Utkarsh Das, 2026, 14:6
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

UVM Based Verification of AMBA AHB Protocol with Highest Functional Coverage

Utkarsh Das, Vinod Solanki
Department of Electronics & Communication Engineering,
Prestige Institute of Engineering Management & Research, Indore, Madhya Pradesh, India

Abstract- In the modern design of System on Chip (SoC), various functional units are integrated onto a single chip, requiring a high-performance bus to enable smooth communication between various units of the SoC design. The AMBA AHB protocol is widely used in the VLSI industry due to the high-speed communication, pipelined architecture, and efficient implementation of read/write operations. However, the traditional approach of design verification using Verilog/SystemVerilog is often time-consuming, less efficient, and may not provide comprehensive functional coverage of the design under test. Therefore, this work focuses on the implementation of the AHB protocol using a single master and single slave configuration and the Universal Verification Methodology (UVM) to achieve the efficient verification of the protocol. The proposed methodology utilizes the key components of the UVM library such as the sequencer, driver, monitor, scoreboard, and agent components with the support of the Transaction Level Modeling (TLM) concept to achieve the desired level of reusability of the test cases. Constrained Random Stimulus and Functional Coverage are also employed to assess the performance of the proposed verification methodology. The QuestaSim tool was used for verification tasks in the design, simulation, monitoring, and coverage analysis. The results show correct operation of read and write transactions, correct signal interactions, and correct validation of protocol operation through waveform analysis. Coverage results show that all functional scenarios are adequately tested, and verification is complete. This approach demonstrates a scalable, reusable, and efficient verification methodology, reducing verification time and improving reliability before hardware implementation.

Keywords- UVM, AMBA AHB Protocol, Functional Coverage, SoC Verification, Testbench, QuestaSim, RTL.

I. INTRODUCTION

Modern System-on-Chip (SoC) designs integrate multiple functional blocks, such as processors, memory peripherals and units, onto a single chip. This enable efficient communication between the blocks, a high-performance on-chip bus architecture is required [1] ARM's AMBA (Advanced Microcontroller Bus Architecture) is widely used in the VLSI industry for its reliability, flexibility, and compatibility with high-performance systems.

The Advanced High-Performance Bus (AHB) protocol, belongs to AMBA family, supports high-speed data transfer, multiple masters and slaves, and wide bandwidth, making it ideal for modern SoC designs [2] Traditional verification of AHB using Verilog or SystemVerilog can be time-consuming and less efficient. Universal Verification Methodology (UVM) provides a structured approach that allows verification engineers to build modular, reusable, and scalable testbenches, improving verification efficiency and coverage. UVM employs Transaction Level Modeling (TLM) for communication between components like masters and slaves, and provides utilities for reporting, debugging, and coverage analysis. Global messaging tools are available in the System Verilog UVM Class Library and can be used for both general and failure reporting [3]

The existing literature on the AMBA AHB protocol indicates the development of the protocol with advancements in the design and verification process. Initially, earlier research focused on Verilog-based design for achieving efficient communication between master and slave modules [3]. However, it did not emphasize verification techniques. For achieving reliable designs, a key research effort (2009) implemented assertion-based verification

© 2026 Utkarsh Das. his is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly credited.

techniques in SystemVerilog for improved design validation [4]. This methodology improved functional correctness without relying on UVM for modular verification. Recently, a study integrated arbiter algorithms and constrained stimulus, but it did not emphasize reusability [5]. The methodology (2016) further strengthened modular and reusable verification practices by integrating UVM [6]. They applied this methodology for general-purpose verification, but it did not emphasize AMBA AHB protocols. Later studies applied UVM for AHB protocol verification. More recent work integrated assertion-based verification techniques with randomized UVM to achieve higher coverage, but scalability remains a major concern.

This study presents an organized verification environment for the AMBA AHB protocol using SystemVerilog and UVM to give readers a clear understanding of AHB protocol verification using UVM methodology. It examines signal interactions, read/write transactions, and master and slave operations. To guarantee a modular and reusable architecture, important UVM components like the sequencer, driver, monitor, scoreboard, and agents are implemented. The effectiveness and completeness of verification are assessed using functional coverage. Correct protocol behaviour is confirmed by simulation results, which include waveform analysis and transaction validation. In contrast to conventional techniques, this work uses a scalable and coverage-driven UVM approach, increasing productivity and decreasing manual labour. The effectiveness and novelty of the suggested verification framework for AHB protocol validation are highlighted by this integrated methodology.

