iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 9 **Issue:** VI **Month of publication:** June 2021

DOI: https://doi.org/10.22214/ijraset.2021.36213

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: 7.429
Volume 9 Issue VI June 2021- Available at www.ijraset.com

Verification of Open Core Protocol using System Verilog and UVM

Darshan
ECE Department (VLSI Design and Embedded systems), PES College of Engineering Mandya, Karnataka (India)

Abstract: ***The ever-increasing complexity of the integrated circuits design and the scale of the projects are making verification more challenging and time-consuming. As a result, the rapidly expanding VLSI industry necessitates a highly reliable and robust verification mechanism. In this paper, System Verilog Verification and Universal Verification Methodologies were adopted to verify the Accellera Open Core Protocol 3.0 as per specifications. According to the verification plan, the environment was developed under a dynamic approach, and the passive aspects included scoreboard, functional coverage, and system verilog assertions. The presented frameworks had verified OCP achieving successful dataflow signals extensions as per results.***
Keywords: On-chip interface, OCP, Protocol Verification, System Verilog, UVM.

I. INTRODUCTION

Over the years, relentless advances in VLSI technology and continuous increase in the complexity of modern SoC designs led to the integration of more IP blocks into a chip. Due to the increasing frequency and the amount of data traffic between the IP cores, the performance of on-chip buses has become a primary factor for overall system performance. As a result, the demand for a more robust and flexible IP -Core interface increases, the need for an open and flexible standard for the efficient on-chip interface is becoming more prevalent. The Open Core Protocol 3.0. is an openly licensed, core-centric protocol standard, which defines a high-performance, synchronous, bus-independent configurable interface for communication between IP cores. OCP consists of an aggregation of signals that aims to unify the communication among IP blocks and simplify the system integration problems [1]. At the outset, there was no standardized testbench architecture described within the verification industry. This led many organizations to the development of strategies that supports different verification scenarios. The surge of developing powerful and standardized verification architecture laid the foundation for future methodologies developed by major EDA vendors such as ARM Ltd. and Synopsys Inc. collaborated on the Verification Methodology Manual (VMM), a professional publication. It outlines a way for utilising System Verilog to validate complex designs. Synopsys developed Universal Verification Methodology (UVM) by combining the approaches of Reference Verification Methodology (RVM) and Open Verification Methodology (OVM), which is a new standard approved by the Accellera committee for verification of integrated circuit designs. The rapid emergence and evolution of verification techniques have necessitated the need for more robust and flexible methodologies such as the SystemVerilog Verification Methodology (VMM) and Universal Verification Methodology (UVM). These two methodologies are widely used in the design and implementation of verification techniques. The System Verilog verification methodology provides robust testbench architectures that includes functional coverage and assertion coverages [2], [3], [4], [5]. The UVM also supports a predefined methodology that is applied in a structured and planned way [6], [7], [8].

Our verification work is made into two-fold, one is to verify OCP by using System Verilog Verification Methodology and another by using Universal Verification Methodology. The paper is structured as follows: Section I covers the introduction. Section II briefly describes OCP 3.0. Section III illustrates the two adopted methodologies and their implementations works. Section IV discusses the verification plan. Section V discusses about result analysis. Section VI concludes the paper.

II. OPEN CORE PROTOCOL 3.0

As the goal of this work is verification of the Open Core Protocol, one needs to understand the specifications of the protocol as discussed in this section. The Accellera Open Core Protocol 3.0 establishes a point-to-point connection between two communicating components, such as IP cores and bus interface modules. In the taken scenario one entity is a slave and another entity is a master, which is developed alongside the testbench. Only the master can be a governing entity, can issue commands, and records the responses collected from the slave. The slave responds to directives given to it by accepting and responding data to the master. The transaction processes are carried out in three phases; Request, Data Handshake, and Response phases. In the request phase, the master presents the commands to the slave, in the data handshake phase master and slave exchange acknowledgments, and in the response phase slave presents the response as requested by the master.

©IJRASET: All Rights are Reserved

For the dataflow signals, the prefix M is used for signals driven by the OCP master, and the prefix S is used for signals driven by the OCP slave. The dataflow extensions exercised for the verification of the OCP are Burst, Tag and Thread Extensions. Fig. 1, illustrates the summary of dataflow signals interface between master and slave.

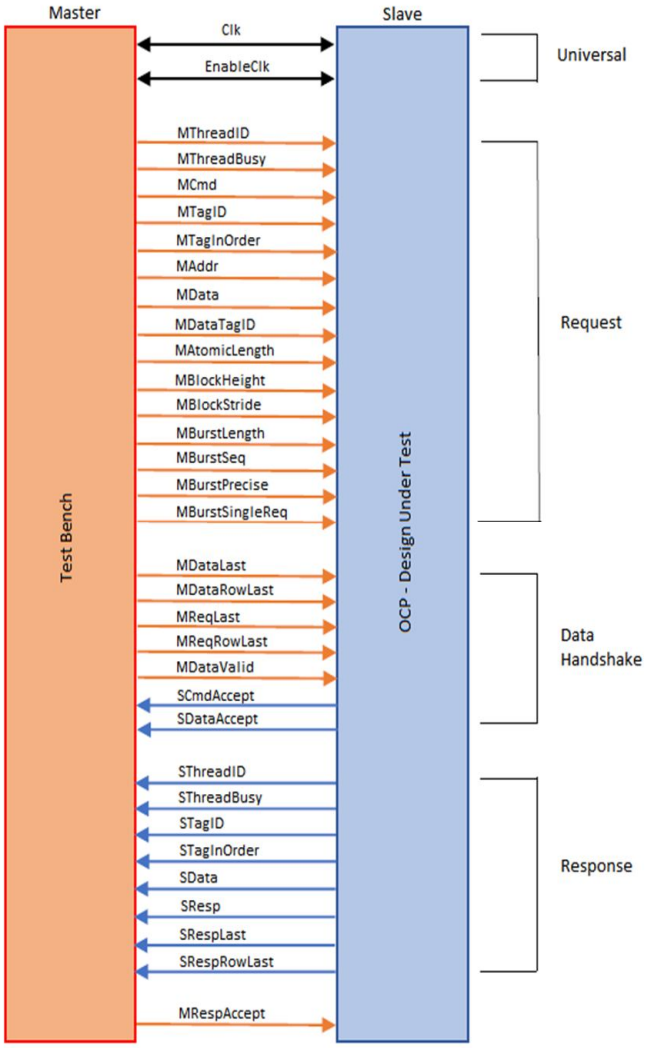


Fig.1. Summary of Dataflow Signals Interface

III. VERIFICATION ARCHITECTURES

The process of testing a design against a set of requirements is known as Protocol Verification. The Verification process evolved as a part of design life cycle, because any faults in the design that are not detected before tape-out can lead to the need for newer stepping and increase the overall expense of the design process. However, as design complexity grows, the scope of verification expands to encompass much more than functionality. While simulation of the design model remains the primary of verification, many alternative methodologies are employed to efficiently verify all aspects of the design prior to tape out. In this section, the verification architectures that are adopted for carrying the verification process of OCP are presented lucidly.

