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.2023.55418

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

Data Transaction using AHB Protocol

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

Abstract: ***The Advanced High-performance Bus (AHB) protocol has gained widespread adoption as a fundamental interconnect standard in system-on-chip (SoC) designs, facilitating efficient communication between various hardware components. This paper presents an in-depth exploration of utilizing the AHB protocol for data transactions within complex digital systems.The primary objective of this research is to investigate the design considerations, analyze the protocol's advantages, and evaluate its performance in data-intensive scenarios. The study delves into the key features of the AHB protocol that make it suitable for handling diverse data transactions, such as read, write, burst, and exclusive access operations. The paper discusses the architectural components of the AHB protocol, including the Master, Slave, and Arbiter modules, and provides insights into their functionalities and interactions. Furthermore, the research presents a comprehensive analysis of the AHB protocol's benefits, including efficient bus utilization, support for multiple bus masters, and the provision for high-bandwidth data transfers.***

Keywords: AHB, Verification, Xilinx Vivado, System On chip (SoC)

I. INTRODUCTION

In the ever-evolving landscape of digital system design, efficient and robust data communication is paramount to ensure the seamless operation of modern computing devices. The Advanced High-performance Bus (AHB) protocol has emerged as a cornerstone in this endeavor, providing a standardized and efficient interconnect mechanism for facilitating data transactions within complex system-on-chip (SoC) architectures. This paper delves into the intricacies of utilizing the AHB protocol to manage data transactions, offering a comprehensive exploration of its design principles, benefits, and real-world applicability. The exponential growth in computing capabilities and the integration of diverse hardware components within a single chip have led to a pressing need for an effective communication infrastructure. The AHB protocol, developed by ARM, has addressed this challenge by offering a versatile and efficient framework that supports various data transaction types, ranging from single transfers to high-bandwidth burst operations. Its versatility makes it an ideal candidate for enabling seamless communication between different functional units, memory modules, and peripheral devices. The AHB is a high-performance bus protocol developed by ARM as part of the AMBA specification. It provides a high-bandwidth, pipelined, and multiplexed bus architecture for connecting various components in a system-on-chip (SoC). The AHB protocol supports multiple bus masters and slaves, allowing concurrent data transactions. This paper seeks to provide a holistic understanding of the AHB protocol's role in data transaction management. It delves into the protocol's architectural components, elucidating the functionalities of the Master, Slave, and Arbiter modules. By establishing a clear understanding of these modules' interactions, the paper sets the stage for a thorough exploration of the protocol's capabilities and advantages. In the context of modern digital systems, where performance and efficiency are paramount, the advantages offered by the AHB protocol are pivotal. Its ability to support multiple bus masters, enabling concurrent data transactions, enhances system throughput and responsiveness. Furthermore, the protocol's hierarchical structure aids in optimizing resource allocation, ensuring a balanced distribution of access privileges among competing masters.

II. LITERATURE SURVEY

Embedded hardware accelerators with constrained resources are increasingly finding applications in security domains. To expedite system-on-chip (SoC) development, an effective approach involving hardware/software co-design and a robust validation platform has become indispensable. The use of Electronic System Level Simulation (ESL), implemented through SystemC, stands as a primary solution for swift hardware modeling and verification. Nonetheless, prevailing simulators often struggle to strike the right balance between precision and performance. Moreover, no ESL simulators have been tailored explicitly for cryptographic SoCs. Addressing this gap, this concise article introduces a novel concept: a virtual prototype (VP) featuring integrated cryptographic accelerators[1]. This VP is designed for a cryptographic SoC built on the RISC-V architecture, with the intent of bolstering functional and performance simulations of the SoC.

©IJRASET: All Rights are Reserved

The VP is meticulously crafted to serve as an adaptable and customizable platform, specifically tailored for cryptographic SoCs through an efficient hardware/software co-design strategy. To achieve a highly authentic hardware emulation, the VP incorporates a flexible AHB-TLM interface and a core timing model[2]. These components collectively enable accurate real-world hardware emulation. Comparative analysis reveals that our custom VP exhibits impressive performance gains over RTL (Register Transfer Level) simulations, demonstrating speed improvements ranging from 10 to 450 times faster. Notably, this enhanced performance comes with a marginal simulation error of merely about 4%.