The rest of this manuscript is structured as follows. Section 1 introduces an overview of the AMBA AHB protocol, emphasizing its significance for high-performance and high-speed system design. Section 2 describes the concept of master and slave for the AHB protocol, including their role, communication process, and key signal interactions. Section 3 describes the verification architecture of the AHB protocol using the UVM methodology. This includes an overview of key top-level elements and their hierarchy using UVM classes, such as sequencer, driver, monitor, agents, scoreboard, and coverage. Finally, Section 4 describes the results obtained from simulating the verification environment, including waveform analysis, verification of read/write transactions, and functional coverage. In conclusion, Section 5 concludes this manuscript by providing an overview of the verification process and emphasizing the significance of the developed verification environment using the UVM methodology.

II. MASTER AND SLAVE CONCEPT IN AMBA AHB

In the AMBA AHB bus architecture, communication takes place with a master-slave principle, where the Master initiates all communication, and the Slave responds to the communication initiated by the Master. The Master supplies all the essential control and data signals to the Slave during communication. The signals supplied by the Master to the Slave during communication are HADDR (Address), HWRITE (Read/Write Control), HTRANS (Transfer Type), and HWDATA (Write Data). The Slave responds to the Master with HRDATA (Read Data) and HREADY (Transfer Status).

In the write mode of communication, the Master supplies the address and the corresponding data to the Slave. The Slave stores the data at the specified memory location. In the read mode of communication, the Master supplies the address to the Slave, and the Slave responds by sending the required data to the Master. The signal HTRANS indicates the type of communication, such as IDLE, NON-SEQUENTIAL, and SEQUENTIAL, to ensure proper communication flow between the Master and the Slave. The signal HREADY allows the Slave to add wait states if needed.

1. Master

The active component of the AMBA bus architecture is the AHB Master, which is responsible for initiating data transfers. The AHB Master generates address, control, and data signals that are used to communicate with the selected Slave device. The Master requests the bus via the Arbiter, after which it sends the address and control signals to the Slave

device, initiating the data transfer process. The Master may send data to the Slave device during a write operation, while during a read operation, the Master receives data from the Slave device. The entire process is synchronized with the system clock, designated as HCLK, with the reset signal, designated as HRESETn [2], [7] as shown in Figure

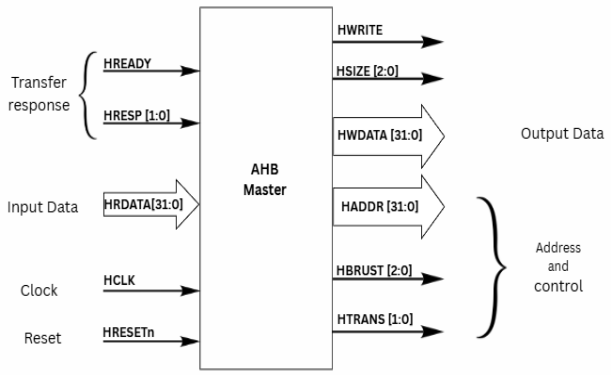


Figure 1: AHB Master

The Master is responsible for the entire communication process, ensuring that the address, control, and data signals are sent in the correct sequence, thus making the communication process efficient [6], [8] . The Master is responsible for sending data to the Slave device during a write operation, while during a read operation, the Master receives data from the Slave device, thus making the AHB bus efficient in supporting different types of data transfers, including single and burst transfers.

The main signals sent by the Master are as follows:

- HADDR – This field is used to specify the memory or peripheral that is to be accessed. It helps the slave identify the memory that is targeted in the operation.
- HTRANS – This field is used to specify whether the transfer is idle, busy, non-sequence, or sequence. It is used to identify the state of the transaction on the bus.
- HWRITE – This field is used to determine whether the transaction is a read operation (0) or a write operation (1).
- HWDATA – This field is used to transfer data from the master to the slave during a write operation. It is asserted only during a write operation, i.e., HWRITE is asserted.
- HREADY – This field indicates whether the current transfer has been completed and whether the bus is ready for the next transfer. This field also helps insert wait states.

