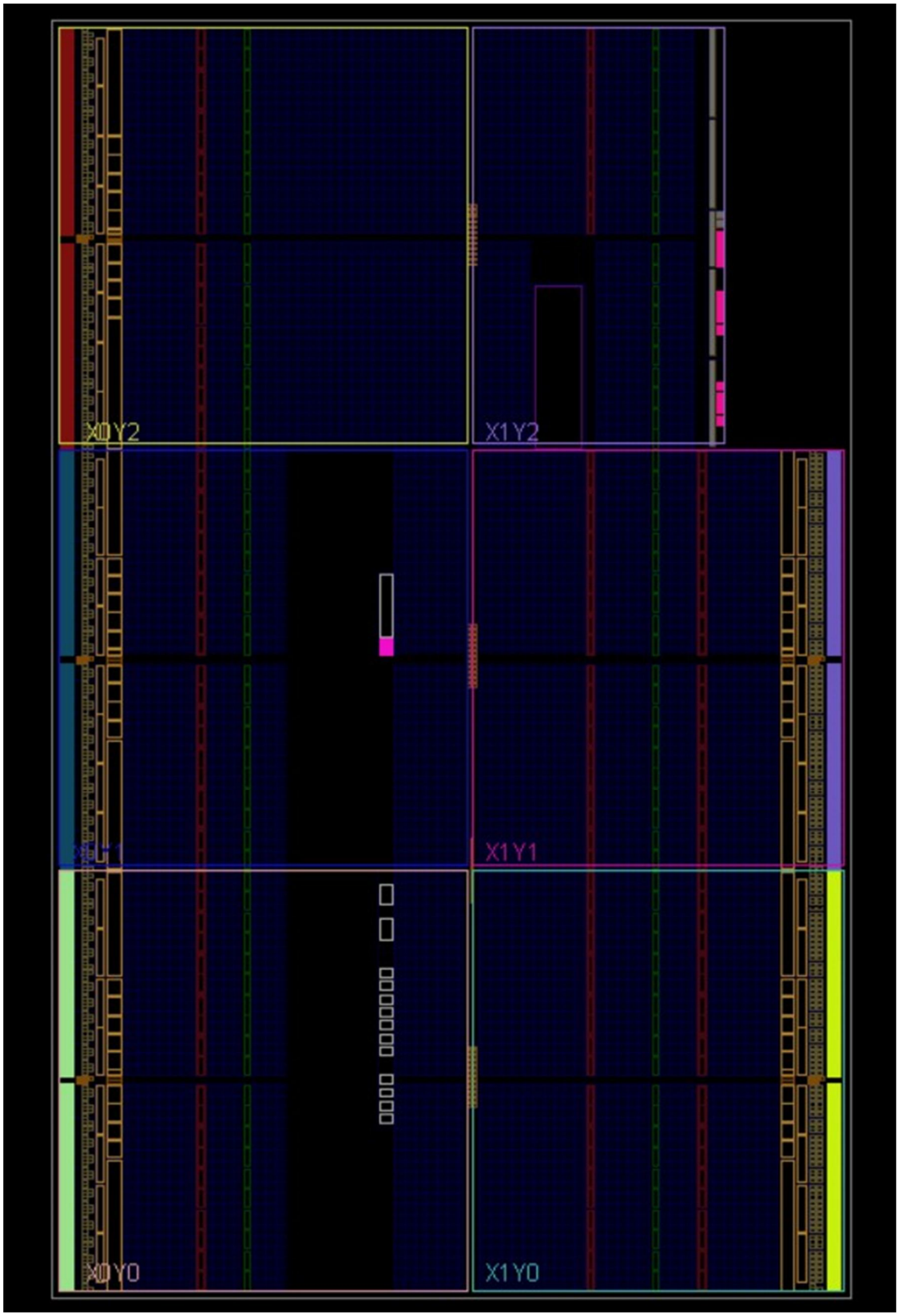


Fig. 11

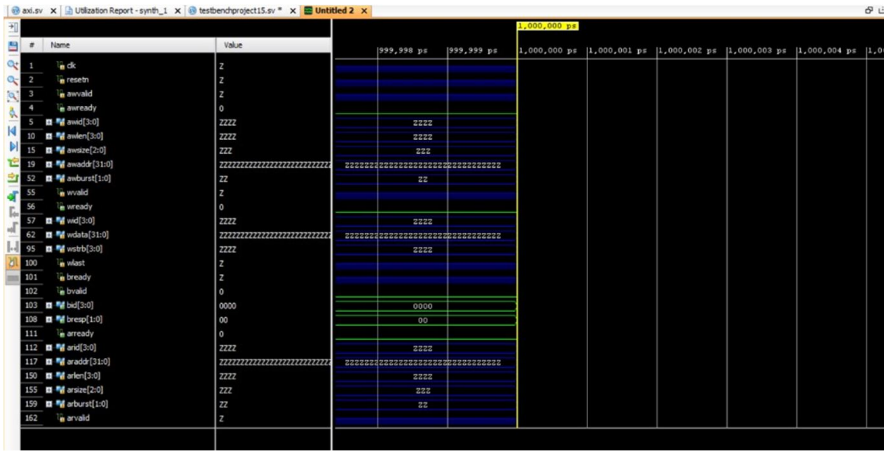


Fig. 12

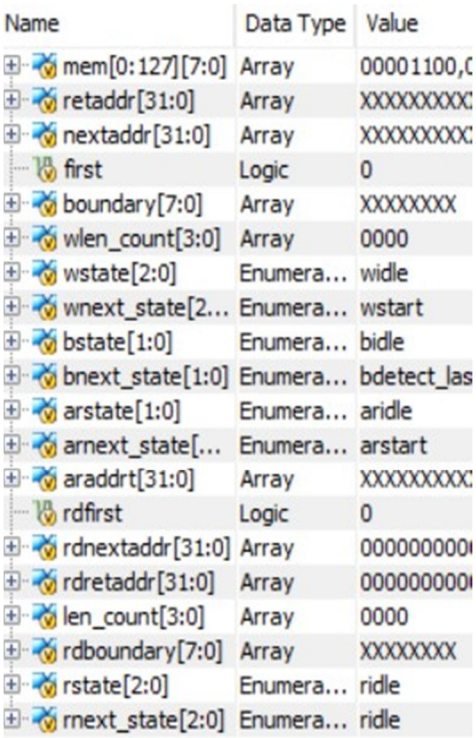


Fig. 13

Performance Analysis 6319). The number of clock cycles needed to transport the real quantity of data is also counted in the total valid count. The write delay is

By counting the clock cycles needed for the master to finish its transaction, write latency can be calculated. Since the interval depends on the master's operation time rather than the bus state, there is no need to count from the address transaction. The number of clock cycles from the beginning of the address transaction to the end of the data transaction is used to calculate read latency.

