iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 11 **Issue:** VIII **Month of publication:** Aug 2023

**DOI:** https://doi.org/10.22214/ijraset.2023.55366

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 11 Issue VIII Aug 2023- Available at www.ijraset.com*

# High Secure Data Transactions in AXI System Bus using Firewall Protection

Pruthvi D[1], Dr. Srividya P[2]
[1]*Mtech Student,* [2]*Assosiate Professor, Department of ECE, RVCE, Bangalore*

***Abstract:*** ***An open-standard, on the chip interconnection protocol called the Advance Microcontroller Bus Architecture (AMBA) is used to link and manage functional blocks in systems on chip (SoC) architectures. An interface definition called Advanced eXtensible Interface (AXI) specifies the IP block interface rather than the connection itself. The point-to-point connections between a supervisor as well as subordinate are defined by the AXI protocol in terms of their signals and time. The existing AXI protocol only has the response mechanism for the slave error and the valid transactions. In this work the high secure data transactions in AXI system bus is introduced using the firewall protection where a certain range of address are protected to access from other master peripherials.***

***This is indicated by a error response from the rresp and bresp signals rather than accessing the address to be accessed for the secure transaction. The entire AMBA AXI is an Finite State Machine (FSM), which can be obtained by composing the FSMs associated with each component. The main intent of the FSM is to validate the flow of transactions entering and leaving a subsystem has to be secured and stable and has to go through the various states of the state machine. The design of the AMBA AXI protocol is done using System Verilog and the formal checks in terms of System Verilog testbench are used to check the internal signals which is aiding the read write transactions of the memory which is acting as the slave for the peripheral. The AMBA AXI System bus is verified using the testbench created in SystemVerilog (SV) Universal Verification Methodology (UVM).***

***The design is also tested for the connectivity of the internals signals of the slave with the master with the help System Verilog code where only the design and the SV files are needed for the verification. The read and write transactions for the various burst transactions in terms of Fixed, Increment(INCR) and Wrap transactions are tested to check the ability of the memory to respond to vari- ous signals given by the master. The complete design is verified using the Xilinx Vivado 2020.2 Design Suite and make sure that the design is free from bugs.***

***Keywords: AXI, UVM, Verification, Xilinx Vivado, System On chip (SoC)***

## I. INTRODUCTION

Today's System on Chip designs employ numerous Intellectual Property (IP) cores and are becoming exponentially more complex. Advanced Extensible Interface (AXI) interconnection IP, one of the most used interconnect IPs, is one example of such an IP. Reusable components are necessary to reduce the time as well as difficulty associated with verification of designs as systems get bigger and quicker.

Due to the increasing complexity of system-on-chip (SOC) designs, most common protocols and interface IP allow verification professionals to evaluate basic functionalities; nonetheless, this is starting to pose a serious problem.

Specialised verification approaches, including as the Universal Verification Methodology (UVM), are employed to slow down the verification process. Because of the specific coding style used by these approaches, the code may be used in any testing environment based on the identical methodology. The five channels of the AXI memory transactions—write data, write address, write response, read data and read address—are all verified as part of the process of verifying memory transactions. The test bench is designed to validate the memory interactions within the AXI protocol, including how data is moved between locations throughout read and write phases at the identical and distinct addresses. A typical system, as seen in Figure 1, is made up of multiple Manager and Subordinate components that are interconnected in some way. For the connections between devices, the AXI protocol offers a single interface description.

*1)* Manager and the interconnect.
*2)* Subordinate and the interconnect.
*3)* Manager and a Subordinate.

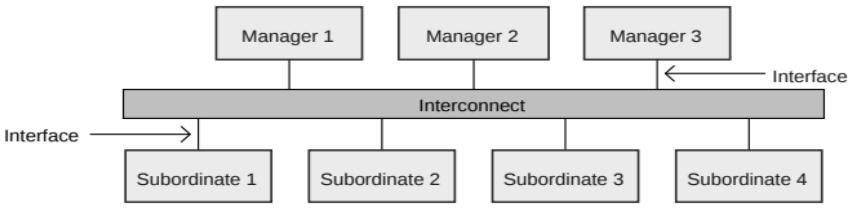


Figure 1: Interface and Interconnect[12]

Numerous distinct connectivity architectures are supported by the aforementioned interface specification. The device with the interconnection is the same as a different device having identical Manager as well as Subordinate connectors to which the actual Manager along with Subordinate systems can be attached. One of three connectivity topologies is often used by systems:

*a)* Channels for data and request sharing.
*b)* Multi-layer, with numerous request and data channels;
*c)* shared request channel;
*d)* multiple data channels.

The request for communication channels bandwidth need is often much lower that the information channel frequency required for the majority of systems. By utilising a common request connection with numerous data lanes that allow concurrent information transfers, these types of systems can strike a fair compromise among system efficiency and interconnection complication.

## II. LITERATURE SURVEY

The Paper [1] depicts about the process for designing and verifying SoCs using Instruction Level Abstractions (ILAs), which have been synthesised. The ILA is a model of SoC technology that simulates firmware-visible state modifications at the instruction-level level. The ILA provides comprehensive verification for firmware working with hardware acceleration devices and is akin to an instruction-set architectural concept for programmable processors for hardware injectors. A modest SOC design based on the 8051 microcontroller was used by the author to assess the technique, and this revealed 15 faults.

An algebraic method for functionally verifying gate-level, arithmetic devices is presented in Paper[2]. Its foundation is the extraction of a distinct bit-level polynomial equation computed directly by the circuit using its gate-level implementation. The technique may be applied to either derive a mathematical function that was used by the device or to check the mathematical functions calculated with that circuitry against its established specifications.

Checking integer multipliers with various architecture is the main topic of Paper 3. The author suggested a method for producing BDDs with a maximum of 1024 bits directly, without going through any circuits. When creating BDDs, the author uses the extra variable approach to limit the complexity of the BDD length to a third order quadratic. A technique for equivalence testing and validating up to 32-bit optimised multipliers has been presented for evaluating optimised multiplication.