2. Slave

The AHB Slave is responsible for responding to the requests that are generated by the AHB Master. The bus signals are always monitored by the AHB Slave, and when a request is selected by the address and control signals, it performs the required operation. The AHB Slave also decodes the address that is sent by the AHB Master. The slave checks whether it is selected, receives data during a write operation, and stores it in the specified memory location. The slave is also responsible for sending the data when a read operation is performed by the master. The flow of data is controlled by the status signals [1]. The slave adds wait states when it is not ready and performs operations in sync with the HCLK clock signal, as shown in Figure 2. The primary signals that the Slave uses:

- HRDATA – This signal is used by the Master to obtain data from the HRDATA signal during the execution of the read operation.
- HREADY – This signal is used by the Master to check if the Slave is ready with the data, whether it has to insert wait states.
- HRESP – This signal is used by the Master to check the status of the operation, whether was successful, i.e., OKAY, or an error occurred, i.e., ERROR.
- HSELx – This signal is used by the Master to select the Slave, as the address is decoded.
- HADDR – The address is sent by the Master, which is used by the Slave to identify the location of the memory.
- HWRITE – This signal is used by the Master to check whether the operation is a write operation, i.e., 1, or a read operation, i.e., 0.
- HWDATA – The data is sent by the Master, which is stored by the Slave during the write operation.

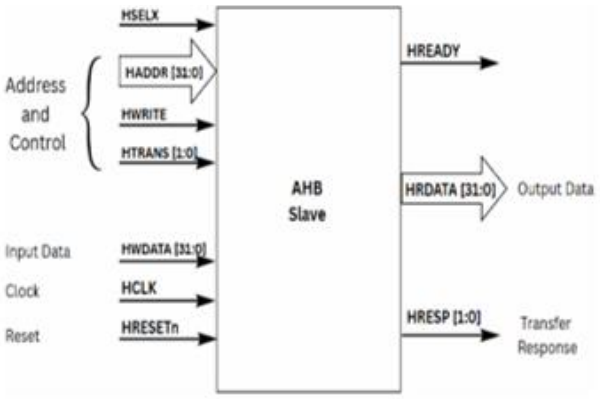


Figure 2: AHB Slave

3. Master-Slave Interaction Flow

The AHB transaction is initiated by the Master, and the signals sent include HADDR, HWRITE, HTRANS, HWDATA, and HSELx. These signals are received by the Slave selected for the transaction. The Slave then decodes the signals sent by the Master and performs the operation. The operation performed by the Slave is based on the type of transaction, either a write or a read. In the case of a write, the Slave receives the data sent by the Master and writes it to the address. In the case of a read, the Slave sends the data received to the Master. The Slave also sends status signals, including HREADY and HRESP, along with the completion of the operation. The completion of the operation is indicated by the HREADY signal, which goes high, and the HRESP signal indicates a successful operation, i.e., OKAY.

4. Clock and Reset

These are the global signals used in the AMBA AHB protocol, and they are used by all components:

- HCLK – All the bus operations are synchronized with the clock signal, hence ensuring proper synchronization and operation of the Master and the Slave.

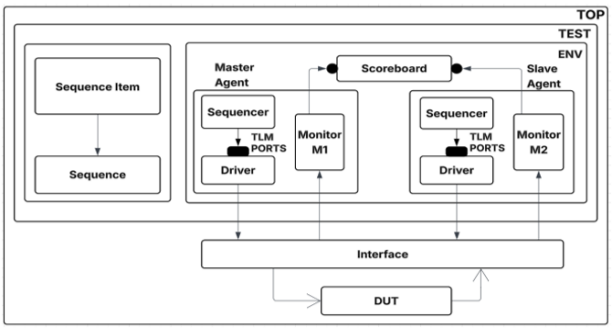


Figure 3: UVM Testbench Architecture

- HRESETn – It is an active low reset signal that resets all the components in the bus to the desired state at the beginning of the operation.