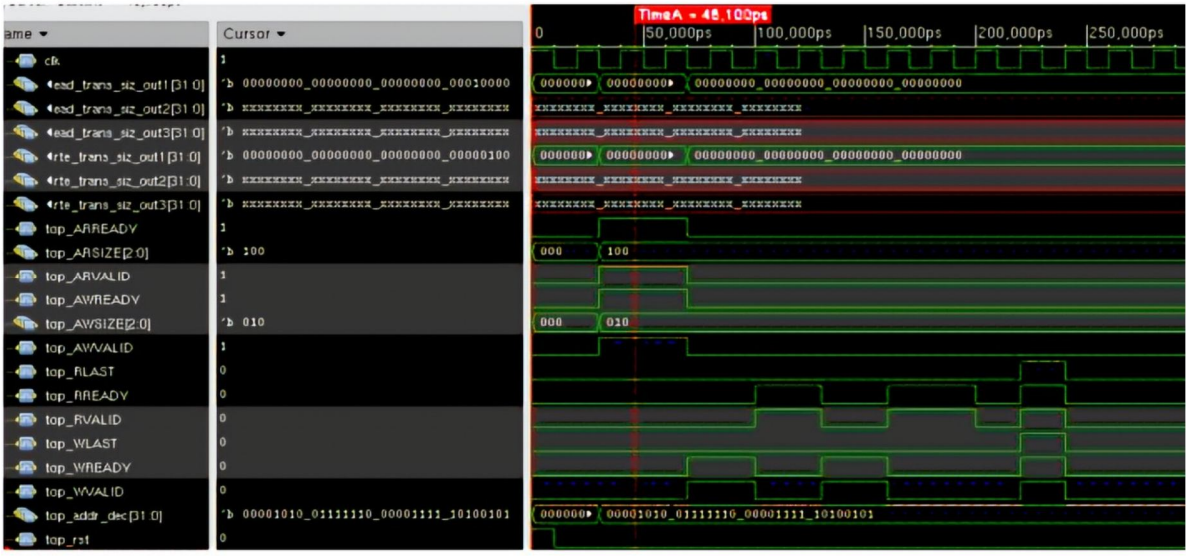


Fig. 14 Total transfer size

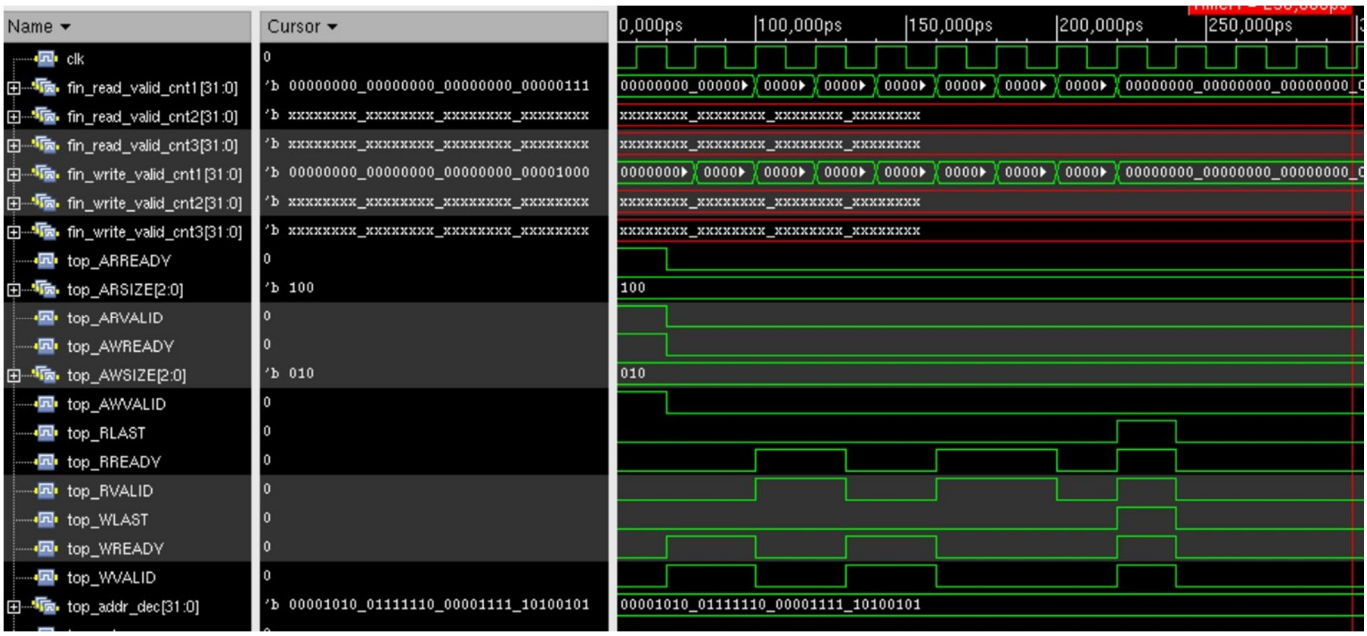


Fig. 15 Total valid count

C. Utilization Report

1) Conclusion

It becomes more difficult to improve system performance as SoC complexity rises. Thus, it becomes crucial to comprehend underlying system behavior and develop the performance monitor. The purpose of this research was to design bus monitor as an AXI protocol performance analyzer. Deriving important information about the data exchanges between the bus protocol's master and slave can be greatly aided by this. High frequency SoC applications that support up to 194.62 MHz can use the bus monitor.