In Paper[4], methodologies are presented that are then applied to two instances of NoC components, offering positive as well as negative testing scenarios: A fundamental router as well as a Daniel router are examples. While Daniel router represents a complicated programmable open-source scenario, the Base router provides a straightforward case study to demonstrate suggested approaches. With a few settings and implemented algorithms, Daniel router offers the possibility to alter router design. This paper's integration of a complete UVM environment with several verification methods is its second major contribution. Utilising a repeatable and modular UVM framework and elements for NoC, target techniques involve error insertion and identification. The network response is examined based on the technique and error kind.

Any AXI device may be tested using the structure of the verification framework presented in Paper[5] for the AMBA AXI interface. To verify whether the verification method includes all potential circumstances, a functional completeness model was created. Every testcase generates a coverage report that can be utilised afterwards to evaluate the testcase's efficacy. With the use of both random and guided test cases, complete coverage can be attained.

### *A. AXI Write and Read Channel Architecture*

With master and slave devices used in the AXI protocols are depicted in Figure 1. Five independent channels for reading and writing are available; they are read data, read address along with write response, write data, and write address. Figure 1 depicts the write channel design. The transmission of information from a master device to a slave device is done through a write data channel. The write response signal is used by the slave to signify to the master that data transmission is complete.

This mutual handshake technique is the foundation of the AXI interface. In order to let the slave know that an address as well as information regarding the channel is genuine, the master transmits an appropriate signal. Slave notifies master that it is prepared to take the data by sending a ready signal. By this manner, a handshake among the master as well as slave takes place before any information can transfer over the channel. In order for a write operation to take place, the master must first communicate the necessary address controls and data over the write address channel. Write data channels, which can be 32 or 64 bitwide, are used to transfer write data from the master to the slave. Slave delivers notification over the write response channel once the transaction is complete. Figure 2 depicts the write channel design. For reading data, the read address line includes address and control information.

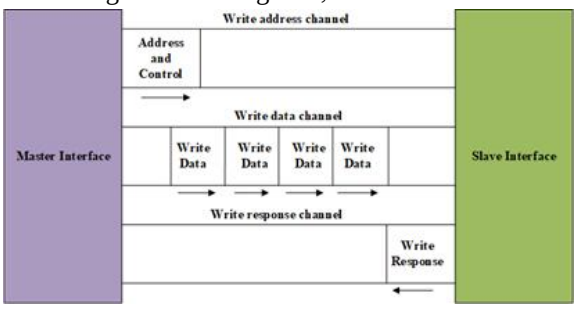



Fig. 2. Write Channel Architecture

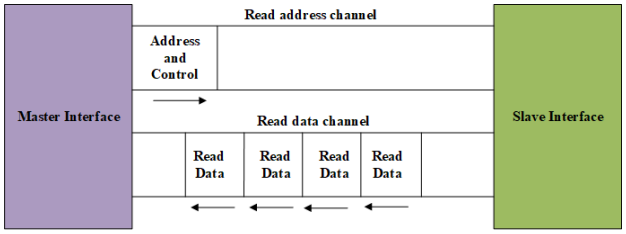



Fig. 3. Read Channel Architecture

*1) AXI Write and Read Channel State Diagram*

The Figure 4 displays the AXI write channel's state diagram. Three cycles of the clock are required for the writing process. The initial state is IDLE if the ARESETn signals remains low, at which point the awready signal is raised to indicate that the slave is prepared to take the address over the write channel. When the master asserts awvalid, signaling that the address over the associated channel is valid, it will move into the DATA phase. In a result, the master specifies the initial address at which it wants to write the information. Writing data there is the subsequent action. A handshake takes place between the master and slave when the master asserts wvalid, signaling that the data on the channel is valid. By using the address listed in, write data is transferred from the master and recorded in the slave. The answer will be sent in the third cycle after the data has been sent to the slave. When sending a response, the slave is going to assert the bvalid signal. Eventually another data transmission occurs, and the cycle continues.

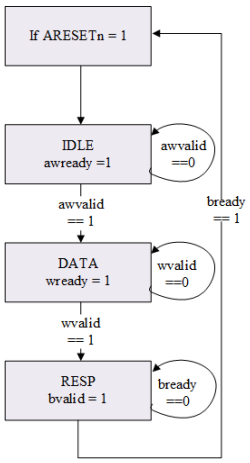



Fig. 4. Write Channel State Diagram

Figure 5 displays the AXI read channel's state diagram. Performing a read requires two clock cycles. The slave is prepared to take the read address in the first state, IDLE, when the aready signal is emitted. A master device will transmit an address over and araddr channel and emit an arvalid signal. The real data transfer occurs in the next state, DATA. Slave asserts the rvalid signal to indicate that the data on the read channel is acceptable. The main device will issue the rready signal, and the master's device will read the data.. Figure 3 and Figure 4 depicts about the methodology where the initial analyses on the protocol architecture for the memory is taken place which is then followed by the finite state machine (FSM) designs to meet the Advanced Microcontroller Bus Architecture AXI protocol requirements. The complete design is completed using the SystemVerilog and then followed by integrating the memory module to the FSM's. Once the design is ready, the testbench is developed using SystemVerilog UVM which is used for checking the design intent.

