Integration and Verification of IP Cores on SoC

Arpitha O Naik
ECE Dept.
R.V College of Engineering
Bengaluru
arpithaonaik.ec17@rvce.edu.in

Elizabeth Kuruvilla
ECE Dept.
R.V College of Engineering
Bengaluru
elizabethk.ec17@rvce.edu.in

Arunkumar P Chavan
ECE Dept.
R.V College of Engineering
Bengaluru
arunkumarpc@rvce.edu.in

2021 IEEE Mysore Sub Section International Conference (MysuruCon) | 978-1-6654-3888-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/MYSURUCON52639.2021.9641547

***Abstract*—As predicted by Moore's law, due to the shrinking transistor sizes, the density of ICs in a chip has increased exponentially and the VLSI trends have seen a sudden drift from computation-based cores to communication-based cores. As a result, the primary challenge in SoC design is to efficiently integrate all IP cores while maintaining their ability to perform their specific functions and communicate with one another, and the quality of integration must be verified at the SoC level, which confirms the same. The paper showcases the design of the various IP Cores such as interface controller block, PCIe block, Ethernet block, throttle and pipeline blocks, and various protocols used to transfer information between the cores via Network-on-Chip. It also explains the process of verification of the SoC design using Universal Verification Methodology(UVM).The IP Cores were integrated with no errors, zero undriven inputs ports and multi-driven ports. The developed test cases for verification were found to achieve a 100% pass after subsequent runs on the SoC. Additionally, the use of UVM helped to achieve a functional coverage of 88.35%, which is seen to be higher than coverage achieved in the absence of UVM.**

***Index Terms*—Intellectual Property Cores, Network on Chip, System On Chip, Universal Verification Methodology.**

I. Introduction

In the modern times, a mobile phone or a tablet can perform all the functions that were limited to be performed only by a desktop and laptop. This is conceivable because a CPU's entire mother board is synthesised on a single silicon chip known as System on Chip[1]. SoCs have Intellectual Property Cores which perform specific functions and communicate with each other using Network on Chip(NoC) and interfacing protocols[2]. It is essential for inter-connectivity of the IP cores to be performed correctly to ensure smooth transaction of data and control signals among them. Once these IP cores are integrated, design Verification is performed. The design verification step evaluates the quality of the design and ensures its accurate working by uncovering potential errors in both the design and the architecture of the system.[3] Design verification can be performed at the block/IP level as well as at the SoC level. SoC level verification not only focuses on the top-level functionalities of the SoC, but additionally verifies whether the interactions between the various blocks take place in an error-free manner.[4]

Connectivity faults account for a high amount of design integration issues, which can arise from the connectivity specification and the SoC code generation scripts[5].SoC designers may utilise a standard such as an automatic SoC integration methodology IP-XACT to document SoCs, or spreadsheets to describe pin level connections. IP-XACT standard methodology offers a effective solution for low-level RTL simulation[6]. This paper uses IP-XACT methodology following a Bottom-up approach for integration from component level, to SoC core level, and then to final chip level.

The design verification of an SoC requires the use of good verification methodology. The use of Universal Verification Methodology for verification is highly popular for design verification at SoC level [7]. It shows high reusability and reduced verification time. A similar methodology, called AVM is described in [8]. It shows significant use in high-performance communication verification. However, UVM shows higher code coverage compared to AVM. In order to compare the pros of UVM against other methods, [9] compares UVM against traditional verification methods. WISHBONE SoC interconnect architecture is used as a case study, where it is seen that UVM is capable of achieving high functional coverage with fewer regressions of the test on the given design. An algorithm for verification of SoCs based on UVM is described in [10], with reusable testbench schemes, consisting of 6 phases, that can be adopted for any SoC design. Overall, the existing work highlights the advantages of using UVM for design verification. Since SoCs are quite complex in their design, these verification concepts need to be extended and customised to implement on SoCs.

The following section describes the theoretical concepts which form the basis of design integration and verification. Section III discusses the design specifications and implementation methods. The results obtained are detailed in Section IV, along with a comparative analysis on existing work.

II. THEORETICAL CONCEPTS

A. Design Integration

Connectivity and integration can be performed only after having a good understanding on the various interfacing protocols, interfacing details, architecture of the chip , the design and port details of the IP Cores. A network interface protocol is a specification that defines how the data is delivered between the SoC cores. Additionally, it provides information about the structure, encoding, and the speed at which information or data is transferred. The main interfacing protocols used at the SoC level are AXI [11], AHB [12] and APB[13] protocols.