2) Future Scope

The AMBA AXI has boundaries in terms of the data chosen provided regarding burst and beats. The burst data cannot exceed the 4k limit. More than sixteen beat bursts are only maintained with the INCR burst type. For both the WRAP and FIXED

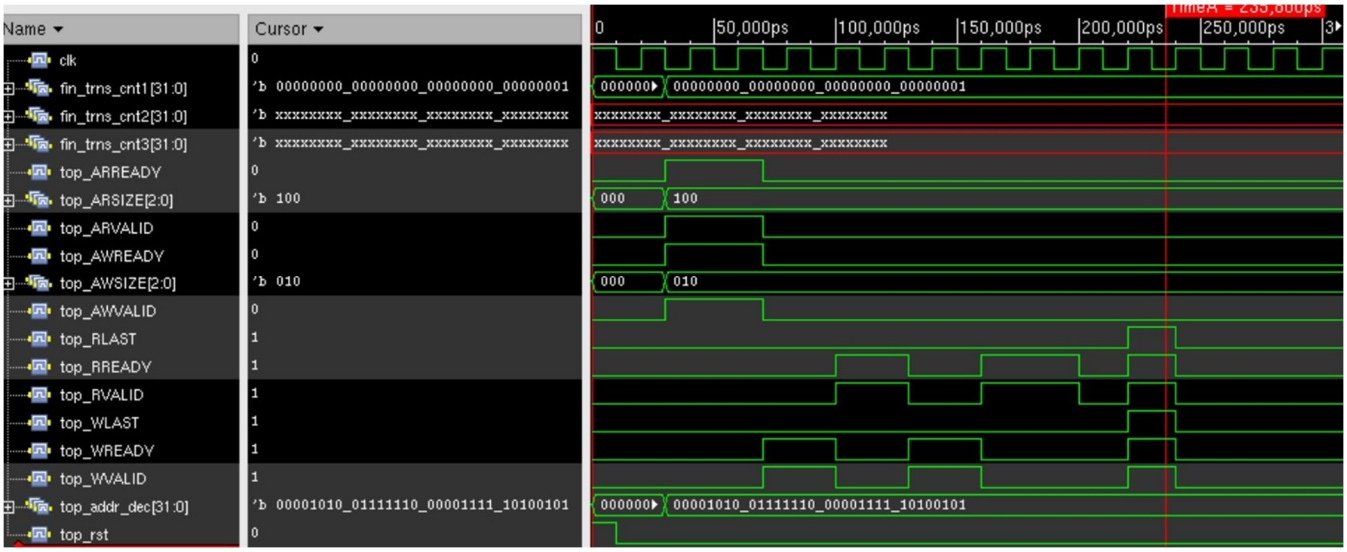


Fig. 16 Total transfer count

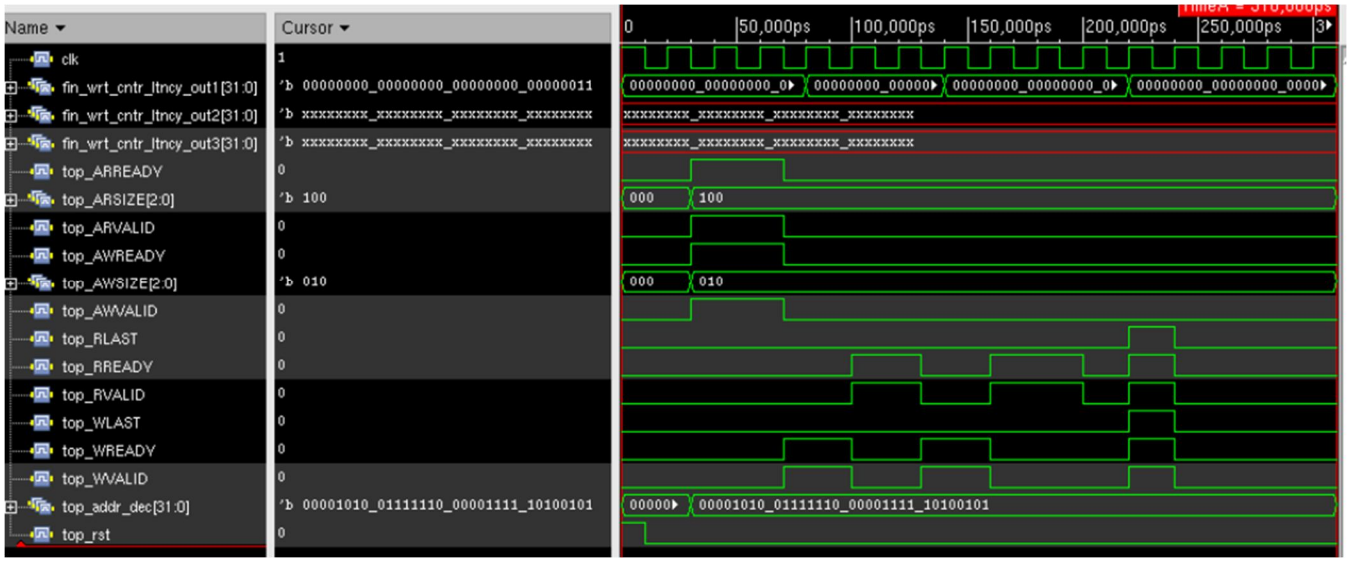


Fig. 17 Write latency count

TABLE IX
UTILIZATION ANALYSIS

Logic Utilization	Master Block		Slave Block	
	Used	Utilization	Used	Utilization
Number of slices	40	4%	54	4%
Number of slice flipflops	70	1%	80	1%
Number of 4 INPUT LUT's	150	3.50%	109	2.50%
Number of bonded IOBs	148	90%	79	50%
Number of BUFG/BUFGCTRLs	2	5%	3	5%
Maximum Operating Frequency	194.62MHZ		180.349MHZ	

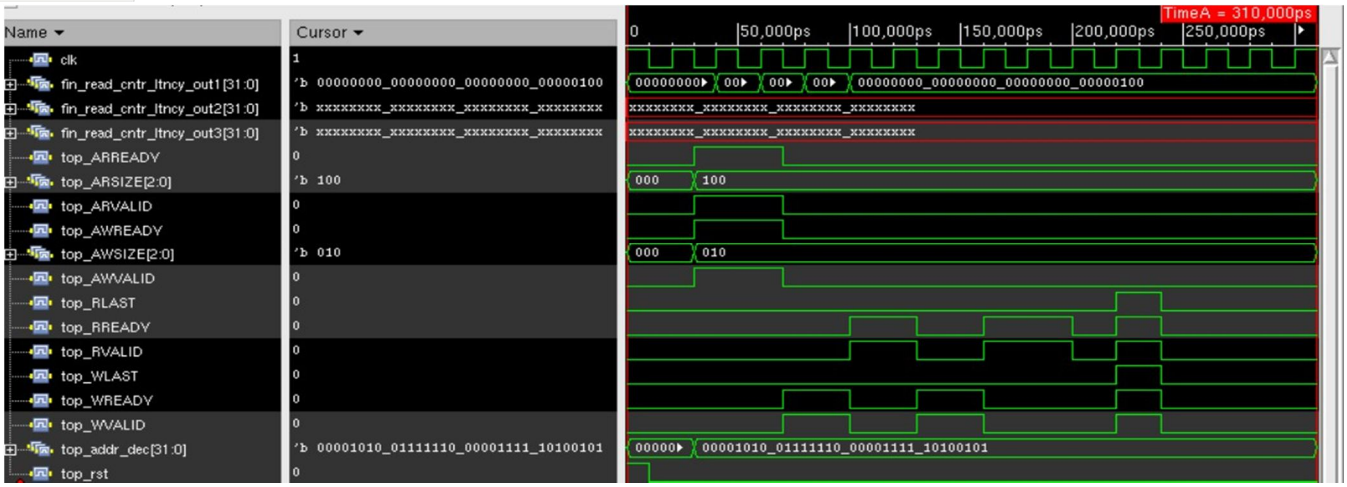


Fig. 18 Read latency count

Burst types, the extreme burst length remains 16 beats. The prerequisites for the AMBA AXI system must befulfilled. The future work for the AXI (Advanced eXtensible Interface) protocol may involve several areas of improvement and innovation to address emerging challenges and advance the capabilities of the protocol. Here are some potential avenues for future work:

a) Enhanced Performance

Research and develop techniques to improve the performance of AXI-based systems, including reducing latency, increas- ing throughput, and optimizing resource utilization. This could involve innovations in interconnect architectures, arbitration algorithms, and transaction pipelining.

b) Support for Advanced Features

Extend the AXI protocol to support new features and functionalities required for emerging applications and technologies. This could include adding support for features such as cache coherency, out-of-order transactions, quality of service (QoS) guarantees, and power management.

c) Integration with Emerging Standards

Explore opportunities to integrate the AXI protocol with other emerging standards and protocols in the field of system- on-chip (SoC) design and communication. This could involve interoperability with standards such as CCIX (Cache Coherent Interconnect for Accelerators) or integration with emerging memory protocols like HBM (High Bandwidth Memory).