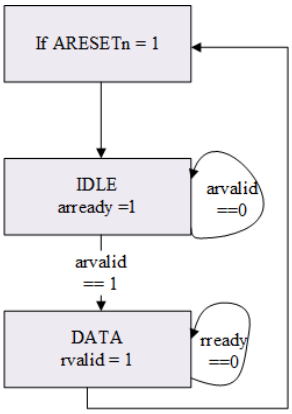



Fig. 5. Read Channel State Diagram

## III.VERIFICATION FLOW

A repeatable verification environment is made using the System Verilog UVM technique. The key benefit of UVM is that we can construct every element by just extending basic classes, which is how System Verilog employs OOPs ideas to make component development simpler and faster. Therefore, components for verification can be reused in other applications. The essential elements of the UVM-based testbench environment are listed below:

1) *Monitor:* Pin-wiggles generated by the DUT are received by this part, which then turns these into transactions and sends them to other associated components like the scoreboard as well as coverage collection. It will disseminate transactions to all of the connected subscribers once it has received them from the DUT.
2) *Sequencer:* All data fields that are controlled by the driver as well as monitored with the monitor are contained in the sequence-item. The information in these fields are provided to the driver through the sequencer in an arbitrary sequence. Between sequences and the driver, the sequencer synchronizes the flow of request and answer sequence pieces. Based on written sequences, this sequencer will produce and deliver AXI transaction messages to the driver.
3) *Driver:* Transactions are transformed into pin wiggles by the driver before being sent to the DUT. Driver awaits the sequencer's transaction. Driver then drives the AXI DUT using the AXI packets it has received from the sequencer. Through the TLM interface, the sequencer as well as driver are coupled. Once the processor completes the present transaction, it will provide the sequencer a response confirming the transaction was successful, at which point the sequencer can begin driving the subsequent transaction.
4) *Agent:* The sequencer, the driver as well as monitor are all included within the agent. Both active and passive agents are possible; active agents have each of the three components as well as transmit stimulation to the DUT, whereas passive agents simply have the monitor. In an agent's connection phase, the sequencer and driver are connected.
5) *Coverage Collector:* The subscriber component that connects to the monitor is this one. It gets a transaction from the monitor and hits the cover points in line with that. In this component, different covering areas are created for each potential address and data value.
6) *Environment:* The agent, scoreboard, and coverage collector are all instantiated there. In the complicated verification environment, it may contain more than one agent. During the connect step, the scoreboard as well as coverage collectors are linked to the monitor.

7) *Scoreboard:* The AXI transactions are collected by this element from the monitor by using the UVM analysis imp's write method, and they are then placed in a queue. To ensure that the design is accurate, it compares the DUT output with the golden output. Shift operation is used to create a golden result according to the signals that are fed to the DUT, and the result is then compared to what is received rdata via the AXI slave device.
8) *Test and Sequences:* This creates a sequence and an environment. The execution phase of the test component is where the sequence is initiated. There are several AXI read and write transactions in the sequence.

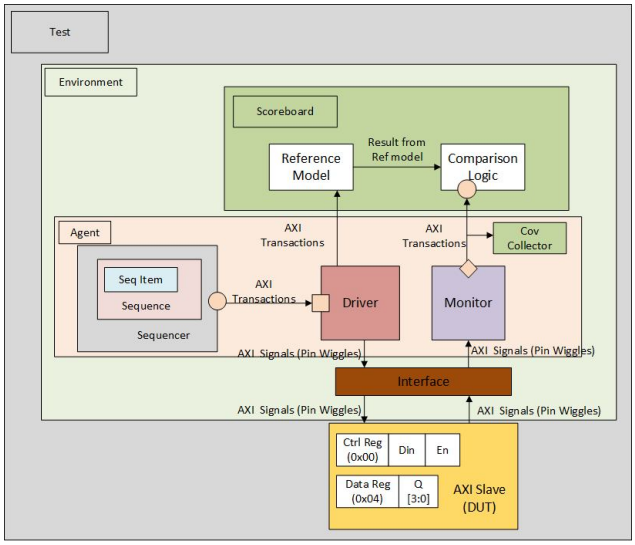


Fig. 6. AXI Testbench Structure

Figure 6 depicts the UVM-based AXI testbench topology. The primary goal of DUT verification is to write data into the device's internal shift register and then retrieve it back. Then it can be confirmed that both reading and writing operations occur correctly over the AXI interface through comparison of the data in the scoreboard. The control register along with the data register are the two registers on the DUT. One bit of data input from the shift register is stored in the control register together with the enable signal. When the En signal is strong, the shift register moves data to the left. Data output from a shift register is stored in a data register. By asserting the necessary AXI write channel signals, the UVM sequence creates an order to write a four-bit information to a shift register. These steps are provided to driver via sequencer. The data is entered into the DUT controller registers and subsequently the shift register when a driver transmits this sequence to the DUT. A different sequence is triggered to read information from the data register once writing is completed in four clock cycles. As a result, shifted information is written into the DUT's data register before being read. The address of the data register and the accompanying AXI reading channel signals are provided by the read sequence. The monitor transforms this DUT answer into transaction and monitors it. Data from the DUT is then compared to data that is put in the scoreboard. If the two data are identical, the AXI interface has successfully communicated the data.

## IV. SIMULATION RESULTS

This section provide details related to results of the design implementation using Xilinx Vivado 2020.2 Design suite. The read and write transfer simulation uses a 100MHz clock frequency. To acquire the results for the various AXI write and read channels, including the channel for writing address, write data, writer response, read address, and read data channels, the System Verilog is components have been simulated with the testbench. The different burst type of transactions such as Fixed, INCR and the WRAP transactions are shown with the help of various read and write signals. Finally the type of the error response provided for the firewall protected address are shown in this chapter.

### *A. AXI write Channels*

AXI Write consists of three channels namely Write data channel, write address channel and the write response channel. AXI Write signals which has a clock of 100MHz. There is a active low reset signal is used here in order to make sure that if the resetn signal is high at the positive edge of the clock cycle then the channels are in the active state. If the resetn signal is low at the positive clock edge then the channels go to the reset state which is the inactive state.