A. AHB System Design

In a typical AMBA AHB system configuration, several key components work together to facilitate efficient data communication. These components include the AHB master, AHB slave, AHB arbiter, and AHB decoder.The AHB master functions as the initiator of read and write operations, employing address and control information to trigger data transfers. It's important to note that only a single bus master is granted access to the bus at any given time, ensuring orderly communication. On the other side, the AHB slave operates as the responder to read or write operations, reacting according to the specific address space range it covers. The role of the AHB arbiter is critical in maintaining bus integrity. It orchestrates data transfers by allowing only one bus master to initiate each transaction, thereby preventing conflicts and ensuring data integrity. While the core protocol is standardized, the design of the arbitration algorithm can be customized to suit the specific needs of the application. An essential participant, the AHB decoder, tackles the task of address decoding for each transfer. The Figure.1 shows the single master AHB system design. This component deciphers the address information and generates select signals that guide the appropriate slave to respond. An important note is that a single, centralized decoder is a consistent requirement across all AHB implementations.

Designing a system using the AHB (Advanced High-performance Bus) protocol involves various components and considerations to ensure efficient data transfer and communication. Here are the key parts of an AHB protocol system design:

1) *AHB Masters and Slaves:* Identify the AHB masters (initiators of transactions) and AHB slaves (responders to transactions) in your system. Determine the functionalities of each master and slave component.
2) *Address Map and Decoding:* Design the address map to allocate address ranges to different slaves. Implement address decoding logic to generate slave select signals based on the accessed address.
3) *Arbitration Logic:* Implement the AHB arbiter that determines which master gets access to the bus when multiple masters request access simultaneously. Choose an arbitration algorithm based on your system's requirements.
4) *Bus Matrix Configuration:* Configure the bus matrix to connect the masters and slaves, ensuring proper routing of transactions. Define the priorities and access permissions for different masters.
5) *Data and Control Paths:* Design the data paths for transferring data between AHB masters and slaves. Implement control paths to manage control signals, transaction types, and response statuses.
6) *Synchronization and Pipelining:* Address clock domain crossings by implementing proper synchronization techniques. Incorporate pipelining to optimize performance and reduce latency.
7) *Timing and Latency Analysis:* Analyze the timing constraints to ensure that data transfers occur within the required time limits. Consider pipeline stages and potential delays in various components.
8) *Performance Optimization:* Optimize the design for factors like throughput and latency. Use burst transfers and pipelining to enhance data transfer rates.
9) *Error Handling and Retry Mechanisms:* Implement error handling mechanisms to handle situations like bus errors or timeouts. Design retry mechanisms to ensure data integrity in case of unsuccessful transfers.
10) *Verification and Testing:* Develop thorough testbenches to verify the functionality of the AHB system. Test different transaction types, priorities, and scenarios to ensure proper operation.
11) *Power Management:* Implement power management strategies to optimize power consumption in the system. Consider techniques like clock gating and power domains.
12) *Documentation and Debugging:* Document the design decisions, configurations, and key parameters for future reference. Incorporate debugging features to aid in diagnosing issues during development and testing.
13) *Integration with SoC:* Integrate the AHB system into the larger system-on-chip (SoC) design. Ensure proper interaction with other components, peripherals, and memory modules.

Figure 1 shows a single Manager AHB system design with the AHB Manager and three AHB Subordinates. A Subordinate-to-Manager multiplexor and one address decoder make up the bus connectivity logic. In order to choose the correct Subordinate during the data phase of a transfer, the decoder keeps track of the Manager's address during the address phase. The multiplexor returns to the Manager the corresponding Subordinate output data. The usage of an interconnect component by AHB, which allows arbitration and signal routing from various Managers to the appropriate Subordinates, supports multi-manager architectures as well.

