UVM-BASED SERIAL PERIPHERAL INTERFACE INTEGRATED WITH APB BUS ARCHITECTURE

[1]Manasa Ganga N, [2]Tejaswini Gull

[1]Student of MTech, [2]Assistant Professor
[1]Department of Electronics and Communication,
[1] Visvesvaraya Technological University, Belagavi, India

Abstract: Quick and dependable communication between processors and peripheral devices is crucial in contemporary System-on-Chip (SoC) applications. This work uses System Verilog and Universal Verification Methodology (UVM) to design and verify an APB-based Serial Peripheral Interface (SPI) Master Core. Flexible data transmission multiple SPI operating modes using CPOL and CPHA configurations configurable master-slave communication and full-duplex serial communication are all supported by the suggested design. Through programmable registers the APB interface facilitates effective control and communication between the processor and SPI peripheral. A reusable and scalable UVM verification environment with a sequencer driver monitor scoreboard and constrained random test cases is also created in order to accomplish a full functional validation. To guarantee that all crucial protocol operations—such as read/write transactions clock synchronization slave selection transmission modes and error handling conditions—are confirmed functional coverage-driven verification techniques are used. In order to continuously monitor protocol behavior and identify timing violations invalid state transitions and communication mismatches during simulation assertions are incorporated. By addressing the various SPI modes transaction scenarios boundary conditions and corner cases high functional coverage and code coverage are attained guaranteeing thorough design verification. ModelSim Questsim and Verilog Complier Simulator (VCS) are used to simulate the suggested controller. The results show stable data transfer dependable communication performance increased debugging efficiency and improved design reliability. The verification results show that integrating System Verilog UVM functional coverage and assertion-driven techniques creates a structured and reusable verification environment that supports dependable communication in advanced SoC architectures improves validation efficiency and strengthens design reliability

Index Terms **- Parallel-to-Serial Conversion, Advanced Peripheral Bus (APB), Serial Peripheral Interface (SPI), Protocol Verification, Universal Verification Methodology (UVM).**

INTRODUCTION

The development of System-on-Chip (SoC) technology has increased the complexity of contemporary digital systems. These systems use a single chip for processing units' memory blocks and peripherals. As a result, communication between components is crucial. Standardized communication protocols are used within a SoC to guarantee data exchange. Two protocols are Advanced Peripheral Bus (APB) and Serial Peripheral Interface (SPI). They facilitate data communication and assist in connecting low-speed peripherals. The AMBA architecture incorporates the APB protocol. It offers a low-power interface for connecting peripherals such as GPIOs UARTs and timers. Because of this it can be used in low-bandwidth applications. Microcontrollers and external devices can communicate quickly and serially using the SPI protocol. It is capable of full-duplex communication. It offers many options for configuration. The advantages of both interfaces are combined in an APB-based SPI protocol. The serial communication interface is SPI and the processor-side communication bus is APB. The SPI peripheral is controlled by the APB master via APB transactions. Data transfer with external devices is managed by the SPI module. This integration makes it possible for the processor and external peripherals to communicate. Power consumption and design simplicity are also preserved. Extensive verification becomes increasingly important as integrated circuits become more complex. It guarantees protocol compliance and functionality. Verification coverage is frequently not achieved by traditional testing methods. They might fail to consider special cases. A tool for creating testbenches is System Verilog. However, as the complexity of the design increases using it alone may become challenging. The creation and administration of multiple tasks are necessary for a System Verilog-based verification environment. Code duplication and reduced reusability may result from this. System Verilog is the foundation of UVM. It provides a framework for verification with predetermined elements and procedures. UVM encourages reusability scalability and modularity. It enables verification engineers to design easily expandable environments. Features that facilitate through verification are supported by UVM. These consist of functional coverage collection factory-based component creation and constrained stimulus generation. Configuration mechanisms and assertion integration are also supported. UVM is therefore chosen for the APB-based SPI protocol verification environment. It provides better scalability enhanced coverage simpler maintenance and increased verification efficiency. A widely used verification framework is the Universal Verification Methodology (UVM). Scalable and modular verification environments are encouraged. Sequencers drivers and monitors are among the components that UVM offers. These make it possible to verify intricate designs. Enhancing design reliability is the goal of the suggested verification methodology.
Additionally, it seeks to validate the APB-based SPI communication system and lessen the verification effort. Techniques for assertion-based verification and functional coverage are applied. They gauge how thorough the verification is. Make sure the protocol is correct. This is completed prior to the hardware being implemented.