d) Security Enhancements

Enhance the security features of the AXI protocol to address growing concerns about system security and integrity. This could involve incorporating mechanisms for secure booting, hardware-based encryption and authentication, and protection against side- channel attacks.

e) Cross-Protocol Compatibility

Investigate ways to improve compatibility and interoperability between different versions of the AXI protocol as well as with other communication protocols commonly used in SoC designs, such as AMBA AHB (Advanced High-performance Bus) or ACE (AXI Coherency Extensions).

f) Standardization and Documentation

Continue to refine and expand the AXI protocol specification, providing clear and comprehensive documentation to facilitate its implementation and adoption by semiconductor companies, IP vendors, and system designers. This includes addressing ambiguities, clarifying requirements, and updating the specification to reflect industry best practices.

g) Power Efficiency

Research methods to optimize power consumption in AXI-based systems, particularly in battery-powered devices and energy-constrained environments. This could involve techniques such as dynamic voltage and frequency scaling (DVFS), clock gating, and power-aware routing algorithms.

h) Machine Learning and AI Acceleration

Investigate how the AXI protocol can be optimized for the efficient implementation of machine learning (ML) and artifi- cial intelligence (AI) algorithms in hardware accelerators. This could involve customizing the protocol to better support data movement, memory access patterns, and parallel computation.

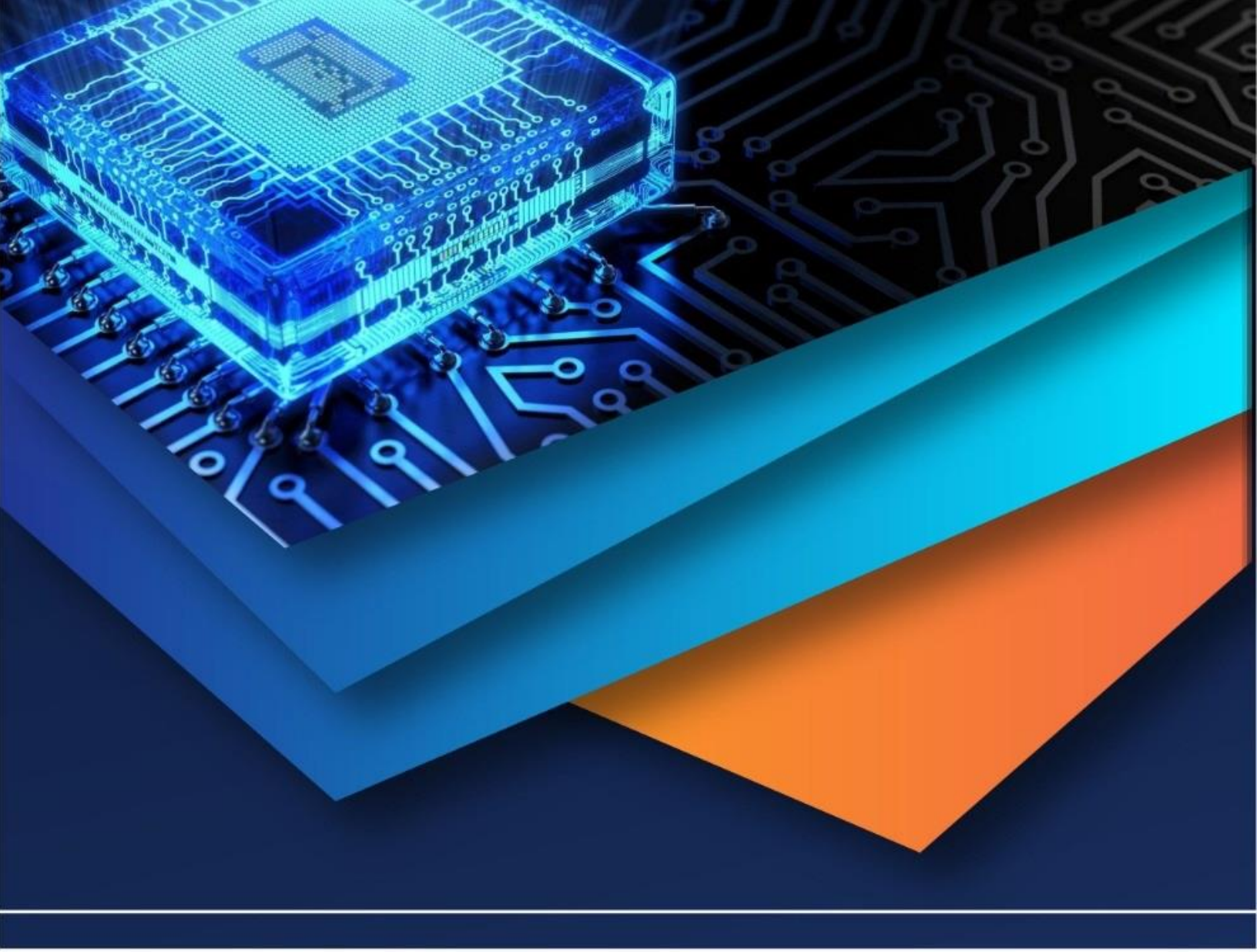
i) Standard Compliance and Certification