A. System Verilog Verification Methodology

System Verilog Verification Framework is a methodology suitable for verification of complicated SoC components, described by the Verification Methodology Manual. [4], [5]. Fig. 2, depicts the components and their connections of the architecture of System verilog verification methodology.

©IJRASET: All Rights are Reserved

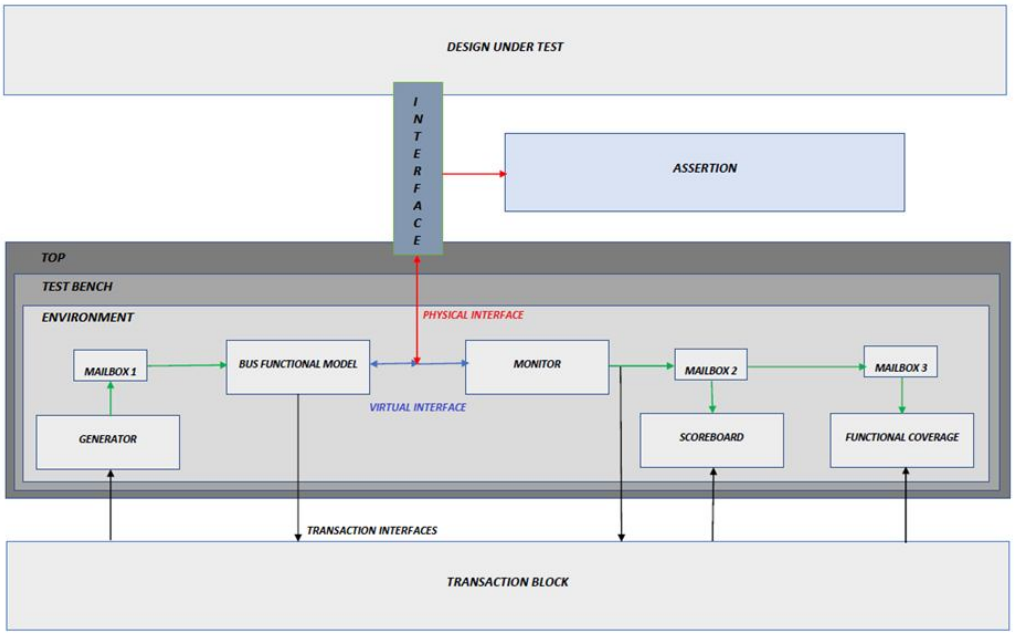


Fig.2. Architecture of System Verilog Verification Methodology

The System Verilog verification hierarchy consists of components to construct the verification environment explained below:

1) Top is the top-level file that generate synchronous signals i.e., Clk, and links the DUT, TestBench, and Interface instances together.
2) Test Bench or Program blocks are mainly used to avoid the race condition problems of test cases.
3) The Environment class serves as a container for higher-level components such as the generator, bfm, monitor, scoreboard, and connected mailboxes.
4) Generator generates the constrained random stimuli that to be driven to DUT.
5) BFM drives stimuli to the DUT via complex waveforms.
6) Mailbox is a way to allow different process to exchange data between each other.
7) Monitor monitors the DUT's input-output interface in order to record design activity and route transactions to higher layers such as Functional Coverage and Scoreboard.
8) Scoreboard verifies the function of design against a reference model.
9) Functional Coverage measures the percentage of stimulation scenarios that are covered.
10) The behavior of protocol characteristics is validated using assertions.
11) Transaction block defines the master properties and placeholder for pin level activities.

The environment simulation was carried out in three stages. Build, Run, and Wrap-Up are the three main phases. The Build Phase assigns and connects testbench components based on the configuration, constrained randomizes the configuration, resets the DUT, and loads commands into registers. The Run Phase runs the testbench components starting from the top: top module synchronizes the signals with generated clock pulses, bfm drives the generated stimuli with a testname wrapped in the testbench to the physical interface and generates the waveforms. The Wrap-Up Phase consists of sweep and report operations: The results of previous cycle are sweeped at the start of the current operation cycle. The scoreboard, coverage and assertion results obtained after executing the run phase are collected in the wrap-up phase. These reports are used for formal analysis to verify the behaviour of DUT [5].

B. Universal Verification Methodology

The UVM testbench is developed by using the verification and analysis components derived from the UVM base class libraries. The UVM package includes base class libraries from which numerous verification components can be derived to construct the hierarchical UVM testbench architecture. The UVM automates the verification process to some extent by introducing new constructs like sequences and adding factory utilities like copy, compare, and so on. For the connectivity of multiple verification components at the transaction level, the UVM enables Transaction Level Model (TLM) based communication interfaces [6], [7]. Fig. 3, depicts the components and their connections of the architecture of Universal verification methodology.