B. Front End SoC Integration using Defacto tool

The design of the IP Cores are sent by the design team along with their functionalities interfacing details, architecture, port details, direction and functions in the form of csv files. The specifications received from the core design owners should be understood and the connectivity has to be done accordingly by updating the .csv files. After updating the csv, a run is launched on the Defacto's SoC Compiler tool. The generated output directory is checked for unconnected inputs, errors, unloaded outputs and multi-driven ports. The number of unintended opens and tie-off are reduced by getting the connections reviewed by the core owners. Once the output files have no undriven inputs and multi driven ports, the top module generated by the tool can be compiled using sandbox for lint and clock domain crossing check and is given to the design verification team where all the IP cores are tested and verified for their functionality. Fig. 1 shows the integration methodology.

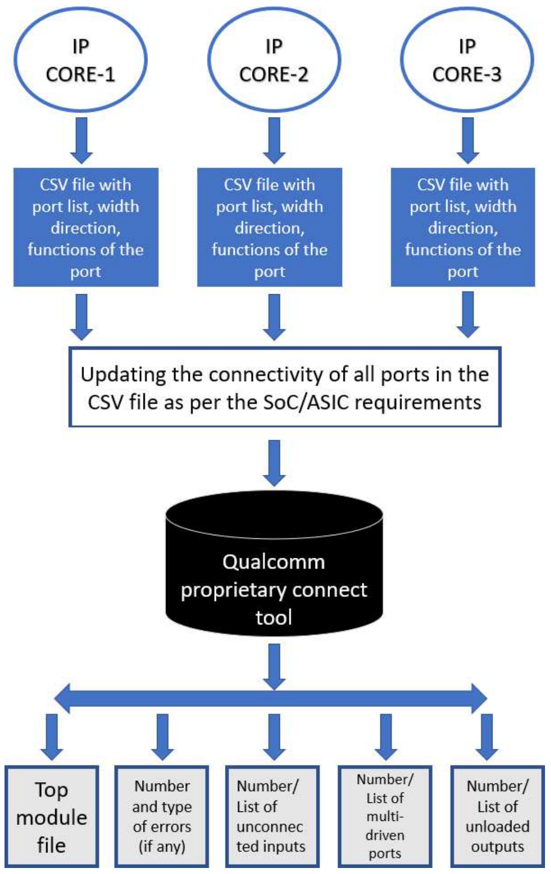

IP CORE-1
IP CORE-2
IP CORE-3
CSV file with port list, width direction, functions of the port
CSV file with port list, width direction, functions of the port
CSV file with port list, width direction, functions of the port
Updating the connectivity of all ports in the CSV file as per the SoC/ASIC requirements
Qualcomm proprietary connect tool
Top module file
Number and type of errors (if any)
Number/ List of unconnec ted inputs
Number/ List of multi-driven ports
Number/ List of unloaded outputs

Fig. 1. Methodology of SoC Integration

C. Design Verification

Design verification is one of the most crucial aspects of the product development process. The goal is to verify that the design meets the system specifications and requirements. In this regard, many verification languages and methodologies have been developed. The most efficient and widely used however, is the Universal Verification Methodology.[14]

Universal Verification Methodology(UVM) is the bench-mark used in digital design verification. It is developed using the concepts and syntax of the System Verilog language, and introduces the best practices for efficient design verification. One of its main features is the concept of re-usability, where UVM components are reused across all types of designs to produce an efficient testbench architecture.

UVM follows 3 main phases as listed below.[15]

1) *Build Phase* - It consists of the build, connect and end-of-elaboration sub-phases. The build sub-phase builds testbench components whereas connect sub-phase connects different testbench components. The end of elaboration displays UVM hierarchy and other functions required after connection.
2) *Run Phase* - Under the run phase, the actual simulation of Design Under Test and the running of the testbench takes place.
3) *Cleanup Phase* - It consists of the extract, check, report and final sub-phases. The expected data from scoreboard is extracted in the extract sub-phase. It checks for errors between expected and actual values in the check sub-phase. The report sub-phase displays result of the check and any last minute operations are done in the final sub-phase.