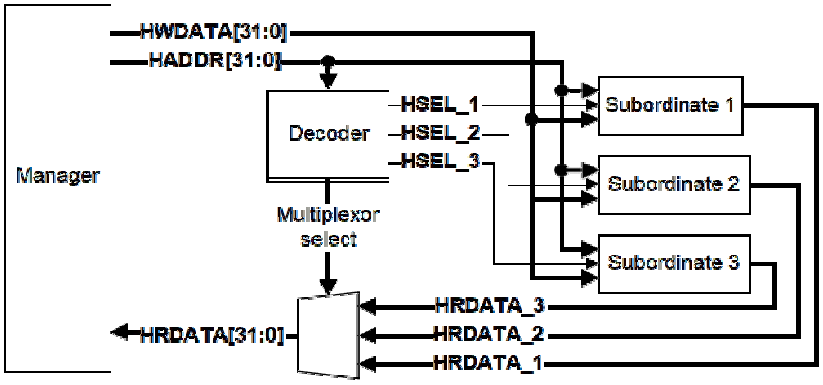


Figure 1 Single Manager AHB system design

B. The Finite State Machine of AHB Master and write operation

The State diagram of AHB master is shown in Figure. 2. The AHB (Advanced High-performance Bus) protocol write channel is a fundamental component of the AHB architecture that facilitates the transfer of data from an AHB master to an AHB slave. This channel is responsible for carrying out write transactions, where the master sends data to a specific address range within a slave device. Here's how the AHB protocol write channel operates:

1) *Transaction Initiation:* The AHB master initiates a write transaction by asserting the HTRANS (transfer type) signal with the value "01" (Write). The HADDR (address) signal specifies the memory address where the data is to be written. The HSEL (slave select) signal indicates the specific slave that will receive the write request.
2) *Arbitration and Slave Selection:* If there are other bus masters competing for access to the bus, the AHB arbiter selects the master based on the predetermined arbitration algorithm. The selected master gains control of the bus, while other masters wait for their turn.
3) *Address Decoding:* The AHB decoder decodes the HADDR signal to determine the specific slave that corresponds to the address range where the data will be written. The appropriate slave device is selected by asserting the corresponding slave select signal.
4) *Data Transfer:* The AHB master places the data to be written onto the HWDATA (write data) signal. The HWRITE signal is asserted to indicate that this is a write transaction.
5) *Data Acceptance and Response:* The selected slave processes the incoming data from the HWDATA signal and performs the necessary write operation. The HREADY (ready) signal indicates that the slave is ready to accept data and that the write operation is complete.
6) *Master Response:* The HRESP (response) signal indicates the status of the write transaction. It can be set to "OKAY" to indicate a successful write or "ERROR" to indicate an error condition.
7) *Transaction Completion:* After the write transaction is completed, the bus is released, allowing other masters to access the bus.

The AHB write channel ensures efficient and reliable write transactions within the AHB protocol architecture. It supports multiple masters and slaves, enabling the smooth transfer of data from masters to slaves across a range of memory and peripheral devices in a system-on-chip (SoC) design. The Finite State Machine (FSM) of an AHB (Advanced High-performance Bus) master is a representation of the various states and transitions that the master goes through during the execution of transactions on the AHB bus. A typical AHB master FSM encompasses the essential phases of initiating transactions, addressing, data transfer, and handling responses. Here's an overview of the common states and transitions in an AHB master's FSM:

The Idle State where the master is not actively involved in any transactions and initiation of a new transaction triggers a transition to the next state. The Address Phase in this phase the master asserts the address and control signals to specify the transaction type (read or write) and the target address and successful completion of address setup triggers a transition to the next state. The Data Transfer Phase in this phase for write transactions, the master places data onto the HWDATA (write data) signal, For read transactions, the master waits for data to be placed onto the HRDATA (read data) signal by the slave and depending on the transaction type, transitions occur when data is written (write transaction) or data is received (read transaction). The response handling phase in this phase the master waits for the HREADY signal to indicate that the slave has completed the operation, The master also monitors the HRESP (response) signal to determine the status of the transaction (OKAY, ERROR) and once the response is received and HREADY is asserted, the master transitions back to the idle state. The error handling phase in this phase if in case of an ERROR response, the master may take appropriate actions, such as retrying the transaction or signaling an error condition to higher layers of the system and depending on the error-handling strategy, transitions may vary.