*1) Write Address Channel*

Figure 7 shows the AXI Write address channel which consists of the addr wrapwr, awvalid, awready, awid, awlen, awsize, awaddr, awburst. addr wrapwr indicates the next address in the wrap write. awvalid indicates the valid address to be written and it also indicates that the master is sending new valid address. awready indicates that the slave is ready to accept the request. awid indicates the unique ID for each transaction. awlen indicates the burst length for the AXI transfer ie., it can range from 1 to 256. awsize indicates the unique transaction size ie., 1, 2, 4, 8, 16,...128 bytes but in this work 4 bytes is used ie., 32 bits of address. awaddr indicates the write address of transaction. awburst indicates the burst type or the type of burst ode used in the transaction i.e., Fixed, INCR or WRAP transaction. In the Figure 7 awvalid is high indicating the master is sending the new address and the written address is valid. awready is high in between the transaction indicating at this slave is ready for the transaction to happen. awid indicating the unique ID of 0x9. awlen taken here is 7 sets of transfer. awsize is 2 indicating that the 4 bytes (32bits) of the address is written. awaddr is indicating the different set of address in the transaction, awburst is 0x2 indicating the WRAP type of the burst transfer.

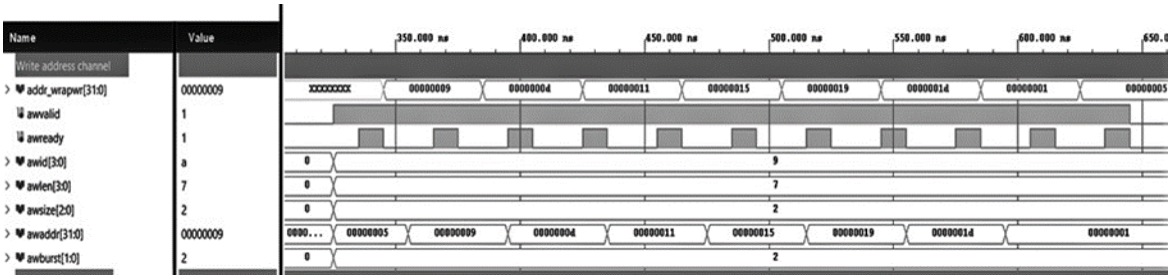



Figure 7 AXI Write Address Channel signals

*2) Write Data Channel*

Figure 8 shows the AXI Write Data channel which consists of the wvalid, wready, wid, wdata, wstrb and wlast. wvalid indicates the valid data is been written and it also indicates that the master is sending new valid data. wready indicates that the slave is ready to accept the new data. wid indicates the unique ID for each transaction. wdata indicates the data that is being written during the AXI write transaction. wstrb indicates the lane having the valid data. Rest of the lanes which does not have the valid data is not used for the write transaction. wlast signal indicates the last transfer in the write burst.

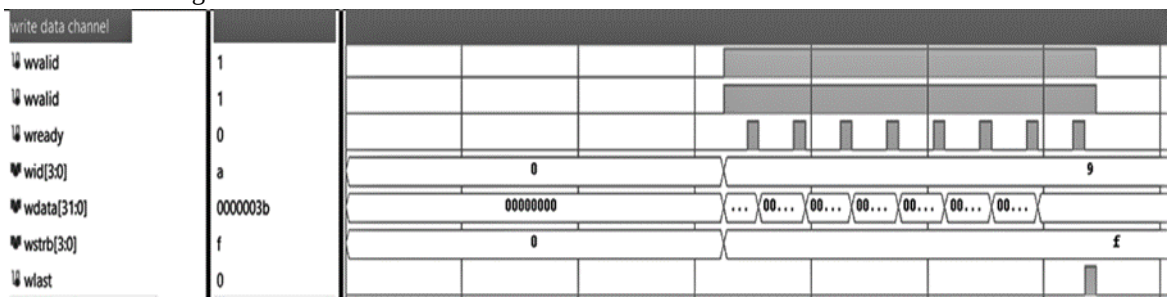



Figure 8 AXI Write Data Channel signals

In the Figure 8 wvalid is high indicating the master is sending the new data and the written data is valid. wready is high in between the transaction indicating that this slave is ready for the write data transaction to happen . wid indicating the unique ID of 0x9. wdata holds the data that is being written during the AXI Write transaction. wstrb is 0xF indicating high for all the 4 lanes meaning that all the 4 lanes are having valid data for the transaction. wlast signal high indicates the last byte of data is being written and after this the write data transaction is completed.

*3) Write Response Channel*

Figure 9 shows the AXI Write Response channel which consists of 4 signals namely bvalid, bready, bid, bresp. bvalid indicates that the slave has the valid response. bready indicates that the master is ready to accept the response from the slave. bid indicates the unique ID for each transaction. bresp indicates the response for the invalid transaction.

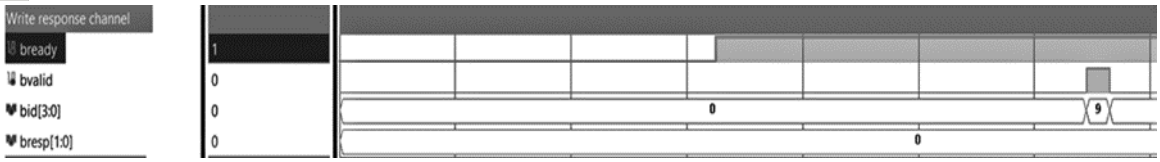



Figure 9 AXI Write Response Channel signals

In the Figure 9 bvalid is high indicating the slave has the valid response. bready is high in between the transaction indicating that this master is ready to accept the response which is sent from the slave. wid indicating the unique ID of 0x9. Since this is a valid write transaction happening the bresp is giving the OK response of 0x0.

*B. AXI Read Channels*

AXI Read consists of two channels namely Read Address Channel and the Read Data Channel. The AXI Read signals which has a clock of 100MHz. There is a active low reset signal is used here in order to make sure that if the resetn signal is high at the positive edge of the clock cycle then the channels are in the active state. If the resetn signal is low at the positive clock edge then the channels go to the reset state which is the inactive state. Here the Read address channel consists of the arvalid, arready, arid,arlen, arsize, araddr, arburst. The read data channel consists of rresp, rvalid, rready, rid, rdata, rstrb and rlast. All these signals operation will be shown in further sections.