III. VERIFICATION ARCHITECTURE FOR AHB PROTOCOL USING UVM

Universal Verification Methodology (UVM) is an industry-standard approach for the verification of complex digital systems, including System-on-Chip (SoC) designs. The main advantage of using this methodology is that it can significantly improve verification efficiency with less development time required. In this work, a verification environment is created using the Universal Verification Methodology (UVM) for the AMBA AHB protocol. This is done in order to verify the functionality of the AHB protocol in a systematic and complete manner [6], [8]

The verification architecture is based on the key aspects of Universal Verification Methodology (UVM). The key aspects of the Universal Verification Methodology are sequencer, driver, monitor, agent, and score. The sequencer is the component that generates a transactional stimulus and passes it on to the driver. The driver then converts the stimulus received from the sequencer into signal activity in the DUT. The signal activity is then converted back into a transaction by the monitor component. The correctness of the transaction is then validated by the scoreboard component as shown in Figure 3. The verification architecture uses Transaction-Level Modeling (TLM), making it modular, reusable, and scalable. Its components exchange transactions, helping organize the testbench and verify AHB read and write operations efficiently. This structure also makes the verification environment easier to maintain and extend.

1. Top-Level Components

The verification flow begins with the UVM test in which the environment is configured, and the sequence is started using sequencer. Driver converts transactions into AHB signal in order to facilitate communication between the master and slave. The monitor monitors the bus activity, and the scoreboard verifies the results of the transactions

performed while functional coverage counts the number of scenarios covered.

2. UVM Environment (uvm_env)
The UVM Environment is the container of all the components involved in the verification process. It instantiates the driver, monitor, sequencer, and score counter with the help of the driver, monitor, sequencer, and score counter.

3. UVM Test (uvm_test)
The UVM Test is used to implement the test scenario based on the verification plan. It configures the environment and manages the sequence execution with the help of the sequence to test the functionality of the design.

4. Sequence Item (uvm_sequence_item)
The sequence item is used to represent a single AHB transaction. It includes key signals such as HADDR, HWRITE, HWDATA, HTRANS, and HSELx. It can be considered as a data packet at the transaction level that includes all the necessary information regarding the read or write operation. It is the class responsible for defining the data structure used by the verification environment. The sequence item is created by the sequence and then passed to the driver using the sequencer. It is used to provide a systematic way of representing the transactions taking place in the AHB protocol within the UVM environment.

5. Sequence (uvm_sequence)
The sequence is used to generate multiple sequence items to provide different test cases such as read and write operations. It is the class that manages the sequence items and the flow of transactions to test different conditions. The transactions are then passed to the sequencer, which then forwards them to the driver.

6. Sequencer (uvm_sequencer)
The role of the sequencer is to manage the sending of transaction-level stimuli to the driver. It can be defined as the interface between the sequence and the driver that enables the correct communication between the two. The uvm_sequencer is the basic class of the UVM library that offers pre-defined functionality that can be used by the testbench. It does not directly interact with the DUT signals but can handle sequence items that include AHB signals such as HADDR, HWRITE, HWDATA, HTRANS, and HSELx.

7. Driver (uvm_driver)
The driver is the component of the testbench that takes the transaction-level stimuli from the sequencer and converts them to signal-level stimuli that can be used to interact with the DUT. It generates the necessary AHB protocol signals such as HADDR for the address signal, HWRITE for the read/write signal, HWDATA for the write data signal, HTRANS for the transfer signal, and HSELx for the slave select signal.

UVM Agent
The UVM Agent can operate in two modes: active and passive.

Active Mode
In the active mode, the UVM agent is said to actively participate in the DUT drive by sending transactions through the sequencer, which are then sent to the driver. The driver then translates the transactions into signal-level activity, which is then sent to the DUT according to the AHB protocol. At the same time, the monitor is said to collect the DUT signals, which are then used for collecting functional coverage information.

Passive Mode
In passive mode, the UVM agent does not generate or send any transactions to the DUT. Instead, it only uses the monitor to observe the DUT signals and convert them into transaction-level data. This data is then used for analysis and coverage collection. Passive mode is useful for monitoring existing traffic without influencing the design behaviour.

Monitor (uvm_monitor)
The main responsibility of the monitor is to observe and sample the DUT signals without interfering with their operation. It is constantly observing and sampling bus signals such as HADDR, HWRITE, HWDATA, HRDATA, HREADY, and HRESP. These signals are converted to transaction-level data and

are directed towards the scoreboard and coverage model for verification of the correctness of DUT operation. The monitor is an important part in the passive observation and analysis of the system.

Scoreboard (uvm_scoreboard)

The scoreboard is used to compare the expected output with the actual output obtained from the DUT with the help of the monitor. It is used to verify the AHB response signals such as HRDATA, HRESP, and HREADY. If any difference is observed between the expected and actual output, it reports an error and helps in identifying the functional issues in the DUT.

DUT (Design Under Test)

The DUT stands for the AHB master-slave module that is being tested by the UVM-based testbench. It takes input signals such as HCLK, HRESETn, HADDR, HWRITE, and HWDATA to perform the necessary operations such as reading and writing from the memory using the AHB protocol. It then generates the output signals based on the input signals received by the DUT. The output signals are then checked to validate the correct operation of the DUT.

Interface

The interface includes all the signals necessary for communication between the DUT and the testbench using the AHB bus. In UVM, for verification components like the driver and monitor to access these signals, a virtual interface is used. The virtual interface functions as a bridge between the verification components and the DUT. It allows for interaction at the signal level between them. The virtual interface allows for efficient and flexible connection between verification components and the DUT, eliminating the need to instantiate the interface.

UVM Class Hierarchy

The class hierarchy of the UVM is structured to support the verification process. At the lowest level, the uvm_object class is the base class for all classes related to the transaction. This includes the uvm_sequence_item and the uvm_sequence classes. The uvm_component class is derived from the uvm_report_object class and is the base class for all the structural components of the verification environment. This includes the uvm_driver, uvm_monitor, uvm_sequencer, and the uvm_scoreboard classes. These classes are used to construct the verification environment. Higher-level classes such as the uvm_agent, uvm_env, and the uvm_test classes provide a hierarchical structure for the verification environment [9]. This is facilitated by the Transaction Level Modeling (TLM) method of communication between the various classes. This structured hierarchy of the UVM classes provides a better scope for reusability and flexibility as shown in Figure 4.

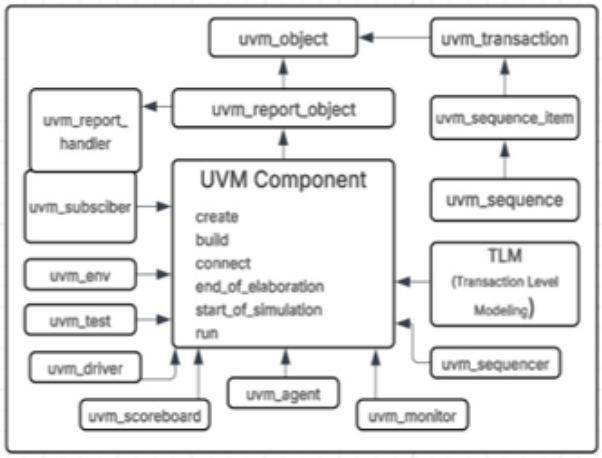


Figure 4: UVM Class Hierarchy

These phases are defined as methods of the class uvm_component.

- Build Phase() – This is where the creation and configuration of all testbench components such as driver, monitor, sequencer, and scoreboard take place. Also, ports, exports, and required parameters are configured here.
- Connect Phase() – At this stage of the testbench, TLM ports and exports connect different components such as sequencer, driver, and monitor to facilitate proper interaction between them.
- End_of_elaboration() – This phase concludes the test bench configuration after all the components have been created and connected.
- Start_of_simulation() – It is used to display the test bench topology and check if all the components have been configured correctly.
- Run Phase() – At this stage of the testbench, sequences are run, and the DUT is driven with

appropriate signals and monitored to perform verification activities.

IV. RESULTS AND VERIFICATION

1. Simulation Overview

The verification of AMBA AHB protocol was carried out using Universal Verification Methodology (UVM) in QuestaSim simulator. The developed verification environment of UVM testbench included sequence, sequencer, driver, monitor, agent, environment, and score board for verification of functionality of master-slave communication of AHB protocol. The test cases were implemented as sequences that generate different kinds of transactions on AHB protocol, such as read and write operations [10]. These operations are applied to Driver using Sequencer. The Driver applies these operations on DUT through Interface.

The Monitor monitors the bus signals (HADDR, HWRITE, HWDATA, HRDATA, HREADY, HTRANS) and sends data to the Scoreboard, which compare the expected results with actual results of the transaction and determine whether there is any discrepancy. This will help verify any read/write transactions and improve the testbench environment. A sample from the simulation waveform illustrates the two-phase pipelined operation of the AHB protocol during read and write transactions. During the write operation (HWRITE = 1'b1), a non-sequential transfer (HTRANS = 2'b10) initiates the address phase at HADDR = 32'h52e004c3. The subsequent sequential transfer (HTRANS = 2'b11) carries the write data HWDATA = 32'hafddea83, with HBURST = 4. For the read operation (HWRITE = 1'b0), the address HADDR = 32'h77c60307 is associated with HBURST = 7, and the corresponding read data HRDATA = 32'h5c064c4b appears during the data phase. These observations illustrate the address and data sequencing and demonstrate the expected read and write transaction behaviour in the simulated AHB interface. as shown in Figure 5.

2. Simulation Transcript

The simulation transcript is available in the QuestaSim environment, and it gives detailed information about the execution of the test and verification results. The transcript indicates that the test "ahb_test" was executed successfully, and multiple read and write transactions were verified using the Scoreboard. Log information, for example, "READ Verified" and "WRITE Verified," as shown in Figure 6 below, confirms that the verification of data transfer between the AHB master and slave is correct [2], [5]. The simulation report is utilized for representing the entire verification results obtained through the verification environment based on UVM. It is observed from the simulation report that 154 messages are sent as UVM_INFO during simulation, and no messages are sent as UVM_WARNING, UVM_ERROR, and UVM_FATAL. This represents that the verification is successful.

3. Highest Functional Coverage Result

The verification results presented in Figure 7 summarize the execution of the AHB testbench in the QuestaSim environment. The simulation completed successfully with zero warnings, errors, and fatal issues, indicating stable testbench execution. A total of 74 informational messages were recorded, including 70 from the monitor ([MON]), two from [Questa UVM], one from [RNTST], and one from [TEST_DONE]. These messages represent transaction monitoring, environment initialization, test commencement, and completion.

The simulation terminated through the $finish system task at 695 ns, at iteration 54 under the root testbench instance /tb. Furthermore, waveform analysis confirmed the expected behaviour of read and write transactions. Combined with 100% functional coverage across all 125 defined coverage bins, these results demonstrate successful execution and support the functional verification of the specified AHB protocol scenarios.

4. Highest Functional Coverage Result

The highest coverage was obtained from the covergroup ‘ahb_cg’ in the module ‘ahb_coverage’. This covergroup is intended for measuring the verification process’s effectiveness by specifying key coverpoints on WRITE and ADDR signal coverage, as well as cross coverage on CROSS_WRITE_ADDR. The WRITE coverpoint is divided into bins for read and write operations, ensuring that both operations are

covered during simulation. The ADDR coverpoint is also divided into bins for low, mid, and high values, ensuring that all address values are covered during simulation. The cross coverage for CROSS_WRITE_ADDR checks the interaction of WRITE and ADDR signals. It can be seen from the coverage report that all bins are covered, thus ensuring 100% coverage as shown in Figure 8.