©IJRASET: All Rights are Reserved

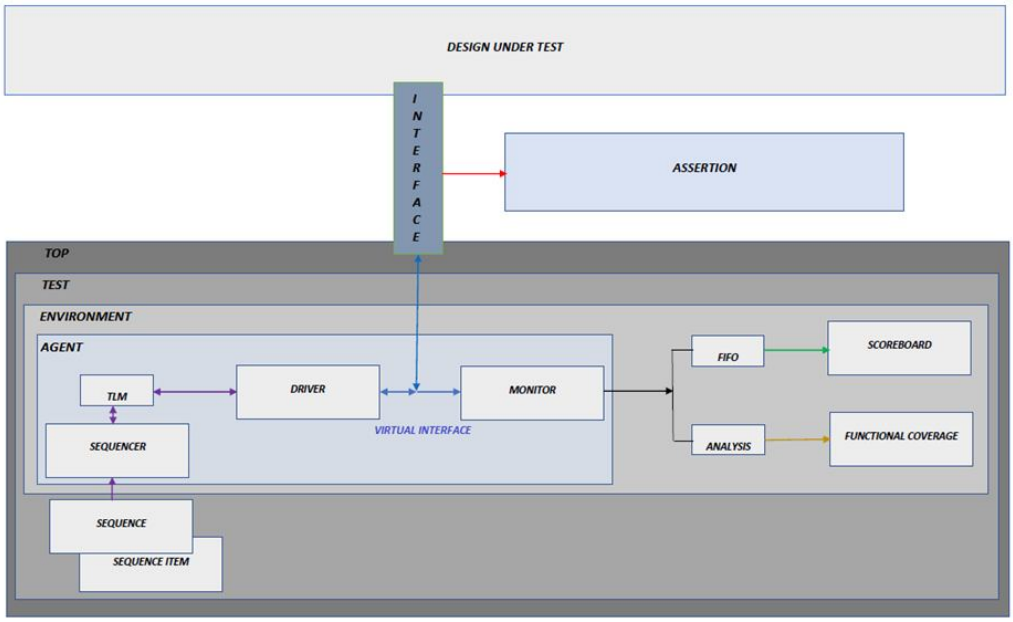


Fig.3. Architecture of Universal Verification Methodology

The UVM testbench hierarchy comprises of a number of verification and analysis components. These components along with their functionality are described below:

1) Typically, the UVM Testbench instantiates UVM Test class and as well as the connections between them.
2) UVM Tests have three major functions: they instantiate the top-level environment, configure it, and apply stimulus to the DUT by running UVM Sequences through the environment.
3) The UVM Environment is a hierarchical component that brings together various interconnected verification components. UVM Agents, UVM Scoreboards, UVM Coverage, and other UVM Environments are common components that are instantiated inside the UVM Environment.
4) The main purpose of the UVM Scoreboard is to check the behavior of a DUT by comparing its functions to a reference model.
5) Functional Coverage measures the percentage of stimulation scenarios that are covered.
6) UVM Sequencer, UVM Driver, and UVM Monitor are all contained within the UVM Agent.
7) TLM is a modelling method for creating very abstract representations of components and systems.
8) The UVM Sequencer acts as an arbiter, allowing different stimulus sequences to govern transaction flow.
9) Individual UVM Sequence Item transactions are received by the UVM Driver from the UVM Sequencer and applied (driven) on the DUT Interface. As a result, by transforming transaction-level stimulus into pin-level stimulus, a UVM Driver transcends abstraction levels.
10) The UVM Monitor collects data from the DUT interface and sends it to the scoreboard and functional coverage for analysis.
11) A UVM Sequence is a type of object that contains stimulus generation activities. All testcase development is done inside sequence exclusively, using body methods (pre body, post body, and body).
12) The UVM Sequence item contains all master properties, methods registration, and constraints.
13) The behavior of protocol characteristics is validated using assertions.

The System verilog and UVM shares similar phases of operation. To ensure a consistent testbench execution flow, the UVM simulation is carried out in an organised manner with a discrete set of three phases: Build Phase, Run Phase and Wrap-Up Phase [6].

IV. VERIFICATION PLAN

The verification plan is essential to any verification effort as it describes the functional requirements of the design and identifies the different features to be verified. The OCP 3.0 verification components are composed of active and passive units. Passive components monitor and log traffic information, whereas active components generate and inject transactions or answer to transaction requests in accordance with OCP standards. [3].

©IJRASET: All Rights are Reserved

The interface forms the connection between the dynamic environment of the testbench and the static entity, which is the OCP - DUT module.

The various components of the testbench accomplish this connection via virtual interface handles. Besides that, analog concurrent assertions are inserted in the interface, which detect the behavior of the OCP during simulations. This section describes the process of functional and formal verification carried out in this work.

A. Functional Verification

The Functional verification process verifies the design from the functional perspective to ensure whether it complies with the specifications. This process involves generating, driving and simulating stimuli to produce waveforms. All master properties from the transaction or the sequence item block are randomized along the datatype and the MData and MDataTagID signals were specified as queue arrays as the data in these signals has to be transferred as unpacked bursts. Here the burst, tag and thread extensions were utilized to carry out verification analysis.

1) *Burst Extension:* The atomic length was divided into byte, half word and word for data transactions. In SRMD burst, only the first request will be sent out for a sequence of data to the MBurstLength. In MRMD, due to 32-bits width size of MData, MBurstLength can have a maximum of 4-word size each comprises of 4 bytes. The slave memory has 8-bit width, so it requires 4 address locations to fit a word-size data.
2) *Tag Extension:* The Tag extension signals gets adds up to the burst properties and performs tagged transactions. MTagInOrder Tags allow out-of-order return of data responses. This operation is carried out for burst sequences except wrap and stream. In wrap, the address gets wraps at aligned MBurstLength*OCP word size and stream uses fifo to store data irrespective of address. In-Order and Out-of-Order types of burst sequences might violate the properties of burst sequences transactions.
3) *Thread Extension:* Multi-Threading allows concurrent transactions. Two threads are utilized to carry out the transaction. Each transfer is assigned to a thread, and if a request is not accepted when a thread is busy, the interface will block all other threads.

All the instantiated verification components are managed by a top module. To exercise all specified features of the protocol extensions, fourteen command testcases and some random stimuli which collectively contributes to the total extensions were generated by constraint randomizing the arguments. The constrained random tests target the different modes of data transfer supported by OCP and check whether the response coincides with the expected behaviour [7]. Individual object transactions from the Generator/Sequencer are received by the BFM/Driver blocks, which drive them on the DUT Interface and generate the waveform. By translating transaction-level stimulus into pin-level stimulus, BFM/Driver transcends abstraction levels.

B. Formal Verification

Formal Verification is a type of functional verification that relies on static analysis to verify the design's functionality without the need for any stimulus. [9]. A variety of formal methods are used to verify a design. Equivalence checking and Property checking are the two methods adopted in the taken scenario.

1) *Equivalence Checking:* Equivalence Checking is an operation that proves that two different implementations or descriptions of the same design have equivalent functionality. The scoreboard in the above situation is a sequential equivalency checking asynchronous FSM model that works in tandem with the DUT to ensure that each data flow precisely follows the given OCP protocol. The scoreboard monitors the proper operation of coherent systems, noting any inconsistencies. They don't make the waveforms, but they do show the status of the comparison.
2) *Model Checking:* Model checking, also known as property checking, is a technique for determining if a finite-state model of a system meets a set of requirements. This comprises Assertion Coverage and Formal based Functional Coverage. Functional Coverage aims to define observations within a DUT that indicate the execution of specific functionality. In this work, a coverpoint encloses all of the protocol properties to measure the applied stimuli scenarios are covered. Explicit Bins, and Implicit Bins are utilized to measure the hits of the applied arguments for the vector signals. Model checking is done with the System Verilog Assertion (SVA), which creates a collection of assertions based on the design criteria to check the behaviour of properties. In this work, three properties are created that verifies the three states, which are idle, write and read operations that covers all specifications of the protocol.