It's important to note that the specific FSM structure may vary based on the AHB master's capabilities, features, and the intricacies of the system it is part of. The FSM encapsulates the logical sequence of events and control signals that guide the master's behavior during AHB transactions, ensuring proper communication and data transfer between the master and the rest of the AHB system.

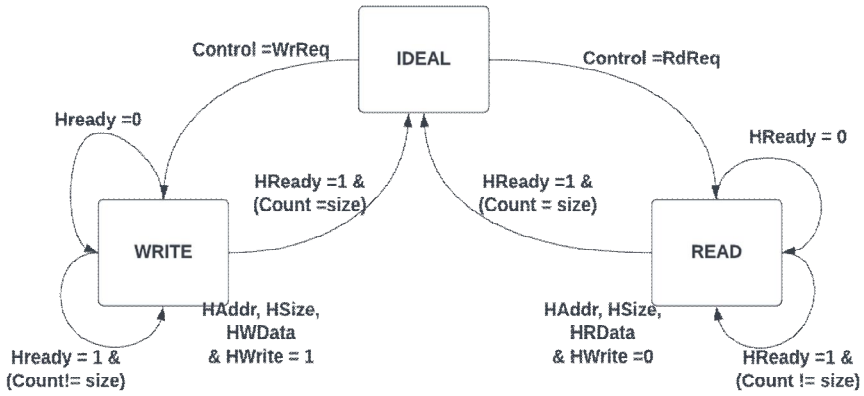


Figure.2 The FSM diagram of AHB master

C. The Finite State Machine of AHB Slave and Read Operation

The state diagram of AHB slave is shown in Figure. 3. The AHB (Advanced High-performance Bus) protocol read channel is a crucial component of the AHB architecture that facilitates the transfer of data from an AHB slave to an AHB master. This channel is responsible for carrying out read transactions, where the master requests data from a specific address range within a slave device. Here's how the AHB protocol read channel operates:

1) *Transaction Initiation:* The AHB master initiates a read transaction by asserting the HTRANS (transfer type) signal with the value "10" (Read). The HADDR (address) signal specifies the memory address from which the data is to be read. The HSEL (slave select) signal indicates the specific slave that will respond to the read request.
2) *Arbitration and Slave Selection:* If there are other bus masters contending for access to the bus, the AHB arbiter selects the master based on the predefined arbitration algorithm. The selected master is granted control of the bus, and other masters wait for their turn.
3) *Address Decoding:* The AHB decoder decodes the HADDR signal to determine the specific slave that corresponds to the requested address range. The appropriate slave device is selected by asserting the corresponding slave select signal.
4) *Data Transfer and Response:* The selected slave responds to the read request by placing the requested data on the HRDATA (read data) signal. The HREADY (ready) signal indicates that the data is available for the master to read.
5) *Master Data Acceptance:* The AHB master reads the data from the HRDATA signal and processes it. If the master successfully reads the data and completes the transaction, it asserts the HREADY signal to acknowledge the successful read.
6) *Response Status:* The HRESP (response) signal indicates the status of the read transaction. It can be set to "OKAY" to indicate a successful read or "ERROR" to indicate an error condition.
7) *Transaction Completion:* Once the read transaction is complete, the bus is released, and other masters can now access the bus.