*1) Read Address Channel*

Figure 10 shows the AXI Read Address channel which consists of the arvalid, arready, arid, arlen, arsize, araddr, arburst. arvalid indicates the valid address to be read and it also indicates that the master is sending new valid address. arready indicates that the slave is ready to accept the request. arid indicates the unique ID for each transaction. arlen indicates the burst length for the AXI transfer ie., it can range from 1 to 256. arsize indicates the unique transaction size ie., 1, 2, 4, 8, 16,...128 bytes but in this work 4 bytes is used ie., 32 bits of address. araddr indicates the read address of transaction. arburst indicates the burst type or the type of burst ode used in the transaction i.e., Fixed, INCR or WRAP transaction.

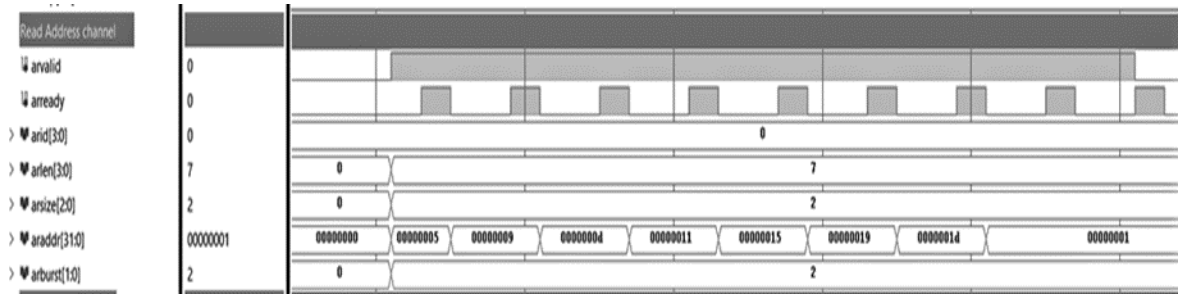



Figure 10 AXI Read Address Channel signals

In the Figure 10 arvalid is high indicating the master is sending the new address and the read address is valid. arready is high in between the transaction indicating at this slave is ready for the transaction to happen. arid indicating the unique ID of 0x0. arlen taken here is 7 sets of transfer. arsize is 2 indicating that the 4 bytes (32bits) of the address is read. araddr is indicating the different set of address in the transaction, arburst is 0x2 indicating the WRAP type of the burst transfer.

*2) Read Data Channel*

Figure 11 shows the AXI Read Data channel which consists of the rresp, rvalid, rready, rid, rdata, rstrb and rlast. rresp indicates the response to the invalid read trans- action. rvalid indicates the valid data is been read and it also indicates that the master is sending new valid data. Ready indicates that the slave is ready to accept the new data. rid indicates the unique ID for each transaction. rdata indicates the data that is being read during the AXI read transaction. rstrb indicates the lane having the valid data. Rest of the lanes which does not have the valid data is not used for the read transaction. rlast signal indicates the last transfer in the read burst.

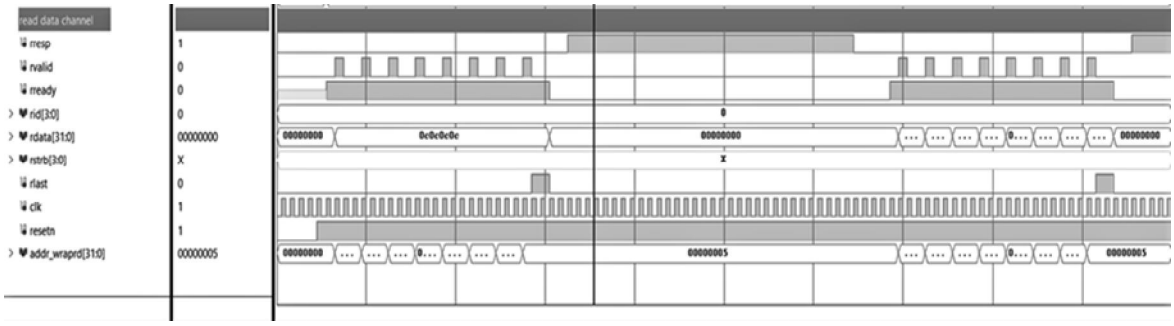



Figure 11 AXI Read Data Channel signals

In the Figure 11 rvalid is high indicating the master is sending the new data and the read data is valid. rready is high in between the transaction indicating that this slave is ready for the read data transaction to happen. rid indicating the unique ID of 0x0. rdata holds the data that is being read during the AXI read transaction. rstrb is 0xF indicating high for all the 4 lanes meaning that all the 4 lanes are having valid data for the transaction. rlast signal high indicates the last byte of data is being read and after this the read data transaction is completed.

*C. Error response for Firewall Protected Address*

Response messaging is offered by AXI for both writing and reading transactions. The response data provided by the subordinate is sent as a signal via the read information channel using rresp for read transactions. Bresp is used to convey the response information for write transactions on the written response channel. Both rresp hence bresp are made up of just two bits, and when these signals are encoded, the following four answers can be transmitted:

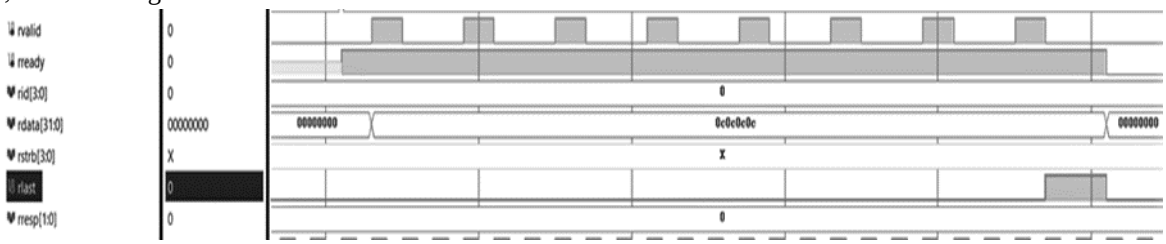



Figure 12 AXI OKAY Response for valid transaction

The OKAY answer, regular access success, or restricted access failure is indicated by the rresp = 00. The response most frequently used in transactions is OKAY. A successful normal access is signaled by the word OKAY. The failure of an exclusive access can also be inferred from this answer. When many managers may access the same subordinate simultaneously but not the same memory range, this is referred to as an exclusive access. The AXI database OKAY Response for a legitimate transaction is shown in Figure 12.rresp = 01 denotes EXOKAY, or exclusive access is permitted. EXOKAY denotes the accomplishment of precisely the ability to read or write component of an exclusive access.