©IJRASET: All Rights are Reserved

V. RESULTS AND DISCUSSIONS

This section gives the result analysis of Open Core Protocol verified in System Verilog Verification and Universal Verification Methodologies. The simulation was carried out using Mentor Graphics QuestaSim tool. Various test scenarios were created in order to verify the functionalities of the OCP thoroughly. All three categories of bursts i.e., precise, imprecise and SRMD/MRMD were implemented in accordance with increment, default 1, default 2, wrap, stream and 2-dimensional block burst sequences.

The waveform shown in Fig. 4, represents the simulation waveform of increment burst in SVVM. In this case, precise increment write-read operation was executed with 0 latency, where the address increments after completion of each burst length of 4 for burst size of 4.

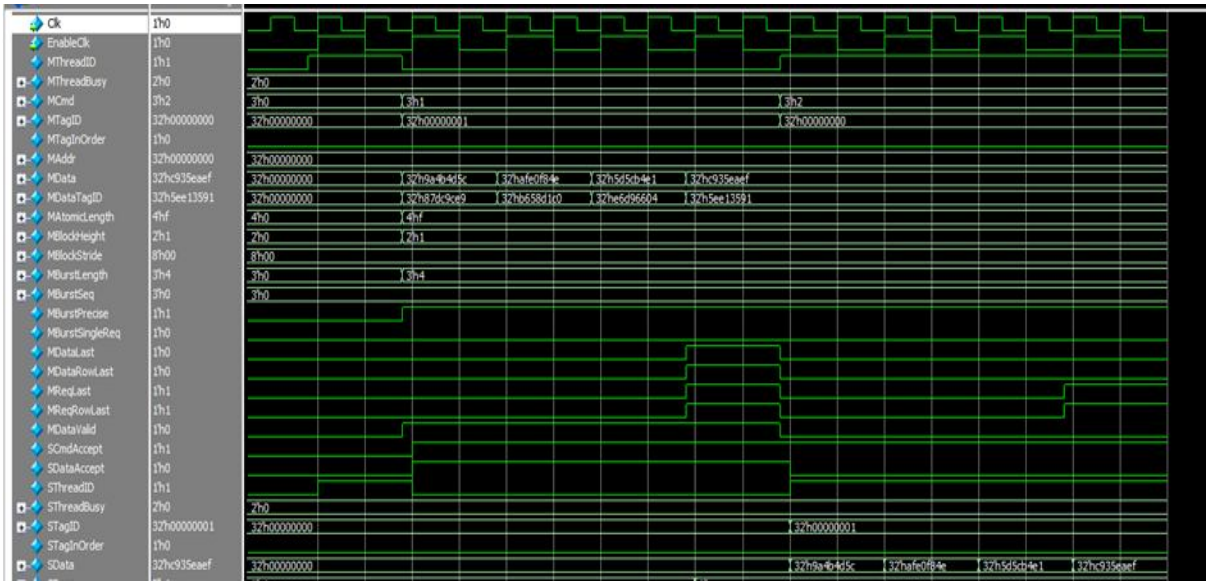


Fig.4. Simulation waveform of Precise Increment burst in SV Verification.

The waveform shown in Fig. 5, represents the simulation waveform of MRMD 2-dimensional burst in SVVM. In this case, imprecise 3 block write-read operation was executed with 0 latency, where the address increments after completion of each burst length of 4 for burst size of 4 and block height increments after completion of 4 burst size.

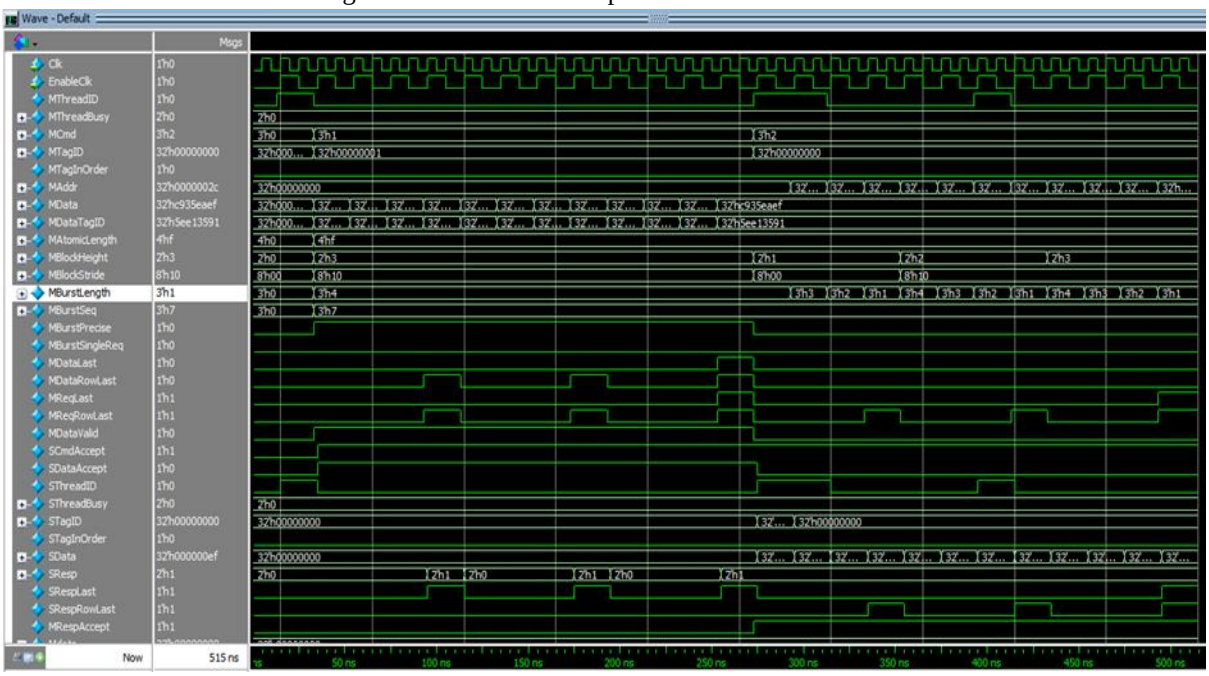


Fig.5. Simulation waveform of MRMD Imprecise block burst in SV Verification.

©IJRASET: All Rights are Reserved

The waveform shown in Fig. 6, represents the simulation waveform of wrap burst in SVVM. In this case, precise wrap write-read operation was executed with 0 latency, where the address wraps to the beginning address after reaching warp boundary.