The AHB read channel ensures that read transactions are performed efficiently and accurately while supporting multiple masters and slaves in a system-on-chip (SoC) design. It enables the seamless retrieval of data from memory or peripheral devices, contributing to the overall high-performance capabilities of the AHB protocol. The Finite State Machine (FSM) of an AHB (Advanced High-performance Bus) slave represents the different states and transitions that the slave goes through when responding to transactions on the AHB bus. The AHB slave FSM typically involves various phases of processing read and write transactions. Here's an outline of the common states and transitions in an AHB slave's FSM: The Idle State where the slave is not involved in any transactions and is awaiting a valid request and the arrival of a valid HSEL (slave select) signal indicating the slave's address range triggers a transition to the next state.

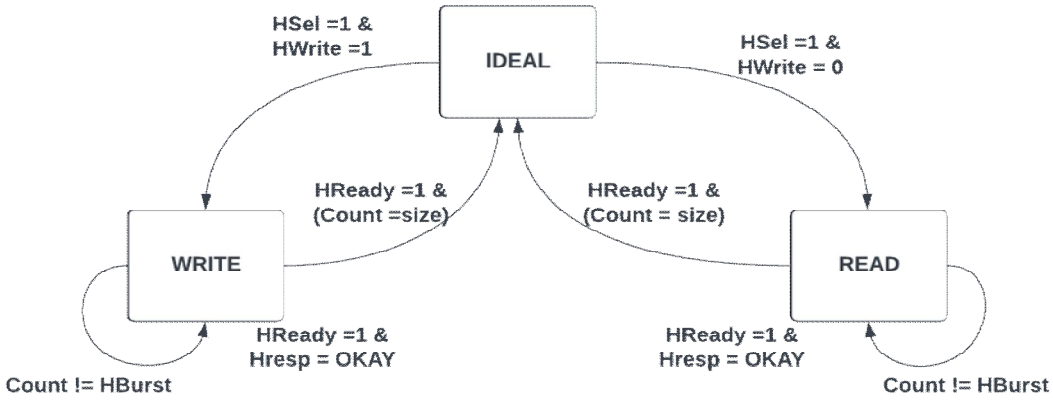


Figure. 3. The FSM diagram of AHB slave

The Address Decode State where the slave decodes the incoming HADDR (address) signal to determine if the transaction is meant for this slave's address range. and upon successful address decoding, the FSM transitions to the appropriate state based on the transaction type (read or write). The read data preparation (Read State) for read transactions, the slave prepares the data to be sent back to the AHB master by placing it onto the HRDATA (read data) signal and transition to the Response Phase. In the write data acceptance (Write State) for write transactions, the slave processes the incoming data from the HWDATA (write data) signal and performs the appropriate write operation and transition to the response phase.

In the response phase the slave asserts the HREADY (ready) signal to indicate that it has completed processing the request, depending on the outcome of the transaction, the slave sets the HRESP (response) signal to "OKAY" or "ERROR." and once the response is asserted and HREADY is acknowledged, the slave transitions back to the Idle State. The error handling phase if the transaction encountered an error (e.g., invalid address), the slave may set the HRESP signal to "ERROR" and take appropriate actions and after handling the error, the slave transitions back to the Idle State. The specifics of the FSM can vary based on the features of the AHB slave and the complexity of the system. The FSM outlines the sequence of events and control signals guiding the slave's response to AHB transactions, ensuring proper communication and data transfer between the slave and the rest of the AHB system.

III.SIMULATION RESULTS

This section presents results of register read and write transactions through AHB protocol. The simulation is carried out with the Xilinx Vivado tool [8]. The Synthesis result of the designwrite transaction is shown in Figure.4. To verify read and write transaction one directed scenario is covered where read and write is done in the register with the particular value. In the ideal state if the hready is 1’b0 and the hres is OKAY than it will goes to the the check mode state. In the check mode if the hresetn, hsel and hwrite all three are 1, it checks for the haddr it should be less than 256 than the next state will be addr decoder. Else next state will ideal state and the hres will be ERROR. The next state is addr decoder in this state the htrans is checked with NON_SEQ and SEQ. If the htrans is NON_SEQ than next addr is haddr and it also check the hwrite value. If hwrite is high than next state is write otherwise next state is read. If the htrans is SEQ than the next addr is retaddr this also checks with the hwrite value. If the hwrite is high than the data is written if not the data will be read.

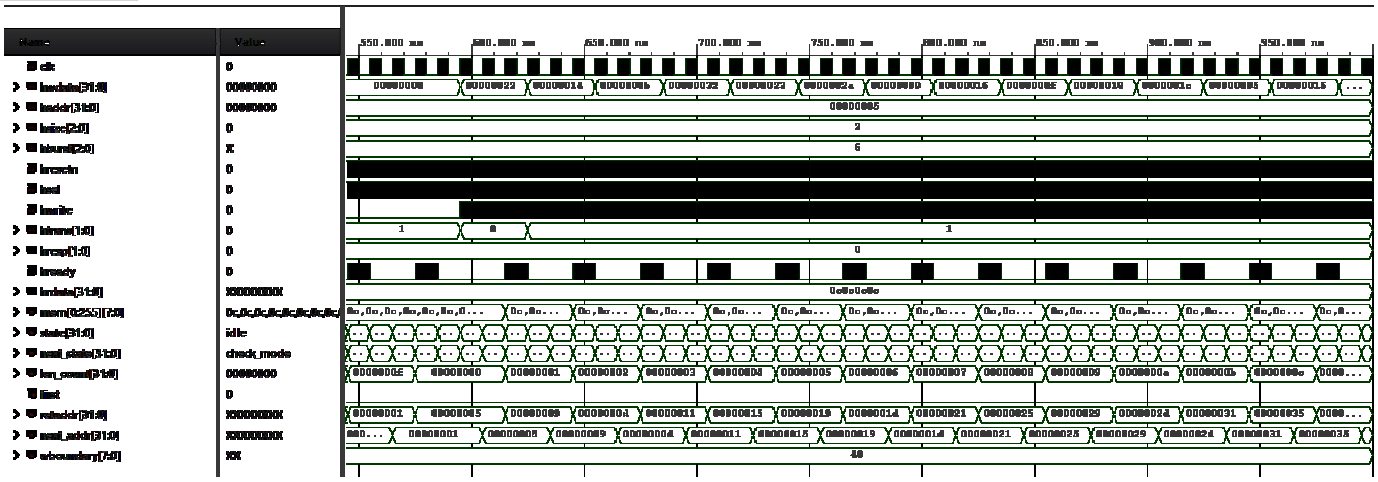


Figure.4 The Write transaction of AHB protocol

In the ideal state if the hready is 1'b0 and the hres is OKAY than it will goes to the the check mode state. In the check mode if the hresetn and hsel are high hwrite is low, it checks for the haddr it should be less than 256 than the next state will be addr decoder. Else next state will ideal state and the hres will be ERROR. The Figure .5 shows the the read transaction of the AHB protocol. The next state is addr decoder in this state the htrans is checked with NON_SEQ and SEQ. If the htrans is NON_SEQ than next addr is haddr and it also check the hwrite value. If hwrite is high than next state is write otherwise next state is read. If the htrans is SEQ than the next addr is retaddr this also checks with the hwrite value. If the hwrite is high than the data is written if not the data will be read.

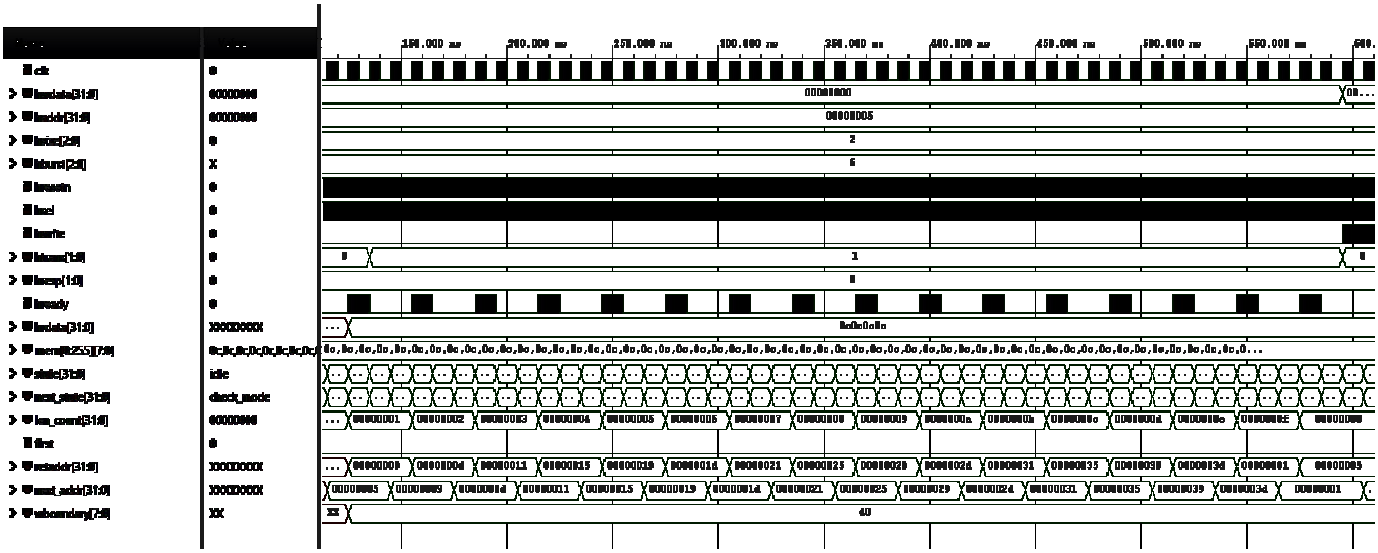


Figure.5 The Read transaction of AHB protocol

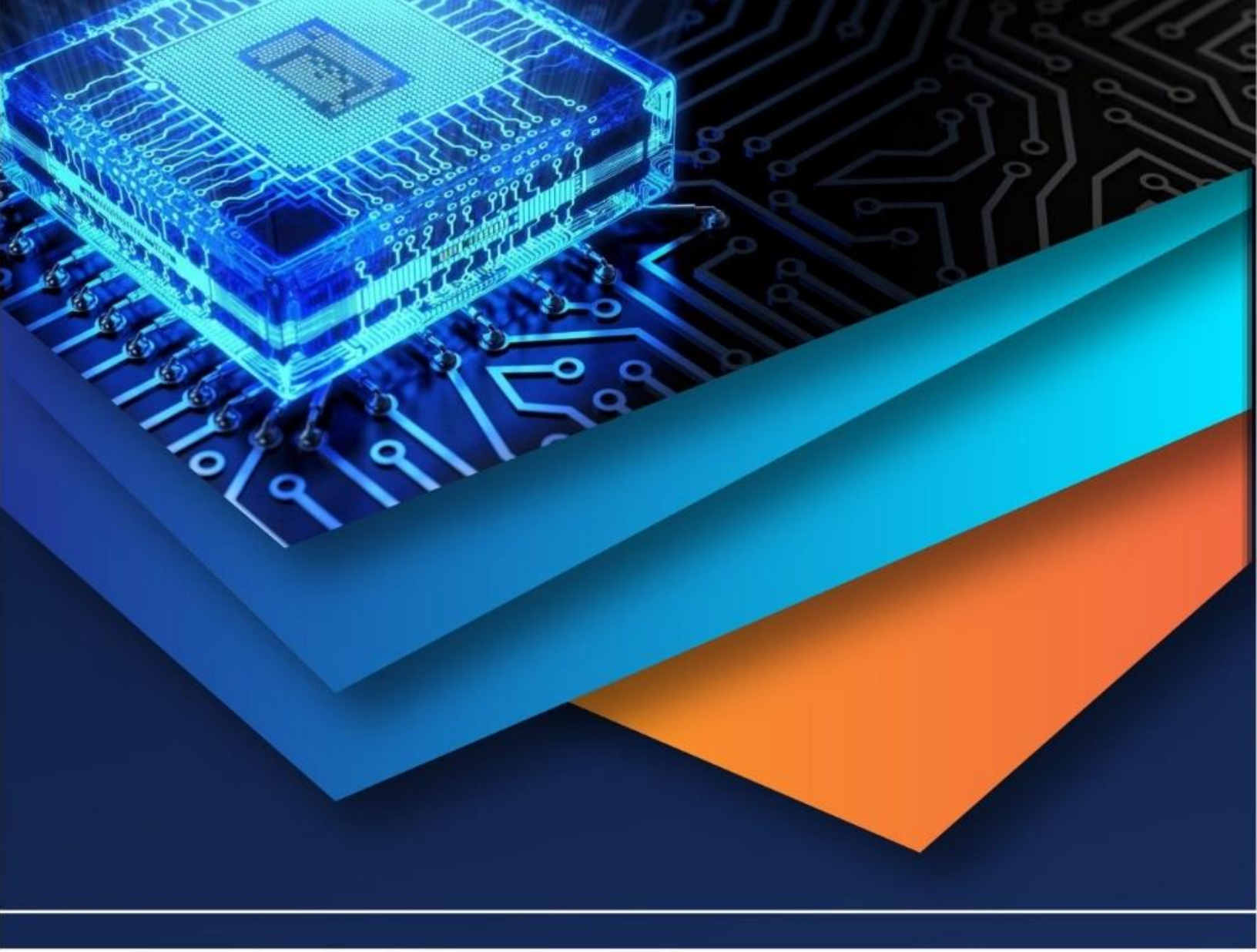
IV. CONCLUSIONS

The AHB protocol is verified by performing write and read transactions from the registers which can be ob- served from the simulation results. In essence, this paper's exploration of data transactions using the AHB protocol underscores its significance in modern SoC designs. By providing valuable insights into design principles, efficiency, and practical implementation, this research equips engineers, architects, and designers with the knowledge needed to harness the AHB protocol's capabilities for optimizing data communication. The findings presented here contribute to the advancement of efficient data transaction management, promising a positive impact on the performance and functionality of advanced digital systems. All simulation is done with the Xilinx Vivado 2020.2 tool. The power consumed by the design is the dynamic power 381.491W and Static power is 1.674W.

REFERENCES

[1] X. Zheng, J. Wu, X. Lin, H. Gao, S. Cai and X. Xiong, "Hardware/Software Co-design of Cryptographic SoC Based on RISC-V Virtual Prototype," in IEEE Transactions on Circuits and Systems II: Express Briefs, doi: 10.1109/TCSII.2023.3267186.
[2] J. Romero, N. Cuevas and E. Roa, "Energy Efficient Peripheral and System Buses for Low-Area and Low-Power SoC Applications," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 5, pp. 866-870, May 2020, doi: 10.1109/TCSII.2020.2984018.
[3] 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.
[4] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal verifi- cation 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.
[5] J. Kumar, Y. Miyasaka, A. Srivastava, and M. Fujita, "Formal verifi- cation 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.
[6] 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.
[7] 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.
[8] 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.
[9] 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 Advancesin Electrical, Computing, Communication and Sustainable Technologies(ICAECT), 2021, pp. 1–6.
[10] T. Strauch, "Dynamic inside-out verification using inverse transactions in tlm," in 2018 Forum on Specification Design Languages (FDL), 2018,pp. 5–16.
[11] 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.
[12] M. Siegel, "Achieving earlier verification closure using advanced formalverifica- tion," in Formal Methods in Computer Aided Design, 2010, pp.275–275.
[13] 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 In- ternational Conference on Electronics and Communication Engineering (ICECE), 2020, pp. 112–116.
[14] 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.
[15] P. Gurha and R. R. Khandelwal, "Systemverilog assertion based ver- ification of amba-ahb," in 2016 International Conference on Micro- Electronics and Telecom- munication Engineering (ICMETE), 2016, pp.641–645.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] 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.

©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)