When rresp = 10 implies subordinate mistake, or SLVERR. When information has successfully reached the subordinate but the subordinate needs to report a mistake condition to their original captain, SLVERR is employed. This is a sign of a failed transfer.

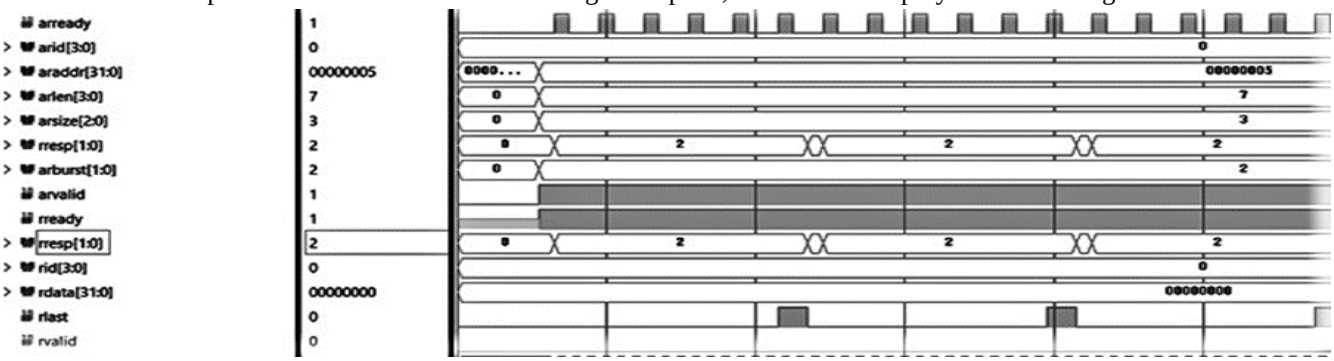



Figure 13 AXI Error Response for Slave Error

For instance, when a prohibited transfer size is attempted or when writing access to a read-only location is attempted. The the AXI database Error Response with Slave Error regarding addresses 0x5 is displayed in Figure 13.

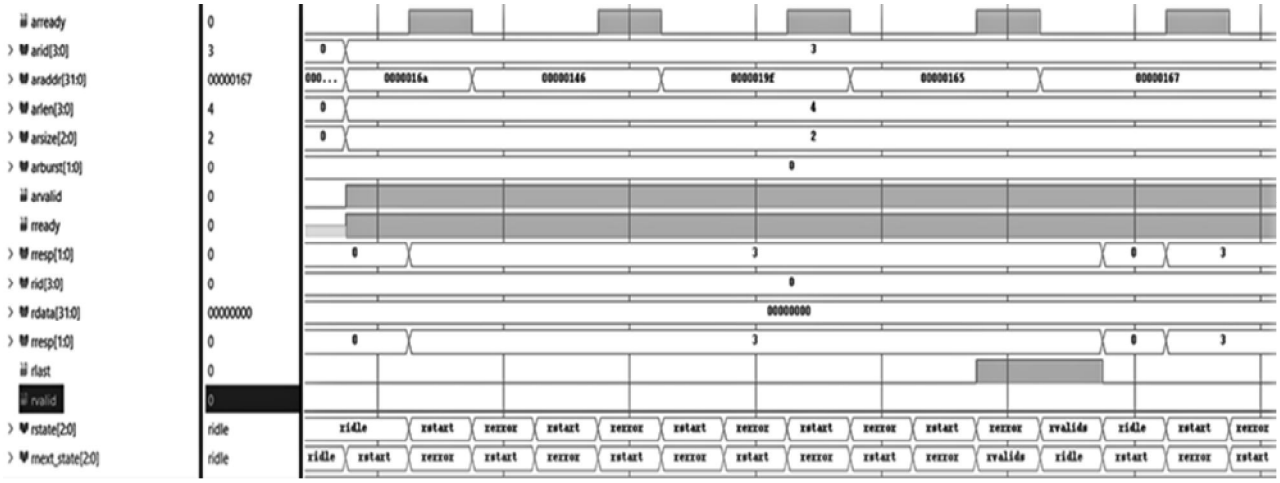



Figure 14 AXI Error Response with States for Firewall Protection

The rresp = 11 indicates Firewall Protection error or the Invalid Transaction. When there has been no supervisor at the transaction's address, an interconnect component will frequently create this message to let users know. Figure 14 shows the AXI Error Response with States for Firewall Protection for the invalid addresses 0x16a, 0x146, 0x19f, 0x165. Figure 15 shows the AXI Error Response for Address Protected under the Firewall Protection for the invalid addresses 0x16a, 0x146, 0x19f, 0x165.