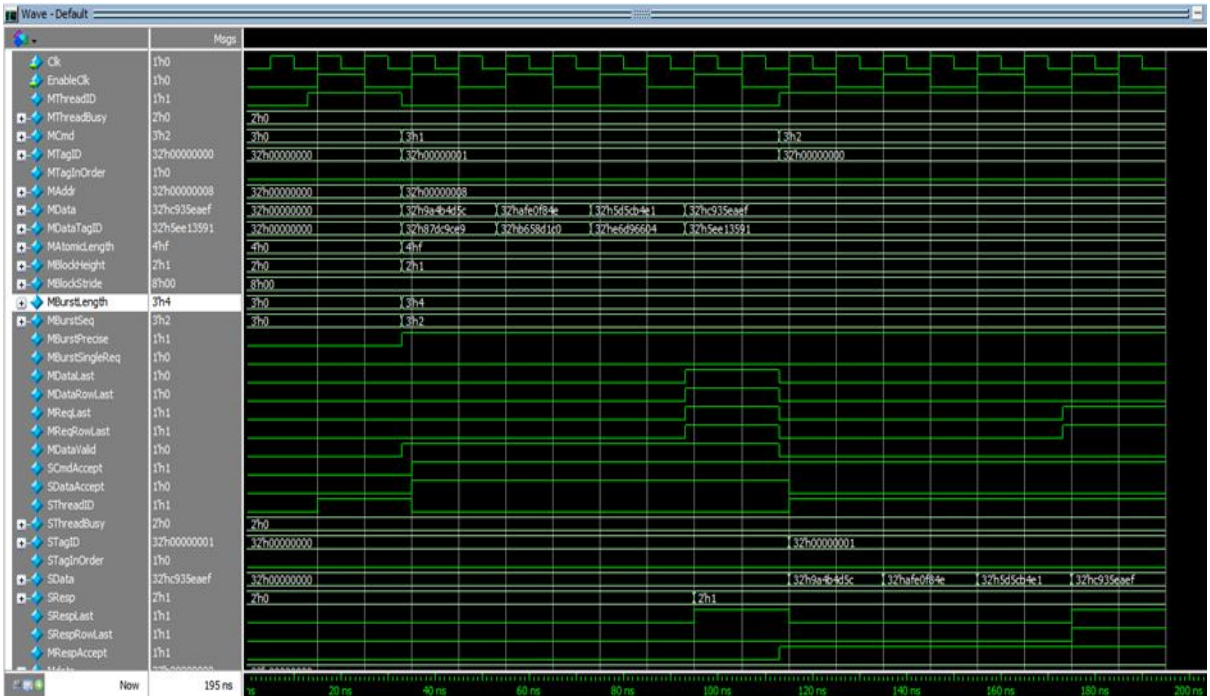


Fig.6. Simulation waveform of wrap burst in SV Verification.

The waveform shown in Fig. 7, represent the simulation waveform of Default 1 burst in UVM. In this case, thread busy operation was executed with 1 latency. When a thread gets busy the command presented to that thread goes to hold state and other threads also blocks.

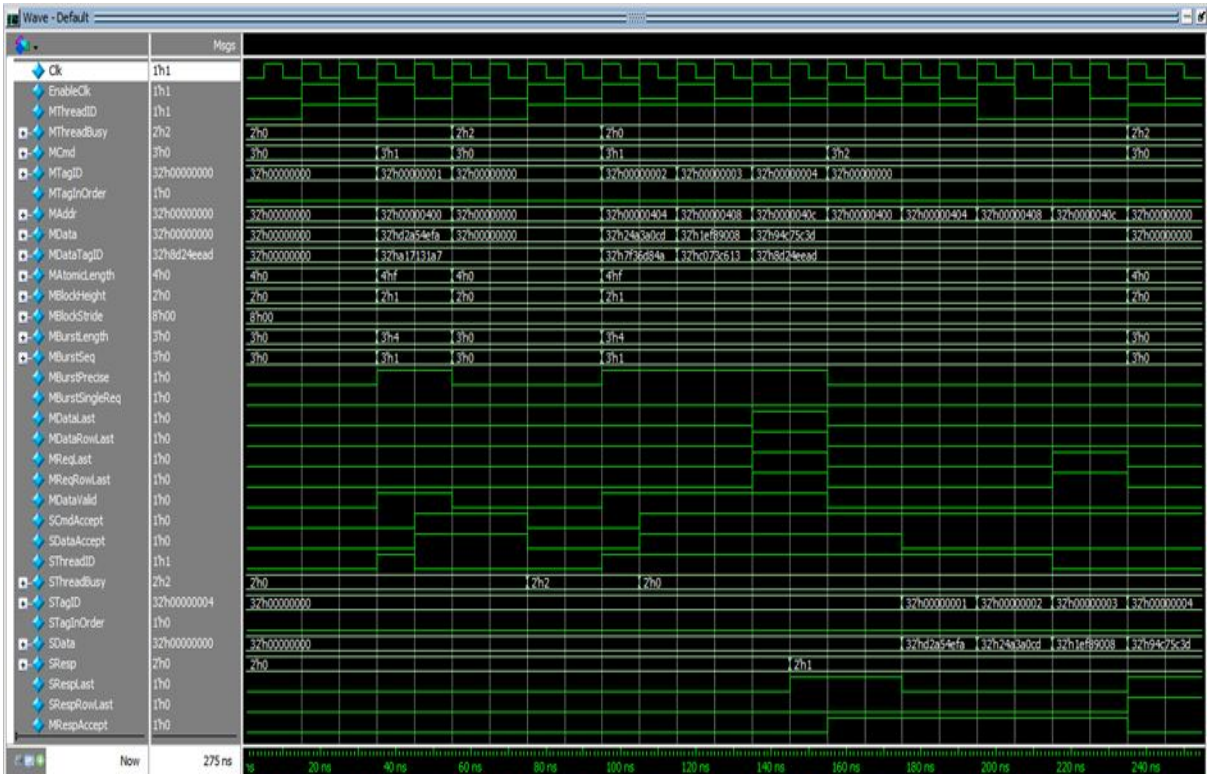


Fig.7. Simulation waveform of Default 1 burst in UVM.

©IJRASET: All Rights are Reserved

The waveform shown in Fig. 8, represents the simulation waveform of SRMD Default 2 burst in UVM. In this case, single request multiple data default 2 operation was executed with 1 latency.