UVM architecture introduces the use of different components in a testbench to carry out specific tasks. These components are customised specific to the needs of the design under test. The specifications of this UVM architecture for verification at the SoC level are elaborated in the following section.

III. Design Methodology and Implementation

A. Design and Interfacing details of IP cores

1) Interface Controller Block: The main function of interface controller block is to provide interface for slower peripheral devices like flash devices and RAM. The block has a bus manager unit, a decoder block, several blocks of register, performance monitor, and RAM. The interfacing

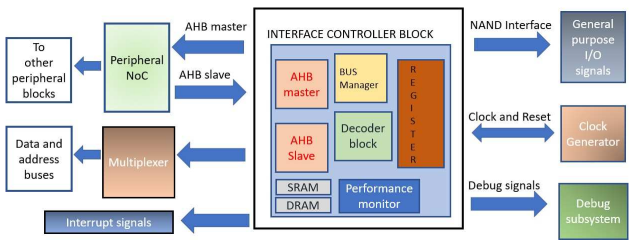


Fig. 2. Design and Interfacing details of Interface Controller Block

details are shown in Fig. 2. The interface between interface controller to the peripheral NoC is AHB master interface and from NoC to interface block it is a AHB slave interface. It is connected to general purpose input-output using a NAND interface. The block has clock and reset signals which come from the global clock controller block. The block also has a set of power related signals which are connected to the always-on subsystem. Majority of the interrupt signals are connected to the modem block. The external data and address busses are sent to the top level status register block. Apart from the above connectivity, there are also memory related pins which are connected to the global memory block.

2) PCIe block: Peripheral Component Interconnect Express (PCIe) is a bus standard which is used to connect a central master to peripheral subsystems such as network interface cards, using serial expansion. The advantage of PCIe is its reduced latency and greater data transfer rates.

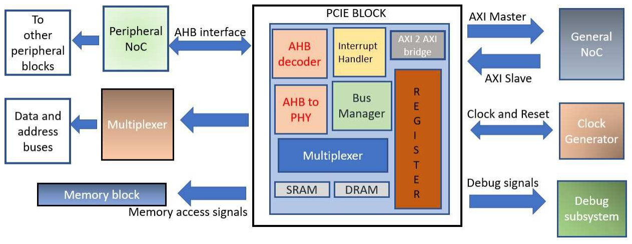


Fig. 3. Design and Interfacing details of PCIe Block

The block diagram of PCIe is as shown in Fig 3. PCIe communicates with peripheral NoC using an AHB interface. Most pins in PCIe are memory related and are connected to the global memory block. As PCIe block helps in interfacing other blocks, it sends out a set of data and control signals to the top level multiplexer which are then sent to various other cores. Like other cores, PCIe receives a set of clock and reset from the top level clock controller block. PCIe is interfaced to the general NoC through an AXI master slave interface. It has its own set of debug signals which come from global debug subsystem. It also has a set of interrupt signals which go to modem block and other IP blocks as well.

3) Ethernet Block: The Ethernet block is represented in Fig. 4. It has ports related to three front hauls which are connected to the Common Public Radio Interface(CPRI) block through a MACSEC to UDP protocol and a Chip-to-Chip (C2C) interface. The block is connected to peripheral NoC through an AHB interface. The modem is interfaced to ethernet block using a MACSEC interface. The block has a series of power signals, debug signals, memory and clock signals which are connected to the corresponding blocks.

4) Throttle and Pipeline blocks: Pipeline blocks are used between two IP cores which are to be integrated but are placed further apart in a SoC chip. Pipelining process quickens the processor operation by executing many instructions which can be performed on the same hardware by interleaving. Throttling is a power reduction procedure in which the frequency of a processor is dynamically adjusted depending on the needs of the peripherals, in order to manage power and reduce the generation of heat.The interfacing details of a throttle block between NoC and the debug system are shown in Fig. 6

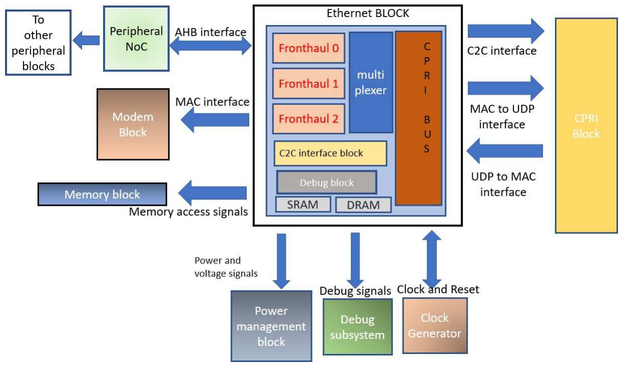


Fig. 4. Design and Interfacing details of Ethernet Block

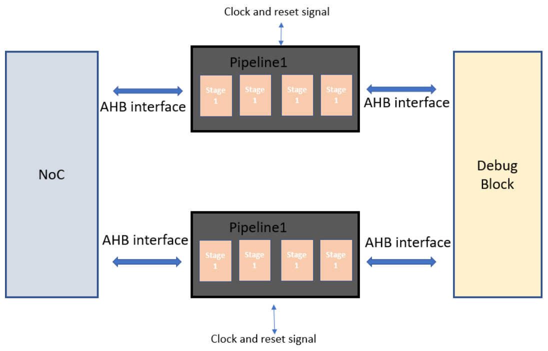


Fig. 5. Design and Interfacing details of pipelines

B. Design of testbench for SoC using UVM

SoC verification verifies the integration of components as described in the above subsections. This is done using the UVM methodology. A UVM testbench consists of various components, which are implemented using the base classes of the UVM Library.

The UVM test is the topmost component of the UVM based simulation. It constructs a testbench scenario and schedules the execution of the lower-level components. It contains the environment, which is essentially a container class for agents. An agent is also a container class, that encapsulates three components - the sequencer, driver and monitor. The sequencer

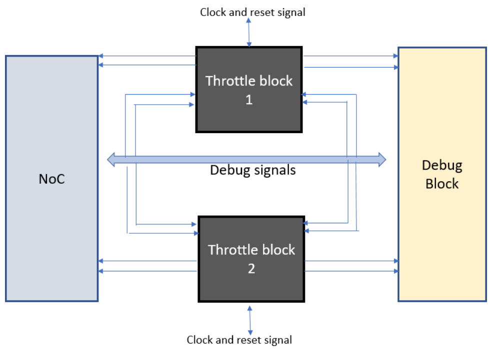


Fig. 6. Design and Interfacing details of Throttle Block

is responsible for generating the stimuli that are input to the design under test. The driver is the UVM component which picks up the sequence items and forwards them to the interface of the design. The UVM Monitor takes the output signals from the DUT and sends them to the scoreboard component for further evaluation. The scoreboard contains the required output and compares it to the output that it receives from the monitor, and declares a pass/fail on the test.

In order to customise this design for a system-on-chip, these testbench components have to be instantiated for each IP core present in the SoC. This customised testbench design is represented in Fig. 7 Each IP core forms a verification

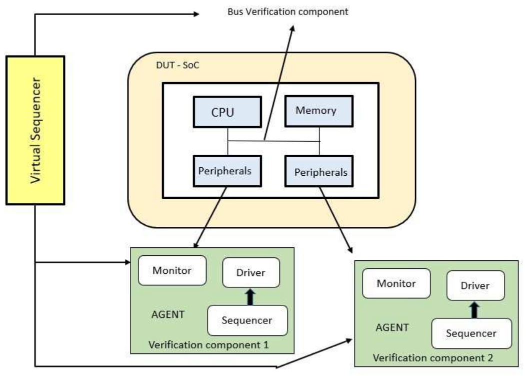


Fig. 7. Customised UVM Testbench for SoC

component, containing the monitor, driver and sequencer. A virtual sequencer is added to ensure that the individual sequencers of all verification components produce stimuli that are in sync with each other. Additionally, SoC level verification requires instantiation of all verification components with the specific test stimulus applied to the block under test.

C. Implementation using UVM class library

The UVM class library is used to build testbenches following UVM methodology. It contains foundation classes, on which user-defined testbench components can be built, using the concepts of inheritance and class objects. The various components and classes of the UVM testbench are shown in Fig. 8

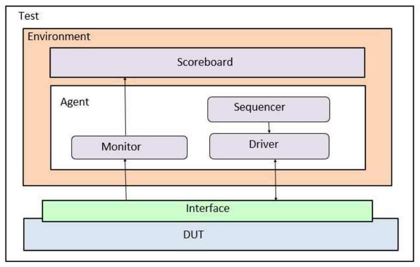
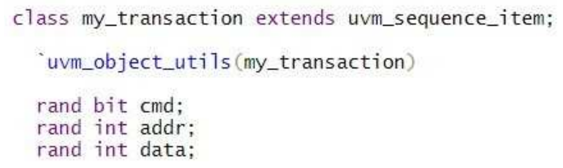


Fig. 8. UVM Testbench components

1) Modelling a sequence item: A sequence item is series of data items that are generated randomly using the concept of constrained randomization present in System Verilog. These sequence items are provided as an input to the design under test. A base class, known as uvm_sequence_item is present in the UVM library. Sequence item classes are built using the methods of this base class, through inheritance. The code snippet used to define the sequence item is shown in Fig. 9.


```
class my_transaction extends uvm_sequence_item;

  `uvm_object_utils(my_transaction)

  rand bit cmd;
  rand int addr;
  rand int data;
```

Fig. 9. Modelling a sequence item

The class my_transaction is defined by extending the uvm_sequence_item class which is present in the UVM library. The data members shown here include the basic memory access variables - an address and data variable, along with a command bit.

2) Building driver and monitor: The driver classes are built by inheriting the features of the uvm_driver base class. Similarly to create a monitor component, a custom class inherited from uvm_monitor is created. The definitions follow the same syntax as presented in Fig. 9. The data items and methods are defined according to the requirements of the design and tests. The sequencer object is passed as a parameter to the driver.

3) Building container and top classes: The agent, environment and top class of the testbench are essentially container classes, containing the lower sub-components defined above. These are defined by extending the respective base classes, uvm_component and uvm_top from the UVM library, following the syntax of Fig. 9. The important feature of these container classes are that the sub-components are built or instantiated in the build phase of these container classes.

The implementation of UVM testbenches was applied to the specific blocks of the SoC, in order to verify its functionalities. An SoC level verification required the verification of the functionalities of the particular block/IP core along with their interaction with the other IP cores present in the SoC. In order to achieve these, a test plan was formulated according to the design of the SoC. The following features were decided to be verified in the testplan:

- Functionalities of IP cores.
- Read-write transactions between the processors and IP cores.
- Memory and register access by the IP cores.

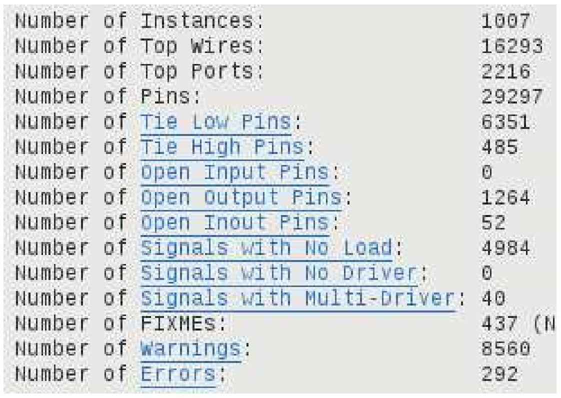
These tests are developed by writing UVM based code, following the steps detailed above. The expected outputs of the tests based on the design specifications are stored in the UVM scoreboard. The tests are then run on the SoC under test, and pass/fail is declared by the UVM scoreboard upon comparison with expected and received outputs.

IV. Results and Discussion

The concepts of SoC integration and design verification discussed so far necessitate a careful and detailed implementation on the given SoC design. The results obtained on implementation are detailed in the sections below.

A. Simulation logs of the connectivity tool

The connectivity was performed using Qualcomm proprietary tools. Fig. 10 shows the report with the number of open input and output ports which is obtained after the connect run is completed. The number of hanging inputs and unidirectional multidriven ports are to be reduced to zero as they can cause errors during the sandbox simulation, lint and clock domain crossing checks. Table I shows the list of errors during the



```
Number of Instances:                   1007
Number of Top Wires:                   16293
Number of Top Ports:                   2216
Number of Pins:                        29297
Number of Tie Low Pins:                6351
Number of Tie High Pins:               485
Number of Open Input Pins:             0
Number of Open Output Pins:            1264
Number of Open Inout Pins:             52
Number of Signals with No Load:        4984
Number of Signals with No Driver:      0
Number of Signals with Multi-Driver:   40
Number of FIXMEs:                      437 (N
Number of Warnings:                    8560
Number of Errors:                      292
```

Fig. 10. Report of unconnected inputs and outputs

entire timeline of the project. The table shows the number of errors and unconnected ports during the start every phase. These were caused mainly due to the addition or deletion of IP core wrappers as per the ASIC/SoC requirements. Phase-1 mainly included completing address and data ports for information transfer among blocks via NoC. Phase-2 mainly included connecting memory related pins,adding pipelines and throttle blocks as per tiling specifications.Phase-3 included connecting interrupt signals, debug, Design for Test(DFT) signals.

TABLE I
Number of errors and unconnected ports during the beginning of different phases

	Phase-1	Phase-2	Phase-3	Phase-4
Error	1583	372	0	0
Unconnected ports	900	730	64	23

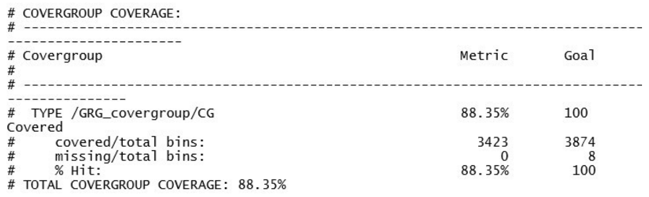
B. Results of UVM test runs

The tests developed as described in Section IV were run against the SoC design under test. Since SoC level verification validates the integration of the blocks, the number of tests passed progressively improves as the number of errors and unconnected ports were reduced. Since the number of tests required to verify the entire SoC is huge, the results concentrate on the tests for the security-related blocks of the SoC. The pass/fail status for these tests were as shown in Table II

TABLE II
Pass/fail status of UVM test runs on the SoC

	Run-1	Run-2	Run-3
Total no. of tests	40	40	40
No. passed	17	29	40
No. failed	23	11	0

The debug was correspondingly performed for the failed tests after each run. The reason for failures were found to be errors in the UVM environment or in the communication interface between blocks, due to which the processor requests were not reaching the IP blocks as expected. The error was rectified for the subsequent rounds of testing, to finally achieve 100% pass on all tests. One of the most important features of UVM is its code and functional coverage. Functional coverage is essential to validate how many features/ functionalities have been verified by the generated and developed tests. The total functional coverage for the developed tests were as shown in Fig. 11. As seen, the functional coverage achieved (88.35%)is



```
# COVERGROUP COVERAGE:
# --------------------------------------------------------------------------------
----------------------
# Covergroup                                          Metric      Goal
#
# --------------------------------------------------------------------------------
--------------
#  TYPE /GRG_covergroup/CG                            88.35%      100
Covered
#     covered/total bins:                               3423      3874
#     missing/total bins:                                  0         8
#     % Hit:                                          88.35%       100
# TOTAL COVERGROUP COVERAGE: 88.35%
```

Fig. 11. Functional coverage achieved

greater as compared to other simple verification methodologies using only System Verilog, and not UVM [16]. The number of features to be verified is also important, as functional coverage validates the features that are verified. The tests developed here included crucial features of core functionalities as well as core-to-core access, as explained in the methodology, and hence, were capable of generating high functional coverage, even compared to other UVM tests[17]. Hence, close to 90% of SoC level features were validated and verified using the developed UVM tests.

V. Conclusion

The concepts of integration and design verification are an indispensable part of modern day SoC development. The main focus of this paper is the integration of the IP cores and its verification. the process of integration requires extensive knowledge on the interfacing details, design, port details, port functionalities, interfacing protocols and the entire SoC architecture.The verification is performed by developing test benches using the Universal Verification Methodology, and further carrying out debug for any failed tests. After performing the connectivity of above blocks, the errors were zero, the number of unconnected inputs were null, the number of unidirectional multi-driven ports was zero, which confirms the quality of integration. On running the UVM testbenches on the given SoC design, and carrying out debug on the failed tests, a 100% pass was achieved on all the designed tests. The functional coverage was found to be around 88.35%, which is a significant improvement compared to tests which do not use UVM methodology. The described work was tested only on the mentioned blocks and designs. This can be extended by progressing on to SoCs with greater number of IP cores, and implementing the discussed concepts of integration and verification onto different SoC designs.

REFERENCES

[1] M. S. BenSaleh, S. M. Qasim, A. A. AlJuffri and A. M. Oheid, ”Design of an Advanced System-on-Chip Architecture for Internet-Enabled Smart Mobile Devices,” 2018 30th International Conference on Microelectronics (ICM), 2018, pp. 323-326, doi: 10.1109/ICM.2018.8704077.

[2] Y. J. Yoon, P. Mantovani and L. P. Carloni, ”System-level design of networks-on-chip for heterogeneous systems-on-chip,” 2017 Eleventh IEEE/ACM International Symposium on Networks-on-Chip (NOCS), 2017, pp. 1-6.

[3] P. Ghosh and S. Rohit, ”Case Study: SoC Performance Verification and Static Verification of RTL Parameters,” 2019 20th International Workshop on Microprocessor/SoC Test, Security and Verification (MTV), 2019, pp. 65-72, doi: 10.1109/MTV48867.2019.00021.

[4] J. Wang, N. Tan, Y. Zhou, T. Li and J. Xia, ”A UVM Verification Platform for RISC-V SoC from Module to System Level,” 2020 IEEE 5th International Conference on Integrated Circuits and Microsystems (ICICM), 2020, pp. 242-246, doi: 10.1109/ICICM50929.2020.9292250.

[5] H. Saafan, M. W. El-Kharashi and A. Salem, ”SoC connectivity specification extraction using incomplete RTL design: An approach for Formal connectivity Verification,” 2016 11th International Design and Test Symposium (IDT), 2016, pp. 110-114, doi: 10.1109/IDT.2016.7843024.

[6] Ma, D., Huang, K., Xiu, S.W., Yan, X.L., Feng, J., Zeng, J.L., Ge, H.T., 2011. An Automatic SoC Design Methodology for Integration and Verification. AMR 383–390, 2222–2230. https://doi.org/10.4028/www.scientific.net/amr.383-390.2222

[7] A. Hussien, ”Development of a Generic and a Reconfigurable UVM-Based Verification Environment for SoC Buses,”International Journal on Microelectronics, 2019

[8] Renuka, Gaddam and Ushashree, V and Reddy, P and Shree, V.Usha, Functional Verification of Complex SoC by Advanced Verification Methodology. , International Journal of VLSI System Design, 2020

[9] K. Salah, ”A UVM-based smart functional verification platform: Concepts, pros, cons, and opportunities,” 2014 9th International Design and Test Symposium (IDT), 2014, pp. 94-99

[10] P. Ghosh, S. Ghosh, P. Singh and S. Mishra, “Case study: Re-visiting SoC verification challenges and best practices”, International Journal of VLSI Design and Test, 2018

[11] N. Gaikwad and V. N. Patil, ”Verification of AMBA AXI On-Chip Communication Protocol,” 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), 2018, pp. 1-5, doi: 10.1109/ICCUBEA.2018.8697587.

[12] L. Deeksha and B. R. Shivakumar, ”Effective Design and Implementation of AMBA AHB Bus Protocol using Verilog,” 2019 International Conference on Intelligent Sustainable Systems (ICISS), 2019, pp. 1-5, doi: 10.1109/ISS1.2019.8907975.

[13] P. Jain and S. Rao, ”Design and Verification of Advanced Microcontroller Bus Architecture-Advanced Peripheral Bus (AMBA-APB) Protocol,” 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021, pp. 462-467, doi: 10.1109/ICICV50876.2021.9388549.

[14] N. B. Harshitha, Y. G. Praveen Kumar and M. Z. Kurian, ”An Introduction to Universal Verification Methodology for the digital design of Integrated circuits (IC’s): A Review,” 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021, pp. 1710-1713, doi: 10.1109/ICAIS50930.2021.9396034.

[15] Ashok B. Mehta. 2017. ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies (1st. ed.). Springer Publishing Company, Incorporated.

[16] M. P. Deepu and R. Dhanabal, ”Validation of transactions in AXI protocol using system verilog,” 2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS), 2017, pp. 1-4, doi: 10.1109/ICMDCS.2017.8211605.

[17] K. S. Pooja, S. Krishnakumar and H.V.R.Aradhya, ”Verification of Interconnection IP for Automobile Applications using System Verilog and UVM,” 2018 3rd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology (RTEICT), 2018, pp. 1119-1123, doi: 10.1109/RTEICT42901.2018.9012309.