Figure 15 AXI Error Response for Address Protected under the Firewall Protection

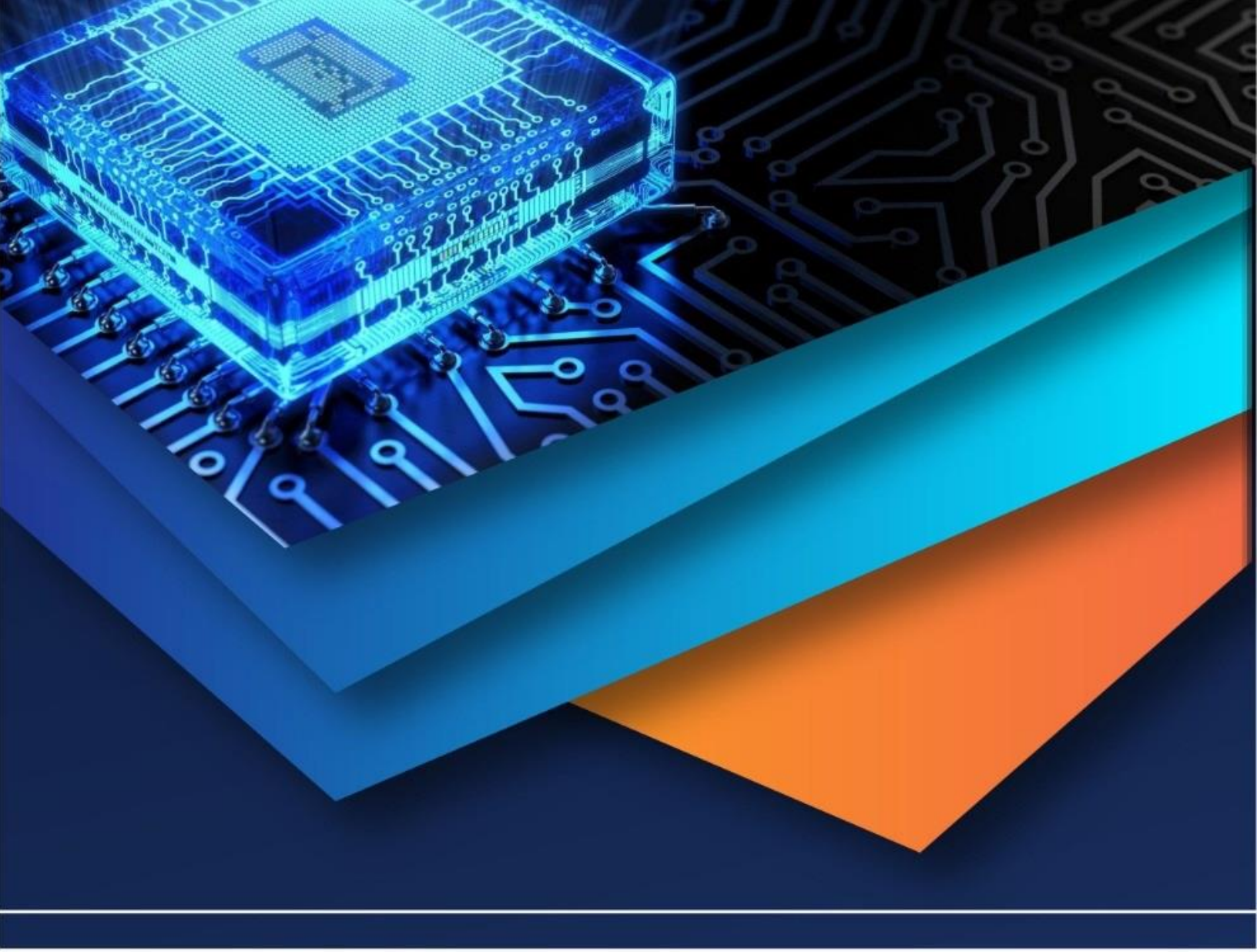
A manager can inform a subordinate whether bytes on the data bus are needed using a write data strobe signal. Write storage strobes are helpful for fast transfer of sparse data matrices across cache accesses. In addition to writing data strobes, unaligned start addresses can also be used to optimize data transfers. Each bytes on the data bus for the writer channel encounters one strobe bit. The WSTRB signal is composed of these bits. A manager must make sure that only byte lanes with valid data have their write strobes set to 1.

## V. CONCLUSIONS

This paper is mainly focused on-designing the Advanced Micro-controller Bus Architecture (AXI) protocol using System Verilog for the high secure read and write transactions. These read and write transactions for the three phases of the channel (Address, Write, and Response) are designed and verified with the supporting signals sent by the master to the slave to perform read and write operations to the memory with various burst transactions that the protocol supports. The Wrap, INCR and Fixed bursts have been tested using the Xilinx Vivado 2020.2 design suite for all the read and write transactions. The present AXI protocol supports the Fixed, INCR and WRAP type of bursts where most of the lanes are left empty while the transactions occur. This leads to the wastage of the memory lanes and more and more memory space would be required for the transactions to be happened for SoCs. In order to reduce this memory wastage the empty lanes can be made used by reusing each empty lane for the read and write transactions. In manual verification, an actual tester generally starts the programmed, selects a testcase, selects on widgets in accordance with the sequence specified by the testcase, and continues the process until the testing requirements is met. It becomes difficult for the tester to discover all testcases, prioritize and choose testcases based on certain criteria, perform each of the chosen examples any error or omission, and check the outcome of performing each testcase when this approach is used to test the application. These issues can be solved by implementing automated verification, and the GUI can be checked automatically using the Capture/Replay approach. In this technique, a human tester runs a series of test cases on the GUI, which the capture tool watches and records. The replay tool runs identical testcases using the GUI afterwards to look for any regressions.