Strengthen efforts to ensure compliance with AXI protocol standards across the industry, including establishing certification programs and compliance testing suites. This helps to promote interoperability, reliability, and consistency in AXI-based designs across different vendors and platforms.

REFERENCES

[1] Schmoldt, H. F. Benthe, and G. Haberland, "Digitoxin metabolism by rat liver microsomes," Biochem. Pharmacol, vol. 24, no. 17, pp. 1639–1641, 1975.
[2] J. K. Glenn and J. Goldman, "Task delegation to physician extenders–some comparisons," Am. J. Public Health, vol. 66, no. 1, pp. 64–66, 1976.
[3] R. P. Patil and P. V. Sangamkar, "A Review of System-On-Chip Bus Protocols," Int. J. Adv. Res. Electr. Electron. Instrum. Eng, vol. 04, no. 01, pp. 271–281, 2015.
[4] H. G. Verification, Amba, Bus, Implementing, Wrap, Using, and Verilog, Int. J. Res. Eng. Technol, vol. 05, no. 03, pp. 201–206, 2016.
[5] V. I. Gavrilov, Acta Virol, vol. 19, no. 6, pp. 510–510, 1975.
[6] H. G. Verification, Amba, Bus, Implementing, Wrap, Using, and Verilog, Int. J. Res. Eng. Technol, vol. 05, no. 03, pp. 201–206, 2016.
[7] A. R. Babu, P. Anand, Y. Kim, and S. Jadhav, "System Verilog versus UVM-based Verification of AXI4-Lite Arbitration," in SoutheastCon 2023. IEEE, 2023, pp. 350–357.
[8] D. C. K. Kho and K. Munusamy, "Transaction-based SoC design techniques for AMBA AXI4 bus interconnects using VHDL," in 2014 11th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON). IEEE, 2014, pp. 1–6.
[9] R. H. H. S. S. Prasad and C. S. Rani, "Development of VIP for AMBA AXI-4.0 Protocol," Indian J. Sci. Technol, vol. 9, no. 48, 2016.
[10] Y. W. Chow, R. Pietranico, and A. Mukerji, "Studies of oxygen binding energy to hemoglobin molecule," Biochem. Biophys. Res. Commun, vol. 66, no. 4, pp. 1424–1431, 1975.
[11] W. Silen, T. E. Machen, and J. G. Forte, "Acid-base balance in amphibian gastric mucosa," Am. J. Physiol, vol. 229, no. 3, pp. 721–730, 1975.
[12] B. I. Korelitz and S. C. Sommers, "Responses to drug therapy in ulcerative colitis. Evaluation by rectal biopsy and histopathological changes," Am. J. Gastroenterol, vol. 64, no. 5, pp. 365–370, 1975.
[13] E. A. Suvorova, "The approach to design of dynamically reconfigurable arbitration units in embedded systems," Radio Ind. Russ, vol. 29, no. 3, pp. 55–67, 2019.
[14] W. H. Parry, F. Martorano, and E. K. Cotton, "Management of life-threatening asthma with intravenous isoproterenol infusions," Am. J. Dis. Child, vol. 130, no. 1, pp. 39–42, 1960.
[15] R. F. Wachter, G. P. Briggs, and C. E. Pedersen, "Precipitation of phase I antigen of Coxiella burnetii by sodium sulfite," Acta Virol, vol. 19, no. 6, pp. 500–500, 1975.
[16] W. Silen, T. E. Machen, and J. G. Forte, "Acid-base balance in amphibian gastric mucosa," Am. J. Physiol, vol. 229, no. 3, pp. 721–730, 1975.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)