International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 11 | Nov 2026 www.irjet.net p-ISSN: 2395-0072

Design and UVM Based Functional Verification of an AMBA AHB to APB Bridge with SRAM Peripheral Integration

Jesmitha S[1], Sahil Jangra[2], Sachin Kumar[3], Sanskar Dubey[4], Shikhar Sinha[5]

[1]*Department of ECE, Bangalore Institute of Technology, Bengaluru, Karnataka, India*
[2345]*VTU, ECE, BIT, Bengaluru, Karnataka, India*

Abstract - *This paper presents the RTL design and UVM-based verification of an AMBA AHB-to-APB bridge written in SystemVerilog. The bridge captures an AHB request, translates it through a three-state IDLE–SETUP–ACCESS finite state machine, drives an APB transaction to an SRAM controller, holds the AHB response while the APB slave inserts wait states, and propagates the APB slave-error signal PSLVERR to the AHB-side HRESP. The design is verified with a UVM environment comprising a transaction class, sequencer, driver, monitor, agent, reference-memory scoreboard and functional-coverage subscriber, supported by SystemVerilog Assertions (SVA) for protocol checking. An automated Questa regression of seven tests covers basic and multiple write/read traffic, INCR bursts, a coverage-closure test, an intentional SVA violation, an APB wait-state test and an APB error-response test. All seven tests pass with zero UVM errors and zero UVM fatal errors, and the functional coverage model reaches 100% of its 15 explicitly defined bins. This figure refers only to those 15 bins and is not a claim that the design is free of bugs. Limitations of the current verification scope and directions for extending it are discussed.*

Key Words: AHB, APB, AMBA, bus bridge, SystemVerilog, UVM, SystemVerilog Assertions, functional coverage, scoreboard, FSM, Questa.

1.INTRODUCTION

Modern System-on-Chip (SoC) designs connect processors, memories and peripherals through hierarchical on-chip buses. The Advanced High-performance Bus (AHB) serves latency-sensitive masters and memories, whereas the Advanced Peripheral Bus (APB) provides a simple, low-power interface for slower peripherals [1], [2]. An AHB-to-APB bridge joins the two domains: it behaves as an AHB slave towards the system bus and as an APB master towards the peripherals, converting each AHB request into the two-phase APB SETUP/ACCESS protocol and relaying wait states and error responses back to the AHB side.

Because the bridge lies on the path of every peripheral access, errors such as corrupted data, incorrect wait-state handling or a lost error response can be hard to find late in a design cycle. Class-based verification with the Universal Verification Methodology (UVM) [3], written in SystemVerilog [4], together with assertions [5] and functional coverage [6], offers a systematic and reusable way to build confidence in such a block.

This work makes the following contributions: (i) an RTL implementation of an AHB-to-APB bridge with an IDLE–SETUP–ACCESS FSM, APB wait-state handling and PSLVERR-to-HRESP propagation; (ii) a complete UVM environment with a reference-memory scoreboard; (iii) SVA-based protocol checking with a dedicated violation test; (iv) a 15-bin functional coverage model with a coverage-closure test; and (v) an automated Questa regression of seven tests.

The rest of the paper is organised as follows. Section 2 gives background on AMBA and UVM. Section 3 describes the bridge RTL, Section 4 the verification environment and Section 5 the coverage model. Section 6 reports results, Section 7 discusses limitations and future work, and Section 8 concludes.

2. BACKGROUND

2.1 AMBA AHB and APB

AHB is a high-performance member of the AMBA family that supports burst transfers and a pipelined address/data structure [1]. APB targets low-bandwidth peripheral access. Every APB transfer begins with a SETUP phase, in which PSEL is asserted while PENABLE is low, followed by an ACCESS phase, in which PENABLE is asserted. The peripheral may extend the ACCESS phase by holding PREADY low and may flag a failed transfer by asserting PSLVERR when the transfer completes [2].

2.2 UVM, SVA and Functional Coverage

UVM is a SystemVerilog class library standardised by Accellera that structures a testbench into reusable components (sequence items, sequences, sequencers, drivers, monitors, agents, scoreboards and environments)

connected through transaction-level analysis ports [3], [7]. SystemVerilog Assertions express temporal protocol rules that are checked continuously during simulation [5], and covergroups measure how much of a specified functional space has actually been exercised [6].

