

Digital Intellectual Property Verification using Universal Verification Methodology

Wafa Naaz Shaik
R V College of Engineering
Bengaluru

Nandu Saseendran
Staff DV Engineer
Analog Devices

Dr. H V Ravish Aradhya
R V College of Engineering
Bengaluru

Abstract—With the increase in complexity of System-on-Chip (SoC) designs to enable greater logic integration driven by Moore's law and beyond Moore principles, the process of verification of the design becomes increasingly complex. To facilitate faster and more efficient verification, a standard methodology like the Universal Verification Methodology (UVM) is essential. UVM provides a library of SystemVerilog (SV) base classes, which can be extended to meet specific requirements. Verification is a crucial component in chip design and fabrication process, necessitating thorough testing at both the block and full-chip levels. This paper focuses on the block-level verification of an IP, General Purpose Input/Output (GPIO) Controller. Utilizing the UVM architecture, a constrained random testbench has been developed to create a reusable and robust verification environment. The testbench supports multiple tests to ensure comprehensive design coverage, aiming for 100% code coverage. Advanced SystemVerilog features, including assertions, coverpoints, covergroups, and multithreading, are integrated into the testbench to enhance its effectiveness.

Index Terms—Intellectual Property, Verification, UVM, SystemVerilog, General Purpose I/O controller, code coverage

I. INTRODUCTION

Digital Intellectual Property (IP) blocks allow for a standardized mechanism to tame the complexity of large System-on-Chip (SoCs) development, while simultaneously improving time-to-market. Digital IP blocks are critical to the success of many product engineering teams, considering the breadth of the product mix. As part of the IP curation process, different flavors of IP blocks are consolidated, curated, and containerized for ready use in other products.

Digital IPs are developed by multiple groups, even within an organization, with variations in functionality. Often, a group redevelops a particular IP to suit its needs even when that IP is available with minor variations or differences. This overhead of design and verification could be saved if a repository with standard but flexible IPs exists and the collaterals maintained in such a way that integration is easy at every stage in the product development process.

Verification of digital IPs using Universal Verification Methodology (UVM) is a critical process in ensuring the functionality, reliability, and compatibility of IP cores integrated into larger digital systems. Ensuring the correctness of these IP blocks poses significant challenges due to their intricate interactions within the system. UVM is a standardized verification methodology, in validating digital IPs. From establishing

comprehensive test environments to implementing advanced verification techniques, UVM provides a structured framework for verifying digital IPs, ultimately contributing to the quality and integrity of modern digital designs.

General Purpose Input/Output (GPIO) Controllers serve as the interface between software and electronic components, facilitating communication and control within a system. GPIO Controllers are standard blocks in almost all electronic products, since they provide a generic mechanism for software control. Hence, verifying the performance and integrity of GPIO controllers is a critical step in the design process, ensuring that they meet specifications, operate seamlessly with other system components, and maintain stability under various conditions.

II. LITERATURE REVIEW

The IEEE Standard for UVM Language Reference Manual [1] and the IEEE Standard for SV Unified Hardware Design, Specification, and Verification Language [2] provide comprehensive guidelines and specifications that underpin modern hardware design and verification processes. The UVM standard, as detailed in IEEE Std 1800.2-2020 [1], outlines a methodology for verifying integrated circuit designs, emphasizing the use of reusable and scalable verification components. This standard facilitates the creation of consistent and interoperable verification environments, which are essential for the efficient validation of complex digital systems. On the other hand, the SystemVerilog standard [2] offers an extensive language framework that integrates design, specification, and verification capabilities. It serves as a unified platform that enhances the expressive power of hardware description languages, enabling more precise and efficient design and verification workflows. Together, these standards provide a robust foundation for the development and verification of sophisticated hardware systems, promoting industry-wide best practices and interoperability.

Paper [3] speaks about how an integrated digital system can be perceived as a network of interlinked IPs, where the overall reliability hinges on the functional verification of these IPs. This process is crucial because these IPs are often reused across various SoCs. Given that different companies supply these IPs, each employing distinct validation processes, a standardized methodology for evaluating IPs was established.

This framework utilizes mutation-based metrics for functional verification and implements an evaluation strategy designed to create sequences aimed at detecting mutants.

Addressing the escalating demands of verification, recent studies and implementations have demonstrated the efficacy of UVM in real-world applications. For instance, the verification of an interconnection soft Intellectual Property (IP) used in automotive applications showcases the practical benefits of UVM. This IP, characterized by high programmability and parameterization, supports a wide range of products and functionalities. Through UVM, both code coverage and functional coverage metrics are efficiently achieved, underscoring the methodology's capability to handle intricate verification tasks effectively. Such implementations affirm the pivotal role of UVM and SystemVerilog standards in advancing the verification landscape, ensuring reliable and efficient validation of increasingly complex SoC designs. [4]

Paper [5] discusses how SoC design requires thorough verification at block and full chip levels. UVM facilitates reusable testbenches, while constrained random testing diversifies scenarios. Coverage-driven methodologies ensure comprehensive validation, augmented by SystemVerilog features for efficiency. This integrated approach is essential for robust SoC design verification.

In the design of SoC systems, random testing is increasingly being applied for the verification of IP cores. To verify the integrated External Memory Interfaces IP core, a restricted random verification method is employed. This method incorporates the flexible generation of parameterized script files and adaptable random test points. The benefits of the restricted random method are twofold: it can uncover design blind spots and test points that are difficult to cover during the design process, and it provides flexible and maintainable test vectors and test points, thereby enhancing the overall efficiency of the verification process. [6].

Reducing the overall chip design and verification time, and consequently the cost, can be significantly aided by any savings in this component. Automation plays a key role in achieving these savings. The largely canonical and regular structures and behavior of most Design For Testability (DFT) IPs make this feasible. As a result, there has been a notable shift towards standardized configurable DFT logic architectures. [7]. In paper [8], an automated approach has been developed to extract hardware memory-mapped register names directly from the Register Transfer Level (RTL) implementation when this information is missing from the register specification document.

Detecting functional conditions or multiplexer selects in large-scale designs is challenging. For the verification engineer to benefit, the reported condition signal should be the parent signal, positioned near the top of the hierarchy with minimal combinational depth relative to the primary inputs. Additionally, maintaining the generated connectivity specification becomes difficult as the SoC design evolves during development and as low power and test circuitry are inserted. [9].

For small, protocol-based designs such as an encoder, a formal verification approach offers significant advantages. Unlike simulation verification, which requires an environment setup before yielding results, formal verification can provide outcomes from the initial stages. Additionally, assertion-based verification is an effective method for verifying parts of incomplete modules. [10].

An effective methodology for the formal verification of serial protocol bridges, an area where simple approaches often fail, has been introduced. This method employs a layered modeling of properties and a functional partitioning of the design. Initially, lightweight simulation is used to bring the system to known states, after which formal verification is applied. This methodology has been successfully demonstrated on two industrial designs of serial protocol bridges, achieving verification goals where standalone formal verification and simulation had previously failed. [12].

A successful implementation of Joint Test Action Group (JTAG) verification, driven by functional coverage and based on the UVM methodology, has been achieved using SystemVerilog. The testbench architecture was designed following UVM guidelines. Approximately ten test cases were written and simulated using Questasim to validate the JTAG functionality. The results indicated 94.06 percent code coverage, 100 percent functional coverage, and 96.45 percent cross coverage. Achieving 100 percent functional coverage signifies that all covergroups were thoroughly covered, ensuring the effective verification of JTAG functionality. [13].

Paper [14] presents the design and verification of the AMBA-based Advanced Peripheral Bus (APB). The verification environment, constructed using SystemVerilog, includes features such as functional coverage, assertion coverage, constrained randomization, and code coverage. Functional coverage ensures that the HDL code performs all specified functionalities, while code coverage confirms that test cases thoroughly exercise the entire RTL code. Assertions act as additional checkers to prevent protocol violations. The APB 4.0 protocol was fully designed and verified, with no data loss detected in the system. AMBA's APB is ideal for low power, low-cost interfacing between high-speed and low-speed systems.

Functional coverage metrics are essential for managing verification efforts and assessing design quality. E-verification environments can be applied at various stages of the design cycle, from high-level architectural models to complete systems. The IEEE Standard 1647-2016 provides a comprehensive definition of the e-language syntax and semantics, offering guidance for tool developers and verification engineers on its proper use. [15].

The functional coverage collection infrastructure proposed in paper [16] offers seamless reusability across projects and blocks within the same project. By adopting this infrastructure, efforts to maintain coverage information are significantly reduced. The direct updating of coverage details into a CSV file provides a convenient and time-saving approach to coverage management, thereby boosting verification productivity and

enabling earlier time-to-market.

III. APPROACH FOR VERIFICATION USING UVM

A. Testbench Top

In UVM, a testbench top is the highest hierarchical level of the testbench. It instantiates and connects all the components required for the verification environment. As shown in Figure 1, the testbench top of the GPIO Controller includes the test environment, the design under test (DUT), and the interface connections between them. The testbench top module (`gpio_top`) instantiates the GPIO controller (`dut`) and connects it to a virtual interface (`gpio_vif`). The test environment (`gpio_env`) is also instantiated and linked to the same interface. The `run_test` method initiates the UVM test, which will drive the GPIO controller and monitor its outputs based on the configured sequences and coverage models.

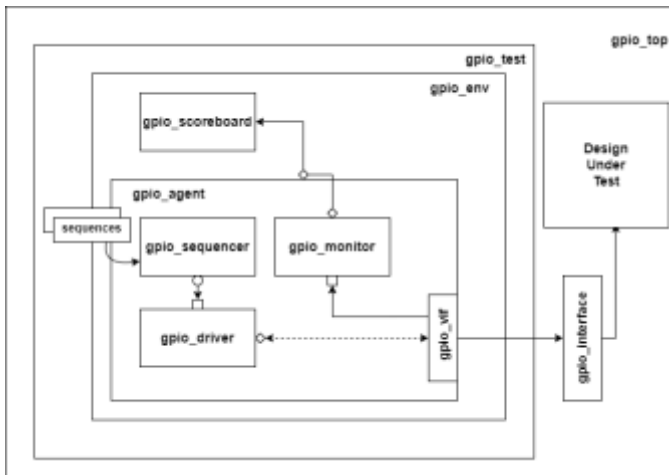


Fig. 1. Testbench Architecture

B. Environment Class

The environment class is responsible for instantiating and configuring all the lower-level components required for verification. This includes the agent (`gpio_agent`, which contains drivers, sequencers, and monitors) and the scoreboard (`gpio_scoreboard`). The `gpio_env` class sets up configurations and passes them down to other components to control the behavior of agents and monitors.

C. Wrappers

A wrapper is used to encapsulate a module or component to provide an interface between the design under test (DUT) and the UVM verification environment. A typical UVM wrapper includes a virtual interface (`gpio_vif`), instantiation of the DUT and the connection logic.

D. Agent

The agent is a crucial element that consolidates the components necessary for driving and monitoring a specific protocol, in this case, the Advanced Peripheral Bus (APB) interface. By bundling the agent's elements, including the sequencer, driver,

and monitor, the agent class improves the modularity of the testbench, making it easier to reuse and maintain. The agent facilitates communication between the test sequences and the DUT by converting high-level sequence items into low-level transactions through the driver and observing DUT behavior via the monitor.

E. Register Model

The GPIO Controller design includes several registers for reading and writing values. These include the pad registers, the input and output registers, as well as registers to set, clear, toggle the values in the pad registers.

To verify the full functionality of these registers, they are replicated in the testbench. This replication is achieved using the Register Abstraction Layer (RAL) model (`gpio_reg_model`), which models the DUT's registers based on their configurations, such as width, read-write, write-only, read-only, default values, and other attributes. These registers are modeled by extending the `uvm_reg` class. RAL abstracts the register operations, providing a high-level interface for interacting with the DUT's registers. This includes reading from and writing to the registers without needing to deal with low-level signal manipulations directly.

F. Reference Model And Scoreboard

In verification, a reference model is a critical component used to predetermine the expected behavior of the Design Under Test (DUT). It acts as a golden standard for comparing the actual behavior of the DUT to ensure its correctness. This model supports automated regression testing by offering a reliable basis for comparison across various test scenarios.

The scoreboard, `gpio_scoreboard` acts as a vital component overseeing the interaction between a DUT and its test environment, ensuring the accuracy of signal exchanges. Comprising of monitors and checkers, it functions like a supervisor. Monitors observe specific signals within the DUT, while checkers analyze this data against expected values or behaviors provided by tests. The scoreboard coordinates these activities, serving as a centralized mechanism for verifying the DUT's behavior according to test specifications. Essentially, it flags discrepancies and potential errors, thereby verifying the integrity of the DUT and the efficacy of the tests.

IV. DESIGN AND IMPLEMENTATION

A. Verification Flow

UVM has gained significant popularity due to its widespread support by major EDA tools. The verification methodology utilized for verifying the GPIO Controller is depicted in Fig. 2. Constructing a UVM verification environment entails designing various testbench components such as monitors, drivers, scoreboards, checkers, and sequencers. Sequences play a crucial role by providing the stimulus. They are invoked during tests. These tests are conducted separately and verified for accuracy through waveform analysis. Upon preparing an exhaustive test list, a comprehensive regression is executed to derive coverage metrics.

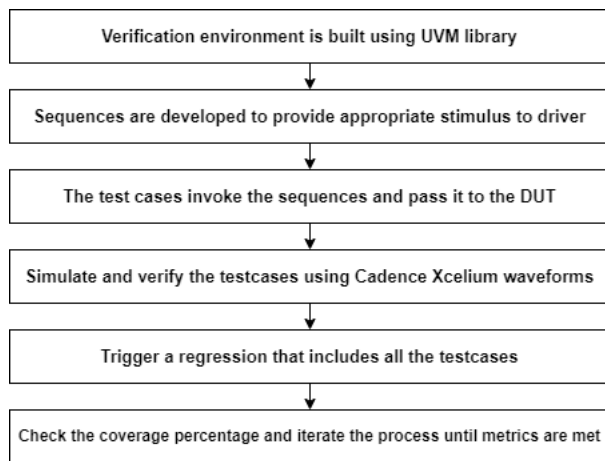


Fig. 2. Verification Flowchart

B. Testbench Parameters

Within the testbench, the parameters utilized across different agents are encapsulated within the `gpio_env_pkg`. These parameters are organized as key-value pairs within the `uvm_config_db` and are managed through the `set()` and `get()` methods of UVM.

C. Virtual Interfaces

In the GPIO verification environment, various virtual interfaces are employed to connect the DUT to the verification testbench. These interfaces are instantiated and configured within the testbench hierarchy. Handles to these virtual interfaces are established in the top module and stored in the `uvm_config_db`, a centralized configuration database in UVM. After being initialized, these handles cascade down the testbench hierarchy until they reach the driver and monitor components. Within these components, the handles are utilized to establish pin-level connections with the DUT. This enables the seamless exchange of signals and data, facilitating comprehensive verification. This hierarchical approach ensures proper communication between the testbench and the DUT, facilitating comprehensive testing and validation of the design.

D. APB Interface

For verification purposes, the GPIO is controlled by an APB Interface Verification IP (VIP) as shown in Figure 3. The utilization of the APB VIP within the design necessitates the presence of an APB agent responsible for managing transactions to manipulate the configuration and registers of the GPIO Controller.

E. Randomization Strategy

Randomization plays a pivotal role in enhancing testing methodologies. To generate diverse transaction scenarios, the sequence of register writing and the intervals between them undergo randomization. Furthermore, weightage is assigned to variable ranges, giving precedence to crucial parameters and those with a greater number of branches.

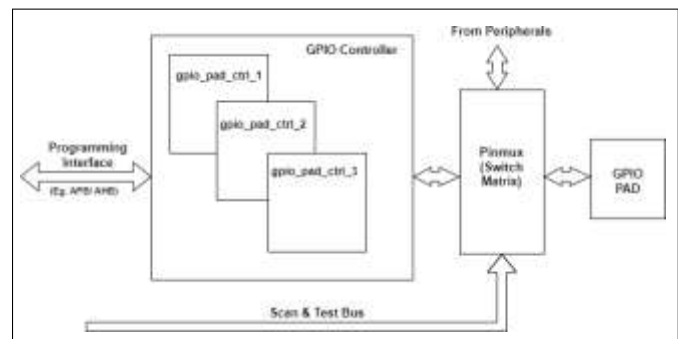


Fig. 3. GPIO Controller Integration

Furthermore, the random data transmission adheres to specific constraints, enhancing the meaningfulness of the generated random data.

V. RESULTS AND DISCUSSIONS

The tests (as seen in Figure 4) were executed collectively as part of a regression run, running concurrently to expedite the process of simulation. Subsequent to running multiple tests, a coverage report was generated using vManager, revealing code coverage percentage. Full code coverage couldn't be obtained, as anticipated. This discrepancy stemmed from an unreachable segment within the DUT, limiting code coverage to approximately eighty percent as shown in Figure 5. Coverage is not yet closed and it is still a work in progress.

Index	Name	Status
(no filter)	(no filter)	
1	gpio_reg_hw_reset_test	passed
2	gpio_reg_bit_bash_test	passed
3	gpio_pad_reg_test	passed
4	gpio_pslverr_test	passed
5	gpio_simple_test	passed

Interfaces	Code coverage
gpio_sim_top	69.5 %
clock_agent_interface	50 %
reset_agent_interface	41.56 %
apb_agent_interface	80.77 %
gpio_agent_interface	30.94 %
apb_interface	61.95 %
gpio_interface [7:0]	95 %
dut	81.33%

Fig. 5. Code Coverage Report

VI. CONCLUSION

In conclusion, the verification of GPIO controller using UVM has been a comprehensive and insightful process. Through the meticulous application of UVM methodologies, it

has become possible to achieve a robust verification environment that ensures the functionality, reliability, and compliance of this component in digital systems.

By leveraging UVM's powerful features, effective validation of the GPIO controller against its specifications has been achieved. This verification process has not only detected numerous bugs and corner cases but has also enhanced the understanding of intricacies of the interfaces.

Furthermore, the modular and reusable nature of UVM components has facilitated seamless integration with different test scenarios and environments, thereby streamlining the verification process and improving productivity.

In summary, the utilization of UVM for the verification of GPIO controller has not only ensured the correctness and robustness of these components but has also demonstrated the effectiveness of adopting advanced verification methodologies in modern digital design projects.

VII. ACKNOWLEDGEMENT

Wafa Naaz Shaik thanks Nandu Saseendran from Analog Devices Inc. for his constant support and guidance in verifying the DUT.

VIII. REFERENCES

The references provided herein serve as essential sources of information and scholarly contributions that have significantly enriched the content of the paper. Each reference encapsulates a wealth of knowledge, insights, and findings related to various facets involved in the understanding of verification using UVM and its implementation on specific IPs, such as the GPIO.

REFERENCES

- [1] IEEE Std 1800.2-2020 (Revision of IEEE Std 1800.2-2017), *IEEE Standard for Universal Verification Methodology Language Reference Manual*, pp.1-458, 14 Sept. 2020, doi: 10.1109/IEEESTD.2020.9195920.
- [2] IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012), *IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language*, pp.1-1315, 22 Feb. 2018, doi: 10.1109/IEEESTD.2018.8299595.
- [3] Y. Serrestou, V. Beroulle and C. Robach *Functional Verification of RTL Designs driven by Mutation Testing metrics*, 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools (DSD 2007), Lubeck, Germany, 2007, pp. 222-227, doi: 10.1109/DSD.2007.4341472.
- [4] 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), Bangalore, India, 2018, pp. 1119-1123, doi: 10.1109/RTEICT42901.2018.9012309.
- [5] D. Yashas, P. S. Hari Babu and N. Shylashree *UVM-based Logic Verification of Input Output Interface*, 2019 4th International Conference on Recent Trends on Electronics, Information, Communication and Technology (RTEICT), Bangalore, India, 2019, pp. 420-423, doi: 10.1109/RTEICT46194.2019.9016934.
- [6] Qingdong Meng, Zhaolin Li and Fang Wang *Functional verification of external memory interface IP core based on restricted random testbench*, 2010 2nd International Conference on Computer Engineering and Technology, Chengdu, China, 2010, pp. V7-253-V7-257, doi: 10.1109/ICCET.2010.5485235.
- [7] L. Sen, A. Roy, S. Bhattacharjee, B. Mitra and S. K. Roy *DFT logic verification through property based formal methods — SOC to IP*, Formal Methods in Computer Aided Design, Lugano, Switzerland, 2010, pp. 33.
- [8] 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 Test Symposium (IDT), Hammamet, Tunisia, doi: 10.1109/IDT.2016.7843024, pp. 110-114, 2016.
- [9] H. Saafan, M. W. El-Kharashi and A. Salem *Formal Based Methodology for Inferring Memory Mapped Registers*, 2016 17th International Workshop on Microprocessor and SOC Test and Verification (MTV), Austin, TX, USA doi: 10.1109/MTV.2016.11., pp. 15-18, 2016.
- [10] M. Girish, G. Gopakumar and D. S. Divya *Formal and Simulation Verification: Comparing and Contrasting the two Verification Approaches*, 2021 2nd International Conference on Advances in Computing, Communication, Embedded and Secure Systems (ACCESS), Ernakulam, India, doi: 10.1109/ACCESS51619.2021.9563305., pp. 41-44, 2021.
- [11] C. Flavio, P. Alberto, R. Barbara and F. Damiano *An ECLIPSE Plug-In for Formal Verification of BPMN Processes*, 2010 Third International Conference on Communication Theory, Reliability, and Quality of Service, Athens, Greece, doi: 10.1109/CTRQ.2010.32., pp. 144-149, 2010.
- [12] S. Gorai, S. Biswas, L. Bhatia, P. Tiwari and R. S. Mitra *Directed-simulation assisted formal verification of serial protocol and bridge*, 2006 43rd ACM/IEEE Design Automation Conference, San Francisco, CA, USA doi: 10.1145/1146909.1147096., pp. 731-736, 2006.
- [13] C. Elakkiya, N. S. Murty, C. Babu and G. Jalan, *Functional Coverage - Driven UVM Based JTAG Verification*, 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Coimbatore, India, doi: 10.1109/ICCIC.2017.8524556, pp. 1-7, 2017.
- [14] P. Dwivedi, N. Mishra and A. Singh-Rajput *Assertion and Functional Coverage Driven Verification of AMBA Advanced Peripheral Bus Protocol Using System Verilog*, doi: 10.1109/ICAECT49130.2021.9392518., pp. 1-6, 2021.
- [15] *IEEE Standard for the Functional Verification Language e*, IEEE Std 1647-2016 (Revision of IEEE Std 1647-2011), doi: 10.1109/IEEESTD.2017.7805158., pp.1-558, 6 Jan. 2017.
- [16] D. Wang, J. Yan and Y. Qiao *Research on Chip Verification Technology Based on UVM*, 2021 6th International Symposium on Computer and Information Processing Technology (ISCIPIT), Changsha, China, doi: 10.1109/ISCIPIT53667.2021.00030., pp. 117-120, 2021.
- [17] R. Shrivastava and K. Sarawadekar *Multi-Voltage GPIO Design and its Physical Implementation*, 2018 4th International Conference for Convergence in Technology (I2CT), Mangalore, India, doi: 10.1109/I2CT42659.2018.9058052., pp. 1-4, 2018.
- [18] Q. Hou and S. Ding *GPIO Based Super-Twisting Sliding Mode Control for PMSM*, IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 68, no. 2, doi: 10.1109/TCSII.2020.3008188., pp. 747-751, Feb. 2021.
- [19] Hiremath, Sujatha and Y. Vinayakgouda *RTL to GDSII Implementation of Advanced High-Performance Bus Lite*, journal = 2021 6th International Conference on Communication and Electronics Systems (ICES),doi: 10.1109/ICES51350.2021.9488947, pp. 303-306, Jul.

2021

- [20] Sumanth Sajjanar and Deepika Prabhakar A Review on BIST Embedded I2C and SPI using Verilog HDL, 2020, url: <https://api.semanticscholar.org/CorpusID:235817964>