## REFERENCES

[1] P. Subramanyan, B.-Y. Huang, Y. Vizel, A. Gupta, and S. Malik, "Template-based parameterized synthesis of uniform instruction-level abstractions for soc verifica- tion," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 8, pp. 1692–1705, 2018.
[2] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal verification of arithmetic circuits by function extraction," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 12, pp. 2131–2142, 2016.
[3] J. Kumar, Y. Miyasaka, A. Srivastava, and M. Fujita, "Formal verification of in- teger multiplier circuits using binary decision diagrams," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 42, no. 4, pp. 1365– 1378, 2023.
[4] S. El-Ashry, M. Khamis, H. Ibrahim, A. Shalaby, M. Abdelsalam, and M. W. El- Kharashi, "On error injection for noc platforms: A uvm-based generic verification environment," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 5, pp. 1137–1150, 2020.
[5] V. Melikyan, S. Harutyunyan, A. Kirakosyan, and T. Kaplanyan, "Uvm verification ip for axi," in 2021 IEEE East-West Design Test Symposium (EWDTS), 2021, pp. 1–4.
[6] N. K. P, D. V, A. M, S. K. R, and E. S, "Design and verification of amba axi3 pro- tocol for high speed communication," in 2022 Smart Technologies, Communication and Robotics (STCR), 2022, pp. 1–5.
[7] P. Dwivedi, N. Mishra, and A. Singh-Rajput, "Assertion functional coverage driven verification of amba advance peripheral bus protocol using system verilog," in 2021 International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT), 2021, pp. 1–6.
[8] T. Strauch, "Dynamic inside-out verification using inverse transactions in tlm," in 2018 Forum on Specification Design Languages (FDL), 2018, pp. 5–16.
[9] H. H. Ardakani, A. M. Gharehbaghi, and S. Hessabi, "A performance and functional assertion-based verification methodology at transaction-level," in 2007 Internatonal Conference on Microelectronics, 2007, pp. 133–136.
[10] M. Siegel, "Achieving earlier verification closure using advanced formal verifica- tion," in Formal Methods in Computer Aided Design, 2010, pp. 275–275.
[11] L. Duan, Y. Hu, H. Liu, W. Feng, and J. Gan, "An efficient formal verification method in i/o multiplexing module based on vc formal cc," in 2020 IEEE 3rd International Conference on Electronics and Communication Engineering (ICECE), 2020, pp. 112–116.
[12] M. W. Anwar, M. Rashid, F. Azam, A. Naeem, M. Kashif, and W. H. Butt, "A unified model-based framework for the simplified execution of static and dynamic assertion-based verification," IEEE Access, vol. 8, pp. 104 407–104 431, 2020.
[13] P. Gurha and R. R. Khandelwal, "Systemverilog assertion based verification of amba-ahb," in 2016 International Conference on Micro-Electronics and Telecom- munication Engineering (ICMETE), 2016, pp. 641–645.
[14] P. Bhamidipati, S. M. Achyutha, and R. Vemuri, "Security analysis of a system- on-chip using assertion-based verification," in 2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS), 2021, pp. 826–831.
[15] M Girish, G Gopakumar, and D. S. Divya, "Formal and simulation verification: Comparing and contrasting the two verification approaches," in 2021 2nd International Conference on Advances in Computing, Communication, Embedded and Secure Systems (ACCESS), 2021, pp. 41–44.
[16] S. Zhang and L. Cao, "Security and fault diagnosis-based assertion-based verifica- tion for fpga," in 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2019, pp. 478–481.
[17] A. Shkil, A. Miroshnyk, G. Kulak, and K. Pshenychnyi, "Assertion based design of timed finite state machine," in 2021 IEEE East-West Design Test Symposium (EWDTS), 2021, pp. 1–4.
[18] A. Hussien, S. Mohamed, M. Soliman, et al., "Development of a generic and a reconfigurable uvm-based verification environment for soc buses," in 2019 31st In- ternational Conference on Microelectronics (ICM), 2019, pp. 195–198.
[19] R. Claris´o, C. A. Gonz´alez, and J. Cabot, "Smart bound selection for the verification of uml/ocl class diagrams," IEEE Transactions on Software Engineering, vol. 45, no. 4, pp. 412–426, 2019.
[20] D. Wang, J. Yan, and Y. Qiao, "Research on chip verification technology based on uvm," in 2021 6th International Symposium on Computer and Information Pro- cessing Technology (ISCIPT), 2021, pp. 117–120.
[21] H. Xu, Z. Li, Z. Li, et al., "Reducing sram reading power with column data seg- ment and weights correlation enhancement for cnn processing," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 11, pp. 2237–2250, 2021.
[22] F. Plasencia-Balabarca, E. Mitacc-Meza, M. Raffo-Jara, and C. Silva-C´ardenas, "A flexible uvm-based verification framework reusable with avalon, ahb, axi and wish- bone bus interfaces for an aes encryption module," in 2019 IEEE Latin American Test Symposium (LATS), 2019, pp. 1–4.
[23] B. Vineeth and B. B. Tripura Sundari, "Uvm based testbench architecture for coverage driven functional verification of spi protocol," in 2018 International Con- ference on Advances in Computing, Communications and Informatics (ICACCI), 2018, pp. 307–310.
[24] D Yashas, P. S. Hari Babu, and N Shylashree, "Uvm-based logic verification of input output interface," in 2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology (RTEICT), 2019, pp. 420– 423.
[25] H. Sangani and U. Mehta, "Uvm based verification of read and write transactions in axi4-lite protocol," in 2022 IEEE Region 10 Symposium (TENSYMP), 2022, pp. 1–5.
[26] K. Mahmoud, R. Ahmed, K. Ayman, et al., "Towards a generic uvm," in 2022 IEEE High Performance Extreme Computing Conference (HPEC), 2022, pp. 1–6.
[27] S. El-Ashry and A. Adel, "Efficient methodology of sampling uvm ral during sim- ulation for soc functional coverage," in 2018 19th International Workshop on Mi- croprocessor and SOC Test and Verification (MTV), 2018, pp. 61–66.
[28] E. Massoud, M. AbdelSalam, M. Safar, and M. Watheq El-Kharashi, "A reusable uvm-systemc verification environment for simulation, hardware emulation, and fpga prototyping: Case studies," in 2022 International Conference on Microelectronics (ICM), 2022, pp. 38–41.
[29] D.-J. Wang and S. Narayan, "Soc verification," in Twelfth Annual IEEE Interna- tional ASIC/SOC Conference (Cat. No.99TH8454), 1999, pp. 25–26.
[30] H. J. Kwon, M.-H. Oh, and W.-o. Kwon, "Verification of interconnect rtl code for memory-centric computing using uvm," in 2021 International Conference on Electronics, Information, and Communication (ICEIC), 2021, pp. 1–4.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY