

AHB Interconnect – Functional Verification

Andrei COZMA¹ and Nicolae ȚĂPUȘ²

¹Tremend Software Consulting

²National University of Science and Technology Politehnica Bucharest

E-mails: andrei.cozma.2525@gmail.com, nicolae.tapus@upb.ro *

* Corresponding author

Abstract. This paper offers an in-depth exploration of the verification procedure pertaining to the integration of an *Advanced High-performance Bus* (AHB) Interconnect module. The verification process holds pivotal significance within chip development, entailing thorough validation and examination of the hardware design before entering the mass production phase. The principal objective of this verification process is to meticulously unearth potential bugs or imperfections embedded in the design, which could potentially trigger undesirable outcomes, compromised performance, or even critical malfunctions in the final product. At the heart of this verification approach lies the functional verification paradigm, centered around simulation-based testing. Within this framework, the AHB Interconnect module is instantiated in a controlled verification environment designed to emulate real-world scenarios. This environment orchestrates input stimuli to the module and captures ensuing outputs generated by it. The environment is meticulously programmed to anticipate specific behavioral patterns from the module. Deviations from these anticipated behaviors are promptly flagged as errors. This meticulous methodology serves to guarantee that the module aligns with its intended operations and strictly adheres to predefined functional benchmarks. The paper is dedicated to Acad. Florin Gheorghe Filip, at his 15th anniversary as the Chairman of the Information Science and Technology Section of the Romanian Academy, and at his 75th anniversary.

Key-words: AHB; Interconnect; Functional verification; Protocol compliance; Concurrency; Arbitration; Master devices; Slave devices; Stimuli and responses; Simulation.

1. Introduction

This paper aims to present comprehensive insights into the AHB Interconnect module, outlining both its essential functionalities requiring testing [1], and detailing the core components constituting the verification environment along with coverage and test results.

Collaboratively crafted with Tremend Software Consulting [2], this design endeavors to construct a verification environment capable of detecting a broad spectrum of bugs within the implementation of an AHB Interconnect module. In the realm of chip fabrication, verification emerges as a pivotal phase, serving to unearth bugs and errors prior to the commencement of the first hardware product's production. Timely bug identification in the early stages of development becomes crucial to avert potential production delays and mitigate escalated financial expenditures for the company [3].

The chip production cycle encompasses four key stages. In the initial stage, a chip architect formulates the chip's specification document, encapsulating comprehensive details concerning features, operational flows, and constraints, including the employed protocol. Transitioning to the second stage, a designer transcribes the information from the specification into code implementation utilizing a Hardware Description Language like Verilog. Subsequently, the third phase encompasses the rigorous verification process. In this phase, the verification engineer initiates the Verilog module constructed by the designer within a dedicated verification environment. This environment conducts tests aimed at ascertaining whether the chip's functionality aligns with the architect's intended specifications. Subsequently, the fourth stage encompasses the physical production of the hardware product [4].

Functional verification entails simulating the chip within a designated verification environment. This environment administers stimuli to the *device under test* (DUT) and captures the resulting outputs. Simultaneously, it processes these stimuli, attempting to generate predictive outcomes. By comparing these predictions against the actual outputs obtained from the DUT, the environment can discern the chip's operational correctness. Notably, the verification process for the AHB Interconnect module adopts a black-box methodology. This implies that the environment lacks access to the internal workings of the DUT and remains unaware of its architectural intricacies. The environment's concern is solely centered around the correctness of the DUT's outputs [5–8].

An imperative aspect entails segregating the roles of the designer and the verification engineer. This separation ensures a robust verification process, as the likelihood of both individuals making identical errors in interpreting the specifications remains minimal.

2. The AHB Interconnect Module

The AHB Interconnect is a complex module that has the role of interconnecting multiple other components and allowing the data transfer between them [6].

The components that the AHB Interconnect module interconnects can be classified as follows:

- Master Devices – these components can initialize data transfers. The data transfers can be read or write transfers.
- Slave Devices – these components have the role of responding to the data transfers from the Master Devices. In case of a write transfer, the Slave Device must save the data, and in the case of a read transfer, the Slave Device must return data.

ARM introduced the AHB (*Advanced High-performance Bus*) protocol and bus in AMBA version 2 in year 1999. The AHB Interconnect module uses the AHB interfaces and protocol to communicate with the Master Devices and Slave Devices. In this paper the following terms will be used:

- AHB bus – the bus architecture that determines the set of signals and their size [6].
- AHB protocol – the set of rules that allows for the data transfer on the AHB bus [6].

2.1. Data transfer flow

The AHB Interconnect module allows for only one data transfer at a time. Because of this reason, the AHB Interconnect module implements a logic for calculating what Master is allowed to make a transfer as shown in Fig. 1. If a Master wants to start a transfer, first it needs to send a request for accessing the bus. The AHB Interconnect knows that each Master has a priority level represented by a number from 0 to the number of Master Devices (lowest number has the highest priority). If only one Master is requesting the bus access, the AHB Interconnect will grant it the access. If multiple Master Devices are requesting bus access at the same time, then the AHB Interconnect must give access to that one Master that has the highest priority [2, 6]. Each Master can make data transfers to any Slave. For that to happen the AHB Interconnect includes logic for decoding the addresses of the transfers. Each Slave is determined by a range of addresses. The AHB Interconnect looks at the address of a transfer and finds the range of addresses that includes it. When the range is found, the AHB Interconnect module will select the Slave that has that range – this is how the Slave will know that it is the one that must respond to that transfer.

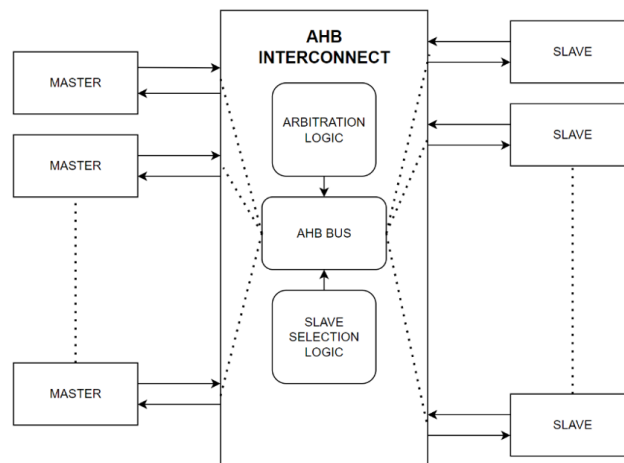


Fig. 1. AHB Interconnect module.

2.2. AHB Interconnects functionalities that need to be tested

The implementation of the AHB Interconnect can be called correct, if each of the following functionalities are tested and validated:

- correct data transfer – protocol needs to be correctly implemented by the AHB Interconnect and all the transfer types need to be supported (SINGLE, INCR, WRAP4, INCR4, WRAP8, INCR8, WRAP16, INCR16) [6].
- correct arbitration of Master Devices requests – The arbitration process needs to respect the priority rule. If there are no requests, the Master with the lowest priority will be granted

access by default. In case of locked transfers, the AHB Interconnect needs to ignore any requests until the lock is dropped.

- correct selection of Slave Devices – the AHB Interconnect must correctly select the Slave based on the transfer address and the ranges of addresses of each Slave.

3. The Verification Environment

Universal Verification Methodology or UVM, was developed by Accellera in 2009 using as platform the *Open Verification Methodology* (OVM). This methodology tries to standardize the structure of any verification environment [7].

UVM uses an object-oriented approach and offers a set of base classes with different roles for all the environment components. Also, the methodology recommends that each environment component to be as independent from the others as much as possible. By adhering to this recommendation, the environment components can be reused in other projects saving time and money.

3.1. The main environment components

The verification environment is composed of multiple components with specific roles. The following sections will describe those components that take a vital role in the verification process as described by UVM.

3.1.1. Agents

The Agent components are the ones that will simulate the behavior of the Master Devices and the Slave Devices. The environment will build one Agent for each Master or Slave component.

Over the course of the simulation, the Master Agents will send to the AHB Interconnect data transfers as if it was a real component. The AHB Interconnect must take care of all the requests from the Master Agents and rout those transfers to the Slave Agents, based on the addresses of the transfers. The Slave Agents will behave like real Slave components, and they will return data if needed, also they can introduce wait states at random, as the AHB protocol allows.

The Master Agents includes three components with specialized roles:

- Master Driver – this component is the one that sends the AHB transactions to the AHB Interconnect. This component has access to the AHB Interconnect interface, and it can assert values to the wires. The Driver must respect the AHB protocol.
- Master Monitor – this component is the one that collects the transactions that the Driver sends to the AHB Interconnect. The Monitor checks that the protocol is respected by the Driver – the burst transfers must follow the rules regarding the sizes and the address values. The information collected will be sent to the Scoreboard component.
- Sequencer – this component provides the data for the Driver to send.

The Slave Agents include only two components:

- Slave Driver – this component has access to AHB Interconnect interface and can save the data that arrives from the Master Agents, can send data back to the Master Devices and can introduce wait states at random.

- Monitor Slave – this component collects the data that arrives to the Slave. The Monitor will check if the AHB Interconnect respects the AHB protocol – the packets that arrive must have the correct sizes and values for the addresses. The data collected will be sent to the Scoreboard component.

The Arbitration Agent is used by the environment to collect all the information from the AHB Interconnect's interface that can be used to analyze the arbitration process. This Agent only includes a Monitor for collecting the data.

3.1.2. Scoreboard

The Scoreboard is the main component that does most of the verification of the AHB Interconnect's functionalities. Using the packets received from the Master Monitors, the Scoreboard will build a prediction. The prediction consists of what Slave should be selected for that said transfer, and the packet itself that should not be altered by the AHB Interconnect. The packets should not experience any data loss and also, the packet order should be kept.

The Scoreboard will receive the data from the Arbitration Monitor. Based on that data, the Scoreboard will check the arbitration functionality. After the Scoreboard analyses the data, it will send it to the Coverage Collector Components.

3.1.3. Coverage Collector type components

The Coverage Collector components are used by the verification engineer to analyze the input values that were used as stimuli for the AHB Interconnect. We want to exercise all the functionalities of the device under test in as many cases as possible. The coverage offers a numeric value that represents a percentage of the input cases that we wanted to test. A specific series of input values represents an event, and we can specify in the coverage all the special events that we assume may hide errors. After a series of tests, or even after one, we can analyze the coverage and see what coverage items never happen. Once we notice that an event did not happen over the course of the simulation, the generation of transactions needs to be revised and improved so that the next time a test is run, that event will take place.

In this environment there are three types of Coverage Collector components:

- The main Coverage Collector – this coverage collector collects all the inputs of the AHB Interconnect and covers the values for every signal, as well as some complex cover items about more special events over the course of a simulation. This main Coverage Collector contains three cover points named as follows:
 - cg – this covers that all the input signals were asserted values from all the important possible ranges. For example, it is important to know that the address signal has had been asserted values from each range of addresses of each Slave. Also, it is covered that during the simulation, all kinds of packet transitions have had place – for example: after a WRAP4 packet, a INCR8 packet followed.
 - cg_arb – this covers all the important events regarding the arbitration process. For example, an interesting case that it is important to know that happened, is the one where a Master sends a request and is waiting for the response, and after some time, it gives up the request.

- `cg_burst_across_slaves` – this covers the cases where the addresses of a burst transfer indicate to more than one Slave. These cases would be interesting, because it is important to know that the AHB Interconnect knows how to split up a transfer and send each transaction to the correct Slave.
- Master Coverage Collector – for each Master Agent, one of these Coverage Collector objects will be initialized. They will cover only the input signals just to see that each Master works as intended.
- Slave Coverage Collector – similar to the Master Coverage Collector, one object of this type will be initialized for each Slave Agent. This Coverage Collector will also cover the signals of the Slave – no special cases are collected.

Fig. 2 illustrates the workflow diagram of the environment. The Master Agents sends stimuli to the AHB Interconnect, and at the same time the stimuli are processed by the Scoreboard in order to generate a prediction. The Slave Agent collects the output results and sends them to the Scoreboard where they are compared to the prediction. During this process the Scoreboard sends the data collected to all of the Coverage Collector type components.

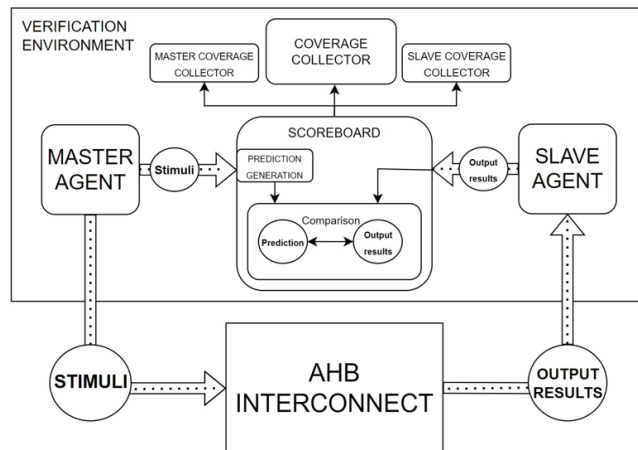


Fig. 2. The flow of the verification environment.

4. Validations of the functionalities

4.1. Validation of data transfer

To validate this functionality there are two main checks that need to be done. The first check is done by the Slave Monitors. The AHB Interconnect has no logic implemented for error packets—in this case, besides the Slave Monitors, the Master Monitors need to perform the same check of the packets that they collect.

According to the AHB protocol, a simple AHB transaction consists of the address and data. This is the smallest data structure that can be determined on the AHB bus. A data transfer or packet can be either a simple transfer that consists of a single AHB transaction, or it can be a

burst transfer that contains multiple AHB transactions. These burst transfers can be classified in one of the two: increment or wrap bursts. The difference between these two is about the value of the addresses of the transactions that make up the packet. In an increment burst, the addresses have incremented values from one transaction to the other. In a wrap burst, the addresses wrap around a certain boundary and form a loop. Both increment and wrap burst can have multiple versions depending on the number of transactions inside them.

The Monitors will collect each transaction from the interface and will build up the packets. Once a packet is built, the addresses will be checked to see if they respect the protocol rules. Another rule that needs to be respected and is checked, is that the first transaction of a packet is of nonsequential type, and the rest of the transactions are of sequential type. If a packet does not pass these checks, an error will be signaled. If Slave Monitors signal an error, it means that the AHB Interconnect outputs wrong packets, while if Master Monitors signal an error, it means that the Master Drivers do not respect the protocol and the algorithm for the drive of packets needs to be checked. After a packet passes the checks, it will be sent to the Scoreboard component.

Fig. 3 describes the diagram of the process of collecting the packets. In the diagram is shown as an example a packet that is composed of four transactions. Master Driver will send one transaction at a time to the AHB Interconnect. The Master Monitor will collect the transactions sent by the Driver from the input interface, while the Slave Monitor will collect them from the output interface. Both type of Monitors rebuild the packet with the transactions that they collect. The Slave Monitor adds its ID to each transaction. Before sending the packets to the Scoreboard the packets must be validated.

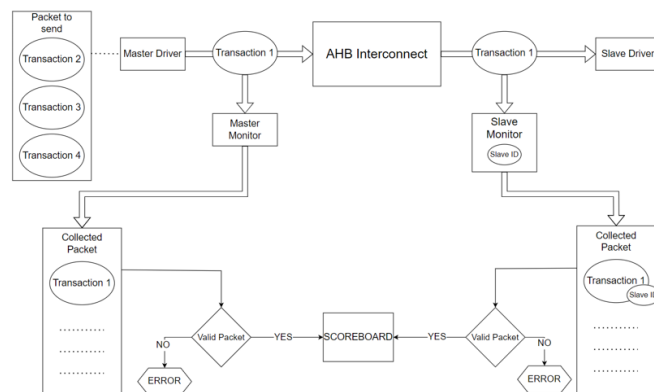


Fig. 3. The process of collecting packets.

The Scoreboard has two FIFOs, one for Master Devices and one for Slave Devices. Once there is at least one packet in each FIFO, the Scoreboard will take them and compare each field inside each transaction of the packets. If the fields match, it can be said that the AHB Interconnect correctly makes the data transfer between Master Devices and Slave Devices. If a single field does not match between the packets, it means that the AHB Interconnect does not do the data transfer correctly. An error can have multiple reasons, for example, the AHB Interconnect can change the order of the packets, or it can completely miss some transactions. In case of an error, the log needs to be checked and sent the error to the designer.

4.2. Validation of Slave selection

The validation of Slave selection is realized at the same time as the data transfer validation. The AHB Interconnect uses two lists named `low_addr` and `high_addr` that represent the ends of each range of addresses for the Slave Devices. The Scoreboard will use the same two lists when it receives a packet. For each transaction of a packet, the Scoreboard will take the address and will search for the range that includes that said address. The position of the range ends from the lists will represent the Slave that is expected to respond to that transaction. The found Slave position will be saved to each transaction structure.

Each Slave Monitor has a unique ID that will be saved to each transaction that it collects.

When the Scoreboard compares all the fields of a transaction as presented in the previous validation process, it will also compare the ID of the Slave and the Slave number calculated with the address ranges. In this way, we will be able to tell if the AHB Interconnect correctly sends the transactions to the correct Slave Devices.

Fig. 4 highlights a diagram of the internal process of the Scoreboard: calculating the Slave ID prediction of the incoming transactions from Master Monitors, and then comparing it to the output transactions collected by the Slave Monitors. In this comparison, each data field inside the transactions are compared, including the prediction of the Slave ID against the real Slave ID.

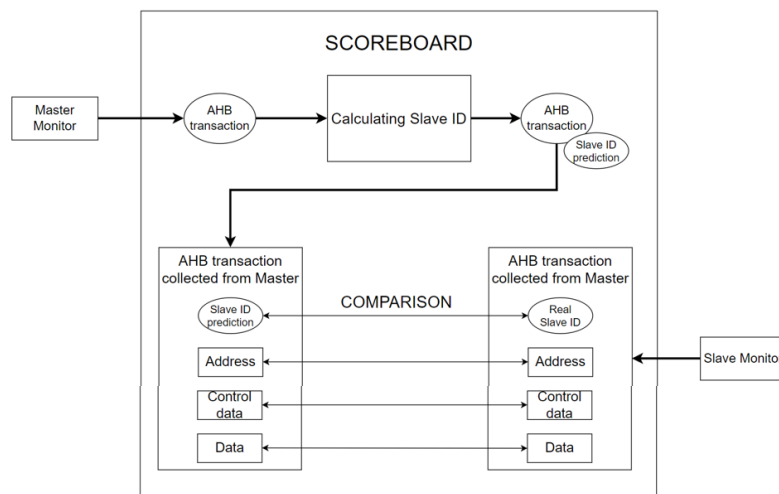


Fig. 4. The validation of data transfer and the selection of Slave Devices.

4.3. Validation of arbitration process

The arbitration of the requests for accessing the bus for transfers is done in two phases. The two phases are the request phase and the response phase. These two phases take place one clock apart.

The Arbitration Monitor will build up arbitration transactions that will contain the following information:

- the requests of every Master – in the request phase.

- the grant signal – in the request phase.
- the grant signal – in the response phase.
- the lock signal – in the request phase.