Literature Survey

The research community's interest in protocol verification using structured methodologies has grown steadily alongside the increasing complexity of SoC designs. Early investigations centred on the APB itself: teams built UVM environments with reusable drivers, monitors, sequencers, and scoreboards, and these efforts laid the conceptual groundwork for coverage-driven verification as a discipline [1]. The lessons learned—how to structure a testbench, how to measure coverage, how to report results—transferred naturally to other protocols.

SPI verification followed a similar evolution. Researchers constructed dedicated UVM testbenches that could exercise the protocol across its four clock-mode variants and under varying data-width configurations, demonstrating that master-slave communication could be validated systematically rather than through ad hoc waveform inspection [2]. Comparative studies showed that UVM environments consistently outperformed hand-written directed-test approaches in terms of coverage completeness and testbench reuse across related projects [3]. The role of the monitor component received particular attention: by translating raw signal transitions into structured transaction objects, monitors make it practical for scoreboards to perform meaningful data checks rather than just signal-level comparisons [4].

As designs grew more ambitious, verification teams began tackling SPI-based memory subsystems and multi-master configurations, leaning heavily on functional coverage databases and assertion libraries to track protocol invariants [5, 6]. More recent work has moved toward integrated APB-SPI controllers that expose programmable parameters—clock divider, bit order, transfer width, and mode—through a register map, requiring the testbench to explore a much larger configuration space [7–9]. Reusable SPI intellectual-property blocks with embedded APB register files have also appeared, aiming to shorten integration time across different SoC projects [10].

Despite this body of work, a self-contained study that combines all three major verification pillars—UVM-structured constrained-random testing, functional coverage analysis, and assertion-based protocol checking—within a single APB-SPI bridge environment has remained relatively rare. The present work attempts to fill that gap by documenting both the bridge architecture and the complete verification strategy, including the coverage results obtained.

RESEARCH METHODOLOGY

A. Why UVM Rather Than Plain System Verilog?

System Verilog on its own is expressive enough to describe almost any testbench concept: object-oriented classes, randomization constraints, cover groups, and concurrent assertions are all part of the language. The difficulty is not capability but organization. Without a shared framework, different engineers on the same project—or even the same engineer at different points in time—tend to structure their environments differently, making reuse across projects painful and debugging unfamiliar code time-consuming.UVM solves this by imposing a well-defined component hierarchy and a lifecycle protocol. Drivers, monitors, sequencers, agents, environments, and tests each have a defined role and a defined interface to their neighbours. The phased startup (build, connect, start-of-simulation, run, extract, report) ensures that every component is properly initialized before stimulus begins. The configuration database lets a test pass setting to deeply nested components without hard-coded paths. These conventions mean that an engineer familiar with UVM can read an unfamiliar testbench far more quickly than one written in an ad hoc style, which directly reduces the time spent understanding code versus actually finding bugs.

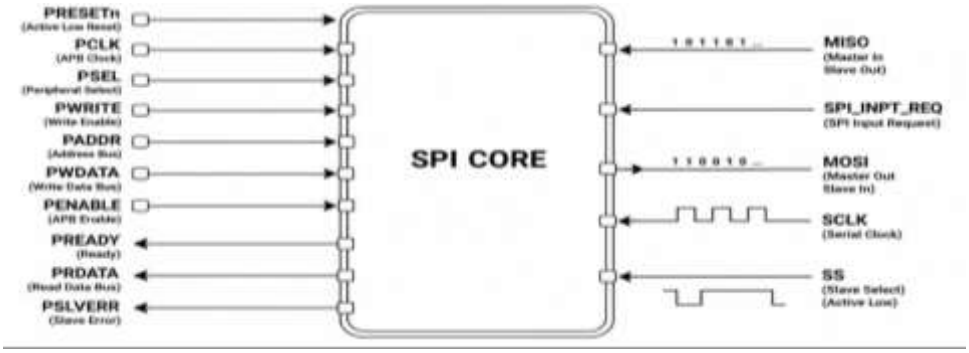

Fig.1 SPI Pin Diagram

The APB-SPI bridge exposes two groups of signals. On the processor side, the APB interface includes the standard PCLK system clock and active-low PRESETn reset; PSEL and PENABLE qualify each bus transaction; PADDR selects the target register; PWRITE distinguishes reads from writes; PWDATA carries write data inbound; PRDATA returns read data to the processor; PREADY confirms that the transaction has completed; and PSLVERR flags any protocol error. On the peripheral side, the SPI interface reduces to four lines: SCLK (the serial clock generated by the master), MOSI (data flowing from master to slave), MISO (data flowing from slave back to master), and SS (Slave Select, which is asserted low to activate a specific peripheral). The bridge also exposes a configurable lsbfe bit that controls whether the least-significant or most-significant bit is transmitted first.

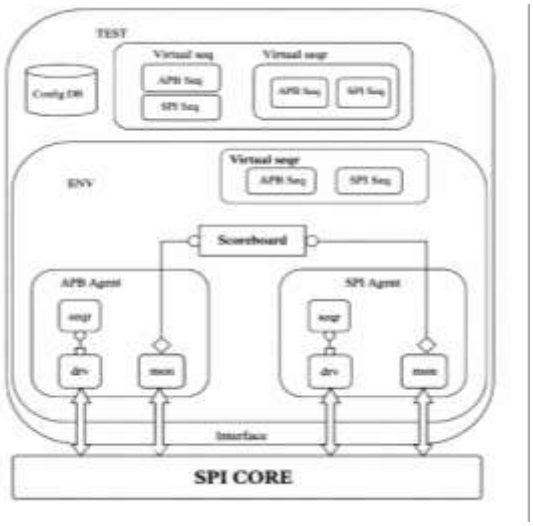

Fig.2 UVM TB Architecture

The SPI Core is instantiated as the Device Under Test (DUT) by the top-level testbench which then binds the SPI System Verilog and APB interfaces to it and starts the UVM root. The UVM Test then chooses a scenario adds data to the configuration database and initiates a virtual sequence. This virtual sequence serves as a conductor directing the SPI Agent to carry out the appropriate serial transactions after instructing the APB Agent to write the desired control and data register values into the SPI Core. Sequencer driver and monitor are the three well-known components of the APB Agent. Transaction objects are extracted from the active sequence by the sequencer and sent to the driver which converts each object into timed APB signal-level interface events. The same signals are observed by the monitor which then reconstructs the transaction objects and transmits them to the scoreboard. The SPI Agent has the same structure: its monitor records the serial bit stream and reconstructs the sent and received bytes while its driver controls the MOSI MISO SCLK and SS lines. A reference model of the anticipated register behavior is stored in the Scoreboard. The scoreboard keeps track of the value entered into each register each time the APB monitor delivers a completed write transaction. The scoreboard determines what the MOSI output should have been given the most recent PWDATA write and what PRDATA should return given the MISO data the SPI driver supplied each time the SPI monitor reports a successful transfer. Any discrepancy is immediately reported as an error saving the engineer from having to navigate through waveforms. Which register addresses were accessed which SPI modes (all four CPOL and CPHA combinations) were used whether both read and write paths were used and whether the lsbfe bit was tested in both orientations are all tracked by functional coverage groups. Concurrent System Verilog Assertions confirm cycle by cycle that data on MOSI is stable for the necessary setup time around each SCLK edge that PENABLE is never asserted without a preceding PSEL and that SCLK stays stable while SS is deasserted.

IV. RESULTS AND DISCUSSION

A. MSB-First SPI Transfer

The waveform shows that a transaction using APB-based SPI is working correctly in a mode. This mode has CPOL set to 1 and CPHA set to 1. When the master is writing data using APB it puts the number 10011001 on the PWDATA bus. At the time it sends the address through PADDR and also sends out PSEL, PENABLE and PWRITE signals. After the SPI controller is set up the SS signal becomes active and serial communication starts. Because CPOL is 1 the SCLK signal stays high when it is not being used. Data is sent according to the rules for CPHA set to 1. The MOSI and MISO lines. Receive data at the same time as the clock pulses.

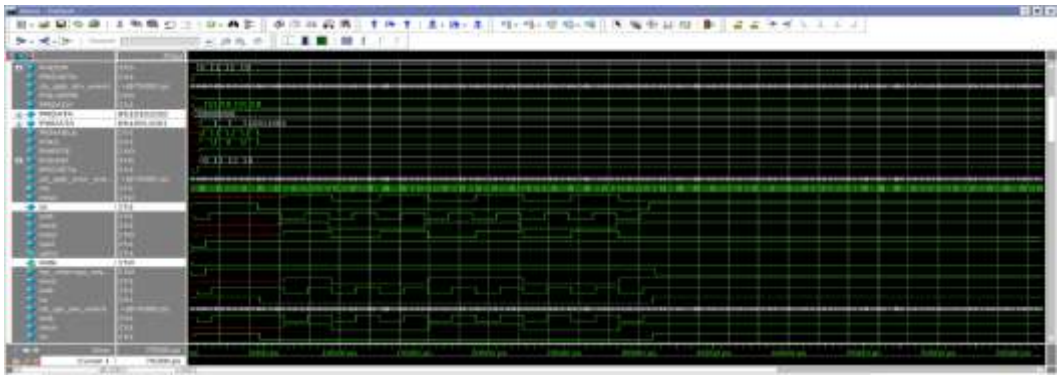

Fig.3 MSB First SPI Transfer

When the transaction is finished the PRDATA bus shows 10101010. This means that the data was received correctly from the SPI slave using an APB read operation. The SS signal going back, to its state means that the SPI communication cycle is complete. This shows that both the APB and SPI interfaces are working correctly.

B. LSB-First SPI Transfer

An effective APB-controlled SPI communication in LSB-first mode is shown by this waveform. When using PADDR to access various register addresses the APB master writes the value 00111000 on the PWDATA bus. PSEL PENABLE and PREADY are control signals that denote completed and legitimate APB transactions.

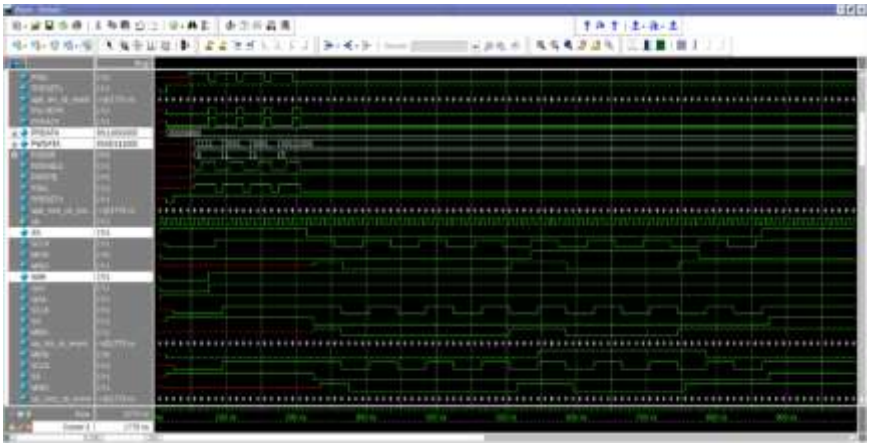

Fig.4 LSB First SPI Transfer

The SS signal turns on after configuration allowing communication with the SPI slave. While the MOSI line transmits data and the MISO line receives data from the slave device the SCLK signal synchronizes the serial transfer. The SPI transfer starts with the least important bit since lsbfe = 1.The PRDATA bus displays 11001000 at the conclusion of the communication indicating that the data was received successfully. The waveform confirms accurate serial data exchange between the master and slave devices correct APB register access and appropriate SPI timing.

C. Power Behavior

Reducing needless switching activity in the SPI interface when no transaction is underway was one design objective. Simulation verified that SCLK cases to toggle as soon as the last bit of each transfer is clocked and doesn't start up again until a new transaction starts. In between MOSI MISO and SS likewise maintained steady logic levels which directly lowers the capacitive switching energy that controls dynamic power in digital circuits. The internal data path of the SPI Core is only activated when both PSEL and PENABLE are asserted minimizing idle-state power.

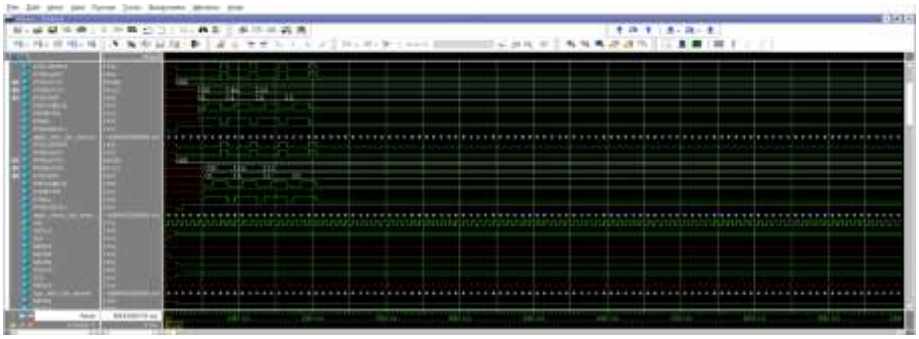

Fig.5 Low Power Consumption

D. SPI Operating Modes

The SPI standard defines four clock modes through the combination of Clock Polarity (CPOL) and Clock Phase (CPHA). The table below summarises these:

Clock Phase	Clock Polarity	Modes of Operation
0	**0**	**Mode 0**
0	**1**	**Mode 1**
1	**0**	**Mode 2**
1	**1**	**Mode 3**

Table 1: SPI Modes

All four modes were exercised in the verification campaign, with assertions confirming that SCLK polarity at idle and the sampling edge matched the configured CPOL/CPHA values in every case.

E. UVM Simulation Report and Coverage Metrics

The UVM end-of-simulation report recorded all expected scoreboard checks as passing. MOSI data-match checks compared the serialised bit stream against the reference model's expected output on a per-byte basis; every comparison passed. MISO data-compare checks validated that the assembled receive register value equaled what the SPI agent had driven; these too passed without exception.

Fig.6 Functional Coverage Report

The majority of the APB-SPI controller features and operating conditions were successfully confirmed as evidenced by the functional coverage reports 93. 06 percent coverage. Throughout the simulation all 38 assertion bins were hit without any misses indicating proper protocol behavior. To ensure comprehensive functional validation the covergroups kept an eye on APB transactions SPI operations register accesses and feature interactions. Ten of the 29 defined scenarios are still untested and can be addressed with more test cases while the remaining 19 were covered. The design is stable functionally correct and successfully verified under a variety of operating conditions according to the overall results.

5. CONCLUSION

By guaranteeing proper protocol operation data integrity and register access the APB-Based SPI Protocol Verification using UVM project effectively validated communication between the APB bus and SPI controller. To attain high verification coverage and trustworthy outcomes a reusable UVM-based verification environment with functional coverage assertions scoreboards and constrained-random testing was created. Support for numerous SPI slaves sophisticated SPI features low-power verification error injection and stress testing to increase verification comprehensiveness and industrial applicability are some potential future improvements.

REFERENCES

[1] C. W. Liao, H. C. Yu, and Y. C. Liao, "Verification of SPI Protocol Using Universal Verification Methodology for Modern IoT and Wearable Devices," Electronics, vol. 14, no. 5, p. 837, 2025.
[2] L. S. Kamireddy and L. S. Kamireddy, "UVM Based Reusable Verification IP for Wishbone Compliant SPI Master Core," International Journal of Engineering and Technology, 2018.
[3] S. Suryawanshi, A. Patil, and R. Kulkarni, "A Unified UVM Testbench Framework for APB-Based Peripheral Subsystems: Architecture and APB-SPI Demonstration," Proceedings of the World Academy of Science, Engineering and Technology, 2024.
[4] M. Prasanthi and K. V. S. Reddy, "Verification of AMBA APB Protocol Using Universal Verification Methodology," International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering (IJIREEICE), 2023.
[5] S. S. Rao and P. V. Lakshmi, "Coverage-Based Verification of AMBA APB Protocol Using UVM," International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), 2022.
[6] A. S. Kumar and B. R. Prakash, "Functional Verification of SPI Protocol Using UVM Based Environment for Flash Memory Applications," International Conference on Signal Processing and Communication, 2023.
[7] A. K. Gupta and R. Sharma, "Design and Verification of AMBA AHB to APB Bridge Using SystemVerilog and UVM," Proceedings of International Conference on Electronics and Communication Engineering, 2023.
[8] S. Ray and J. Bhadra, "Verification of AMBA Protocols Using Assertion-Based Verification and Functional Coverage," IEEE International Symposium on VLSI Design and Test, 2021.
[9] Y. Hu, X. Zhang, and J. Liu, "UVLLM: An Automated Universal RTL Verification Framework Using Large Language Models," arXiv Preprint, 2024.

Copyright & License:

© Authors retain the copyright of this article. This work is published under the Creative Commons Attribution 4.0 International License (CC BY 4.0), permitting unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.