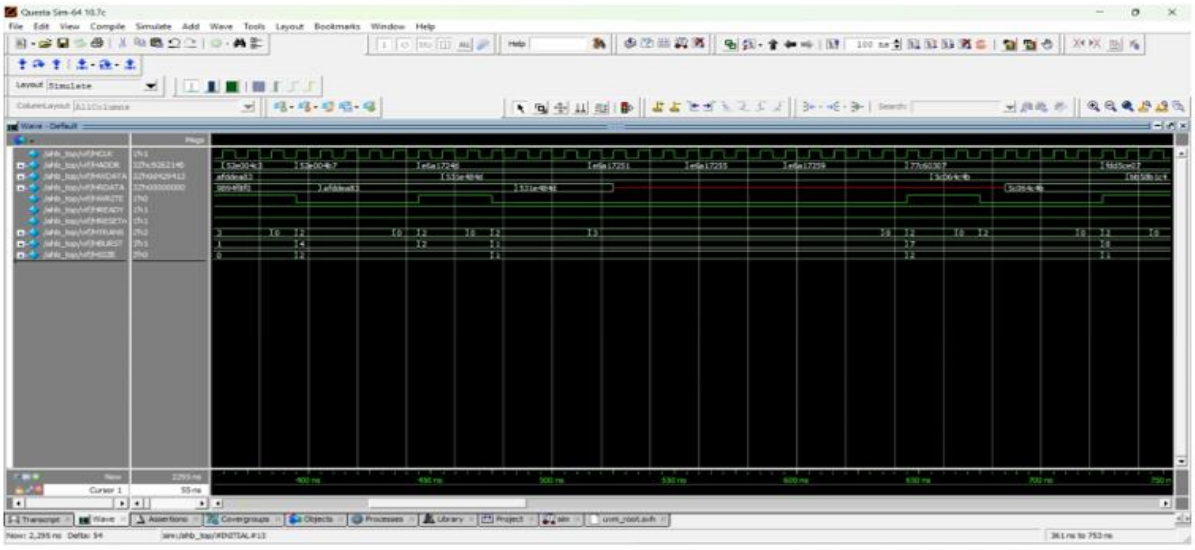


Figure 5: Read and Write Operation.

```
Transcript
#       (Specify +UVM_NO_RELNOTES to turn off this notice)
#
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(215) @ 0: reporter [Questa UVM] QUESTA_UVM-1.2.3
# UVM_INFO verilog_src/questa_uvm_pkg-1.2/src/questa_uvm_pkg.sv(216) @ 0: reporter [Questa UVM]  questa_uvm::init(+struct)
# UVM_INFO @ 0: reporter [RNTST] Running test ahb_test...
# UVM_INFO D:\UVM\ahb\ahb.sv(187) @ 5000: uvm_test_top.env.sb [SB] READ Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(187) @ 15000: uvm_test_top.env.sb [SB] READ Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(187) @ 25000: uvm_test_top.env.sb [SB] READ Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(187) @ 35000: uvm_test_top.env.sb [SB] READ Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(185) @ 45000: uvm_test_top.env.sb [SB] WRITE Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(185) @ 55000: uvm_test_top.env.sb [SB] WRITE Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(187) @ 65000: uvm_test_top.env.sb [SB] READ Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(185) @ 75000: uvm_test_top.env.sb [SB] WRITE Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(187) @ 85000: uvm_test_top.env.sb [SB] READ Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(185) @ 95000: uvm_test_top.env.sb [SB] WRITE Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(187) @ 105000: uvm_test_top.env.sb [SB] READ Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(185) @ 115000: uvm_test_top.env.sb [SB] WRITE Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(185) @ 125000: uvm_test_top.env.sb [SB] WRITE Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(187) @ 135000: uvm_test_top.env.sb [SB] READ Verified
# UVM_INFO D:\UVM\ahb\ahb.sv(185) @ 145000: uvm_test_top.env.sb [SB] WRITE Verified
```

Figure 6: Simulation Transcript Showing Verification Logs

```
Transcript
#
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :   74
# UVM_WARNING :    0
# UVM_ERROR :    0
# UVM_FATAL :    0
# ** Report counts by id
# [MON]     70
# [Questa UVM]     2
# [RNTST]     1
# [TEST_DONE]     1
# ** Note: $finish    : D:/QuestaSim/win64/../verilog_src/uvm-1.1d/src/base/uvm_root.svh(430)
#    Time: 695 ns  Iteration: 54  Instance: /tb
```