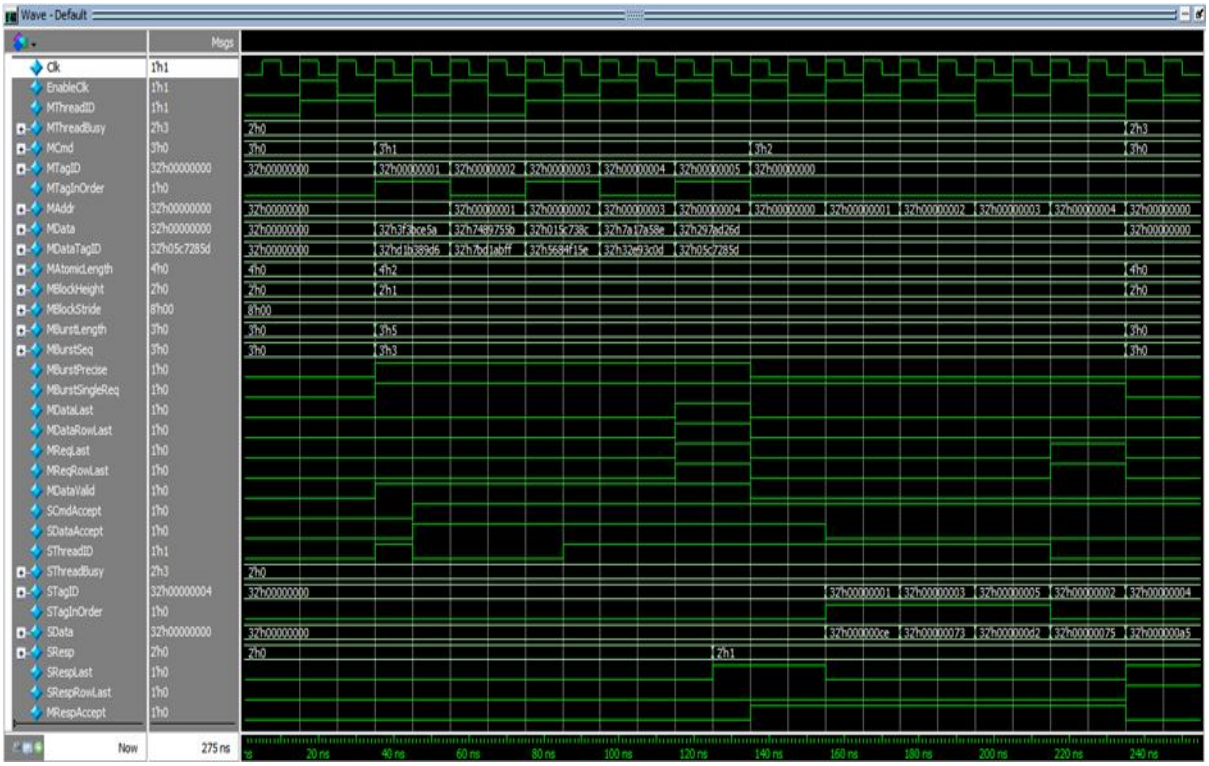


Fig.8. Simulation of SRMD Default 2 burst in UVM.

The waveform shown in Fig. 9, represents the simulation waveform of Stream burst in UVM. In this case, data sequence is saved into a fifo regardless of address.

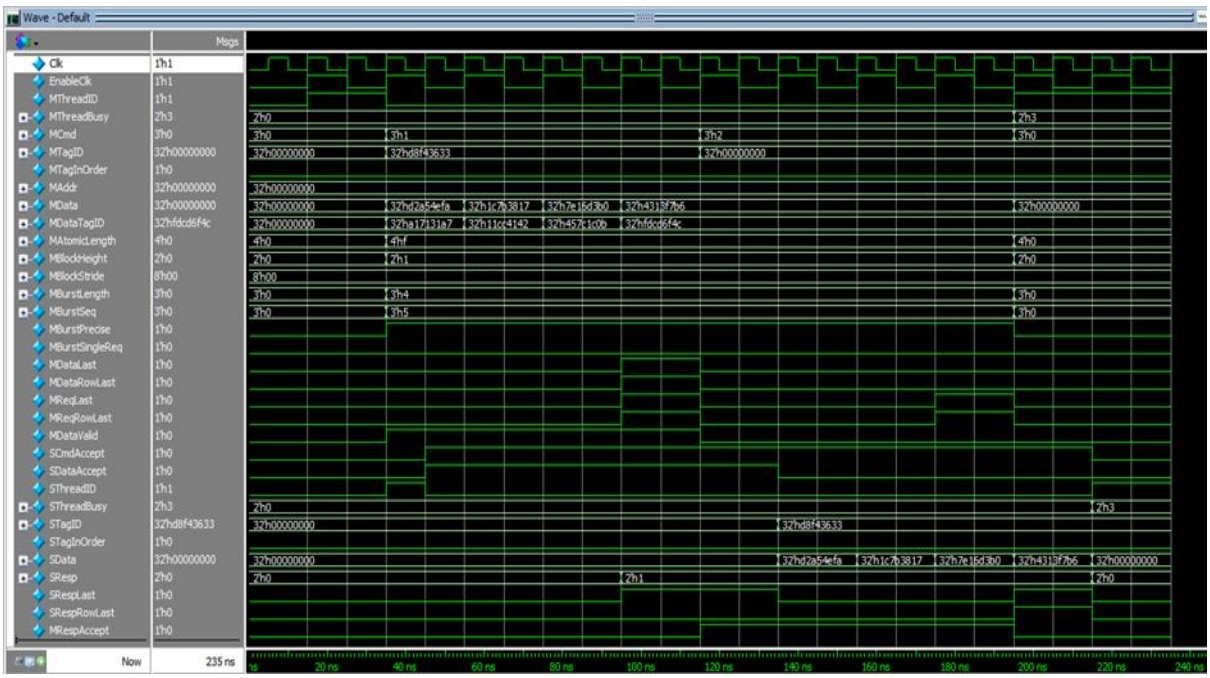


Fig.9. Simulation of Stream burst in UVM.

©IJRASET: All Rights are Reserved

Scoreboard is static in nature and do not supports unpacked array of dynamic data. To overcome this problem a clone of MData signal was implemented, which monitors and plays master data signal to the scoreboard.

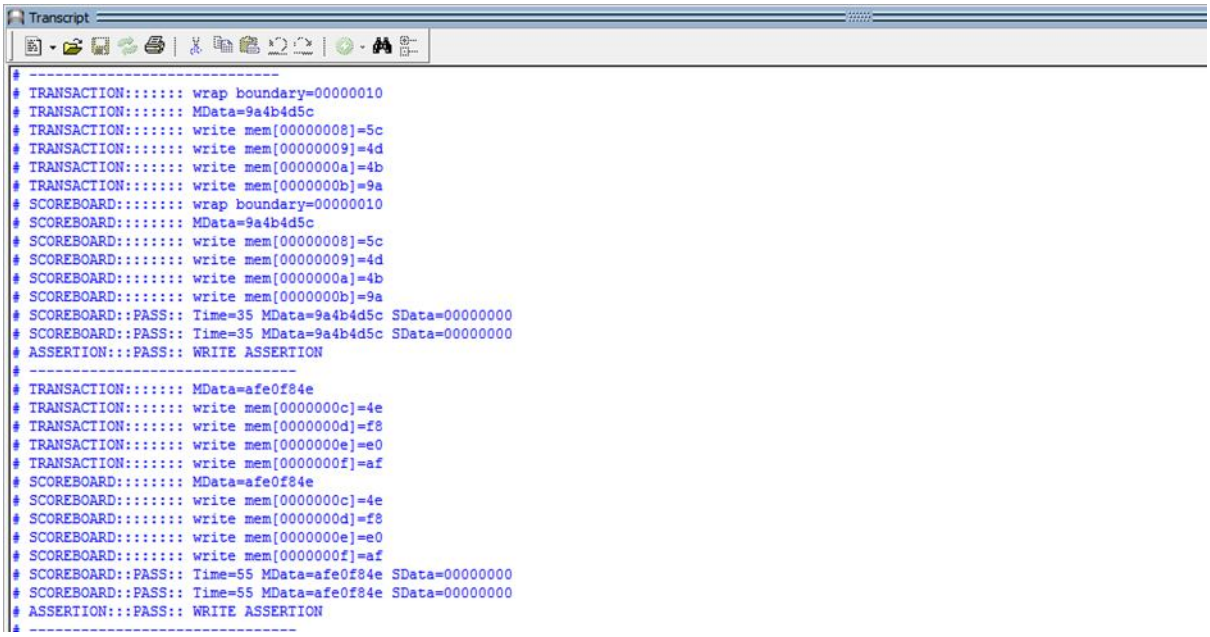


Fig.10. Transcript results of Scoreboard in SV Verification.

The Transcript results of scoreboard are shown in Fig. 10. In this case, the transcript window shows the result of wrap operation, that consists of Wrap boundary values, transaction values, scoreboard values and comparison, assertion results statements.

Assertion and Functional Coverage were used to determine the progress of the verification. The functional coverage collects coverage information defined in the coverage group. Assertion validates the protocol behaviour at DUT interface. 100% Functional Coverage and Assertion coverages are achieved in both methodologies. Fig. 11, depicts the overall functional coverage results in UVM. And Fig. 12, depicts the assertion coverage results in UVM.

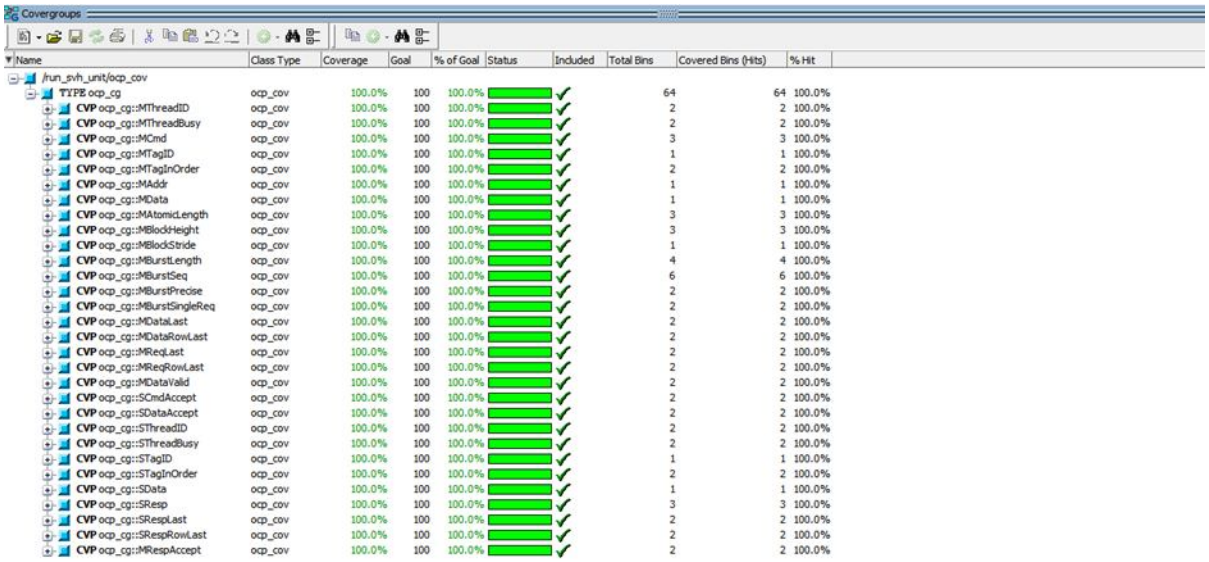


Fig.11. Functional Coverage Report UVM

©IJRASET: All Rights are Reserved

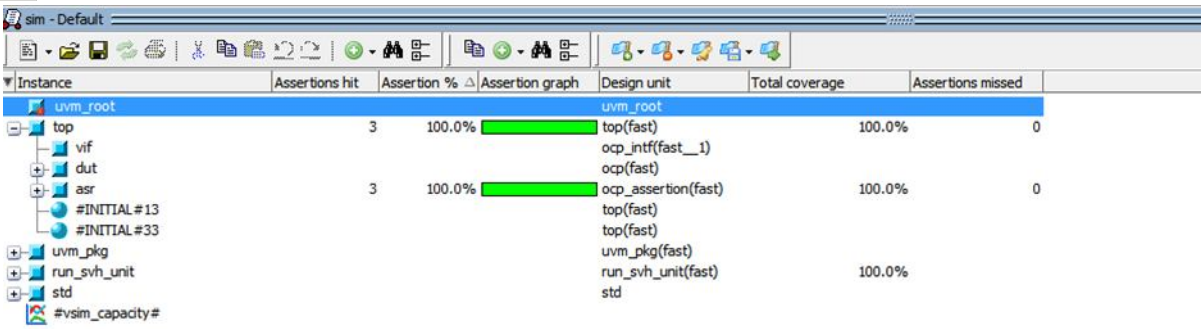


Fig.11. Assertion Coverage Report in UVM

VI. CONCLUSIONS

The proposed verification framework of OCP is discussed briefly along with specified methods of custom verification in System Verilog Verification and Universal Verification Methodologies. The result of verification methodologies indicates that the testbench accomplished verification of OCP, and achieved 100% of functional coverage and assertions that ensure the verification effectiveness of the architectures. More importantly, the presented verification architectures are significant and generalized to block-level verification of SoC designs.