The Scoreboard will analyze the information as follows: If the Master that was granted the access in the request phase was doing a lock transfer, then the Scoreboard will expect that the AHB Interconnect will maintain the access to the same Master. If the previous Master was not doing a lock transfer, then the priority rule applies, and the Scoreboard will expect the Master with the highest priority that made a request to be granted access. Also, the Scoreboard will expect that the Master with the lowest priority to be granted access to the bus if there are no requests.

If the expectations do not match the real output of the AHB Interconnect, then an error will be signaled that says that the device under test does not perform the arbitration process correctly.

5. Results

5.1. Test and regressions

A test defines what type of packets will be used as stimuli for the AHB interconnect module. There is a test for each type of burst transfer in either read or write form. There is a test that will mix up all the read data transfers and another similar test but for write transfers. The most complete test is the one that will create all kinds of data transfers in either read or write form. In all the tests the Master Agents will randomly send requests for data transfers.

The test that will use all types of transfers is the most complete one and will generate the most events and scenarios. Some interesting scenarios that we would make an interesting case, would be the following: after a read packet follow a write packet, also every transition of burst transfers, for example, a single packet is followed by a undefined increment burst transfer. These cases could uncover a bug in the AHB Interconnects code, and this is why it is important to test them.

For complete verification, it would be best to take the device under test through as many events and scenarios as possible. This could be achieved by either running multiple tests or running a single very long test. Running a very long test is not the best solution because of the time it would take for it to finish. Running multiple tests could offer many results quickly. A regression is the set of tests that could be run for a module. The set of tests are defined in a special type of file with the “.vsif” extension that will include the number of tests, configuration parameters for the tests and any other simulation flags.

For this verification, there will be 105 tests during a regression run. We would like to run as many tests as possible – 105 tests are the maximum number of tests that can be run due to memory limitations of the machine that runs the simulation. The regression will run every type of test multiple times, every test using a different seed, this ensuring that they will be all different. The seed is used in the generation of data – a different seed means different stimuli.

The main condition for calling the implementation of the device under test correct is to have all the tests of a regression pass without any errors.

5.2. Coverage results

Although all the tests pass, another metric for calling the implementation of the module correct is the coverage results. It is very important to know that the tests that the AHB Interconnect passed were tests where many different scenarios happen. The coverage of the entire regression is collected and merged and in the following section the most important coverage results will be presented and described.

The screen capture presented in Fig. 5 gives the coverage results of the main input signals. It can be observed that during the simulation all 8 burst type data transfers took place, also all transitions between these burst types happen. It is important to cover the transition between data transfer types because they may hide unwanted bugs, for example the AHB Interconnect can get stuck between certain transitions. Also, every type of burst data transfer was in read and write form. The BUSY cover point is at 50% because of a limitation inside the AHB Interconnect that doesn't allow for this type of state, in consequence 50% is as expected.













ahb_coverage_collector::cg	 95.45%	116 / 117 (99.15%)	n/a
DATA	 100%	3 / 3 (100%)	n/a
RDATA	 100%	3 / 3 (100%)	n/a
ADDRESS	 100%	11 / 11 (100%)	n/a
BURST	 100%	8 / 8 (100%)	n/a
BURST_TYPE_TRANSITIONS	 100%	64 / 64 (100%)	n/a
TRANS	 100%	2 / 2 (100%)	n/a
IDLE	 100%	2 / 2 (100%)	n/a
BUSY	 50%	1 / 2 (50%)	n/a
TRANS_TRANSITIONS	 100%	4 / 4 (100%)	n/a
READ_WRITE	 100%	2 / 2 (100%)	n/a
BURST_READ_OR_WRITE	 100%	16 / 16 (100%)	n/a

Fig. 5. General coverage of inputs.