Figure 7: UVM Report Summary After Simulation

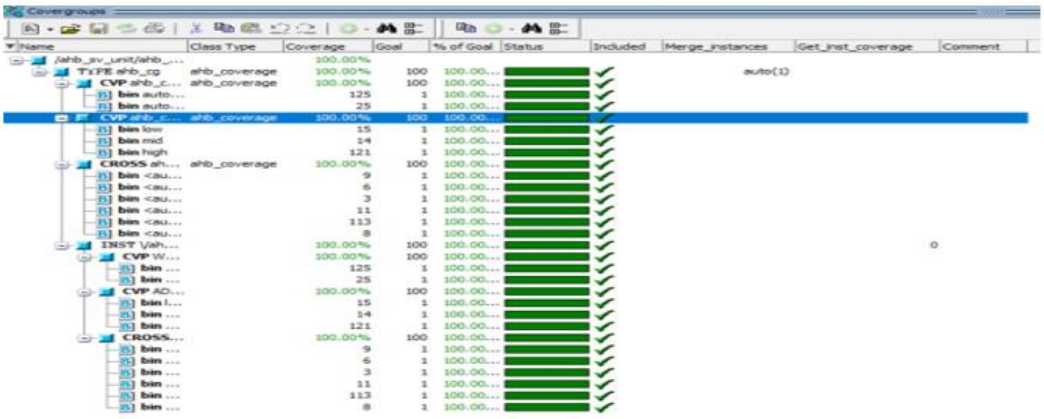


Figure 8: Functional Coverage Results of AHB UVM Verification

V. DISCUSSION AND CONCLUSION

The current research focuses on designing and verifying the AMBA AHB protocol using a structured UVM-based approach. A single-master, single-slave configuration was verified using SystemVerilog, achieving high functional coverage. The testbench uses Transaction-Level Modeling (TLM) interfaces to support communication between its components [4], [9]. Its main components include the sequencer, driver, monitor, agent, and scoreboard. The sequencer generates transactions, which the driver applies to the design under test (DUT). The monitor observes bus signals and forwards transaction data to the scoreboard, which checks the results.

QuestaSim was used to test AHB read and write operations under different conditions. Waveforms and functional coverage were used to check protocol behaviour and system operation. The results show that the defined test scenarios, including selected corner cases, were covered. UVM helped reduce manual effort and identify design issues early. This approach is useful for VLSI and SoC designs that require reliable communication. The verification environment can also be extended for more complex systems and used with industry-standard tools such as Cadence and Synopsys. However, implementing UVM can be challenging because of its complex architecture, debugging through abstraction, and the need to plan phases, virtual interfaces, and coverage models carefully. These challenges can be managed through proper test planning and a good understanding of UVM concepts. Overall, this study shows that UVM provides an effective and scalable approach for verifying the tested AHB protocol scenarios.

REFERENCE

1. F. A. Gouri, V. Deshpande, S. B. A. Gadad, and T. C. Priyatam Kumar, "Diagnostic Access of AMBA-AHB Communication Protocols."
2. S. K. Munuswamy and A. Professor, "AMBA AHB Bus Protocol Checker." [Online]. Available: www.ijert.org
3. ICISS-2019 : proceedings of the International Conference on Intelligent Sustainable Systems (ICISS 2019) : 21-22, February 2019. IEEE, 2019.
4. 2009 Second International Conference on Emerging Trends in Engineering and Technology. IEEE, 2009.
5. K. A. Radhakrishna Rao hodec and pesceacin P. E S, "AHB design and verification AMBA 2.0 using System Verilog," 2018. [Online]. Available: www.IJARIIT.com
6. P. S. Vitankar and A. K. Kureshi, "UVM Architecture for verification," International Journal of Electronics and Communication Engineering & Technology, vol. 7, no. 3, pp. 29–37, [Online]. Available: http://www.iaeme.com/IJECET/issues.asp?JType=IJECET&VType=7&IType=3JournalImpactFactor
7. S. Shibin and L. Babu, "Design and Verification of AHB Protocol using Universal Verification Methodology (UVM)." [Online]. Available: www.ijert.org

8. T. H. N and R. Bothe, "Verification of AHB Protocol using UVM." [Online]. Available: www.ijert.org
9. K. Reshma Nair and B. L. Narayanan, "Systematic Verification of AHB Protocol Using Assertion and Randomized UVM Approach," Journal of Artificial Intelligence and Emerging Technologies, vol. 2, no. 11, pp. 22–27, 2025, doi: 10.47001/JAIET/2025.211004.
10. P. Naveen Kalyan and K. Jaya Swaroop, "VERIFICATION OF AMBA-AHB BASED VERIFYING IP USING UVM METHODOLOGY," 2015. [Online].
11. Miryala, M. (2026). A layered control architecture for trustworthy AI agents on Salesforce Agentforce: Integrating hybrid reasoning, platform enforcement, and human-in-the-loop. International Journal of Engineering and Computer Science, 15(10), 29023–29028.