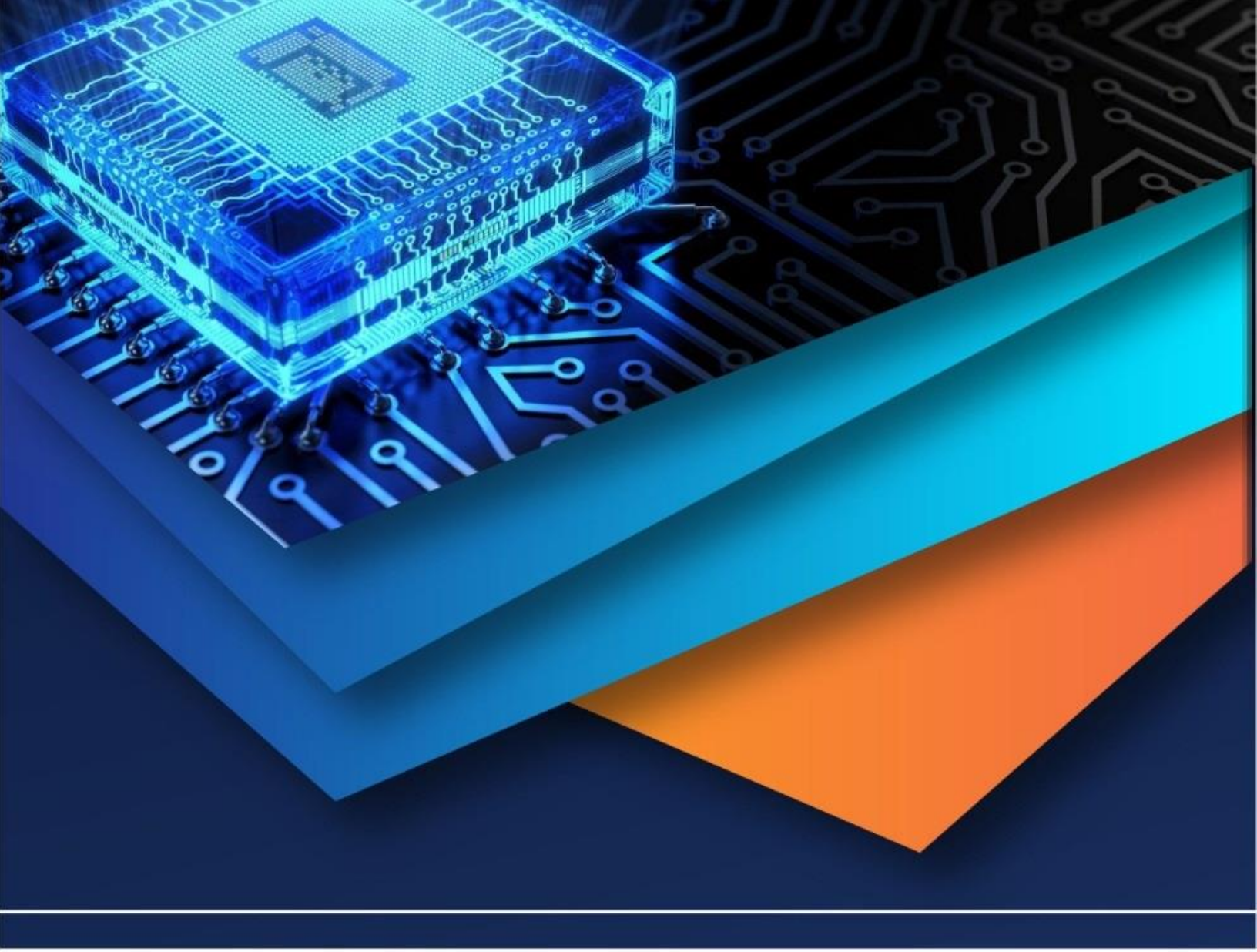
VII. ACKNOWLEDGMENT

Department of Electronics and Communication Engineering PESCE, Mandya. First, I would like to express my gratitude to our guide Dr. Mahesh and HOD Dr. Anand M J, who allowed me to do this project on the topic “Verification of Open Core Protocol using System Verilog and UVM. I would also like to thanks to my parents and friends.

REFERENCES

[1] Open Core Protocol Specification 3.0, Revision 1.0, Accellera Systems Initiative (Accellera), Oct 2013.
[2] S. Zhang, A. I. Ahmed and O. A. Mohamed, "A re-usable verification framework of Open Core Protocol (OCP)," 2009 Joint IEEE North-East Workshop on Circuits and Systems and TAISA Conference, 2009, pp. 1-4, doi: 10.1109/NEWCAS.2009.5290443.
[3] N. Barsotti, R. Mariani, M. Martinelli and M. Pasquariello, "Dynamic Verification of OCP-based SoC," 2005 International Symposium on System-on-Chip, 2005, pp. 22-22, doi: 10.1109/ISSOC.2005.1595634.
[4] M. Keaveney, A. McMahon, N. O'Keeffe, K. Keane and J. O'Reilly, "The development of advanced verification environments using System Verilog," IET Irish Signals and Systems Conference (ISSC 2008), 2008, pp. 325-330, doi: 10.1049/cp:20080683.
[5] Chris Spear: “System Verilog for Verification: A Guide to Learning the Testbench Language Features,” Second Edition, Springer Science+Business Media, 2008.
[6] Accellera, Universal Verification Methodology (UVM) 1.2 User’s Guide, October, 2015.
[7] Accellera, Universal Verification Methodology (UVM) 1.2 Class Reference, June, 2014.
[8] B. Vineeth and B. B. Tripura Sundari, "UVM Based Testbench Architecture for Coverage Driven Functional Verification of SPI Protocol," 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2018, pp. 307-310, doi: 10.1109/ICACCI.2018.8554919.
[9] Ping Yeung and K. Larsen, "Practical Assertion-based Formal Verification for SoC Designs," 2005 International Symposium on System-on-Chip, 2005, pp. 58-61, doi: 10.1109/ISSOC.2005.1595644.
[10] SystemVerilog 3.1a Language Reference Manual Accellera’s Extensions to Verilog®, Accellera Organization, Inc. 2004.

©IJRASET: All Rights are Reserved



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)