The coverage regarding the arbitration feature is highlighted in the image presented in Fig. 6. The ARB_POSSIBILITY cover point describes events regarding the possibility of arbitration at one moment in time. The arbitration is not possible during a locked transfer, and the events covered are revolving around this possibility, for example if there were any bus requests when the arbitration was or wasn't possible, and if there were no bus requests when the arbitration was or wasn't possible. Another event covered is described by the GIVE_UP_REQUEST. This cover point refers to the event in which a Master gives up its bus request during the simulation. Other interesting cases that are covered are regarding the number of simultaneous bus requests at one moment of time. We would like to see that there were no requests, or only one, multiple requests or even the scenario where all the Master Devices make a request at the same time.

ahb_coverage_collector::cg_arb	93.75%	99 / 106 (93.4%)
GRANT	100%	8 / 8 (100%)
GRANT_TRANSITIONS	100%	64 / 64 (100%)
BURST	100%	7 / 7 (100%)
LOCK	100%	2 / 2 (100%)
ARB_POSSIBILITY	100%	4 / 4 (100%)
SIMULTANEOUS_REQUESTS	100%	5 / 5 (100%)
GIVE_UP_REQUEST	100%	2 / 2 (100%)

Fig. 6. Arbitration coverage.

Some more interesting scenarios are the ones regarding the Slave Devices selected over the course of a burst transfer presented in Fig. 7. These cases would be the ones where the addresses of the transactions of a burst are part of the ranges of different Slave Devices. We would cover the cases where the two types of bursts contain transactions with addresses from a single Slave, or two, or even more.

ahb_coverage_collector::cg_burst_across_slaves	100%	6 / 6 (100%)
INCREMENT_ACROSS_SLAVES	100%	3 / 3 (100%)
WRAP_ACROSS_SLAVES	100%	3 / 3 (100%)

Fig. 7. Overall coverage report of selection of Slave Devices during a burst transfer.

6. Conclusions

The conclusion drawn from the regression analysis and high coverage rates is that the AHB Interconnect implementation is deemed accurate. Utilizing the UVM methodology, an environment was constructed to facilitate a diverse range of tests mimicking real-world device usage scenarios. The test outcomes yielded affirmative results, with extensive coverage achieved, encompassing critical events.

This environment possesses versatility beyond its applicability to the current AHB Interconnect implementation. It stands primed for assessing not only this specific module but also other analogous implementations. Moreover, certain components of this environment hold potential for repurposing in distinct projects. For instance, the Monitors can be harnessed in the development of any module that leverages the AHB bus and protocol.

Acknowledgement. This paper is a collaboration between Tremend Software Consulting and CloudPrecis POC 124812 project team.

References

- [1] Functional Verification. Semiconductor Engineering. Accessed: March 31, 2023 [Online]. Available: https://semiengineering.com/knowledge_centers/eda-design/verification/functional-verification/.
- [2] Tremend Software Consulting, Chip Design and ASIC Verification Basics in 2021, Accessed: March 14, 2022 [Online]. Available: <https://tremend.com/blog/engineering-insights/chip-design-and-asic-verification-basics-in-2021/>.
- [3] T. MEAD, What is the Process of Designing a Chip?, 07 Decembrie 2021, Accessed: June 21, 2022 [Online]. Available: <https://www.sondrel.com/blog/what-is-the-process-of-designing-a-chip>.
- [4] K. ALBIN, The cost of bugs, 2014, Accessed: November 20, 2022 [Online]. Available: <http://systemsemantics.com/2014/08/the-cost-of-bugs/>.
- [5] WWW.TESTBENCH.IN, Verilog For Verification, Accessed: May 2, 2023 [Online]. Available: http://testbench.in/TB_34_WHITE_GRAY_BLACK_BOX.html#:~:text=In%20Black%20Box%20verification%2C%20the,the%20box%20doesn't%20matter.
- [6] ARM Limited, AMBA AHB Protocol Specification, 2021.
- [7] Accellera Systems Initiative (Accellera), Universal Verification Methodology (UVM) 1.2 User's Guide, 2015.
- [8] P. T. LAMBE and M. KULKARNI, *Functional verification of AMBA AHB LITE Interconnect using Systemverilog*, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering **6**(7), 2017, pp. 5420–5426.