3. BRIDGE RTL DESIGN

3.1 Architecture

Fig. 1 shows the structure of the bridge implemented in rtl/ahbtoapb.sv. The AHB-side inputs are HSEL, HADDR, HWRITE, HWDATA and HBURST. When a request is accepted, the bridge latches the address, write data, direction and burst type into request-capture registers (addr_reg, wdata_reg, write_reg and burst_reg). A state register, current_state, holds the FSM state. The RTL uses separate sequential logic for state and request capture and combinational logic for next-state computation, APB output generation and AHB response generation.

On the APB side the bridge drives PSEL, PENABLE, PWRITE, PADDR and PWDATA to an SRAM controller and samples PRDATA, PREADY and PSLVERR. Towards the AHB side it returns HRDATA, HREADY and HRESP.

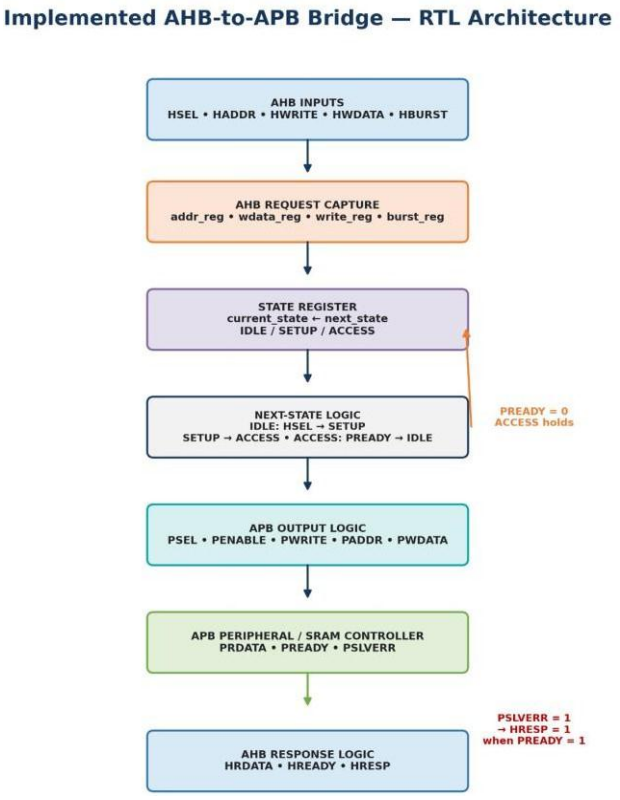


Fig. 1. Architecture of the implemented AHB-to-APB bridge

3.2 Finite State Machine

The bridge is controlled by the three-state FSM of Fig. 2. In IDLE the bridge captures a request when HSEL = 1 and keeps HREADY = 1. In SETUP it asserts PSEL with PENABLE = 0 and drives HREADY = 0, stalling the AHB master. In ACCESS it asserts both PSEL and PENABLE and drives HREADY = PREADY. The bridge moves from IDLE to SETUP when a request is selected, from SETUP to ACCESS unconditionally, and leaves ACCESS once the APB peripheral signals completion with PREADY = 1.

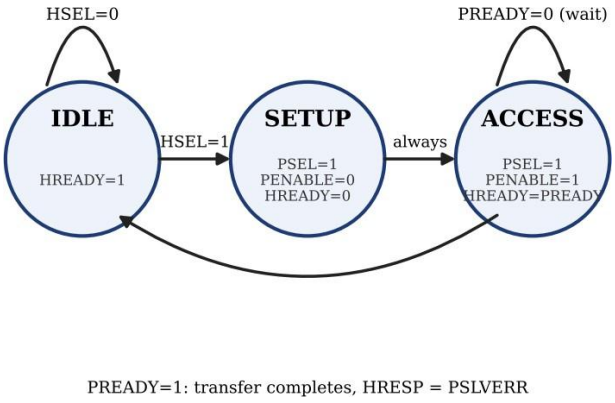


Fig. 2. Bridge FSM with the main output values of each state

3.3 Wait States and Error Response

While PREADY = 0 the bridge remains in ACCESS and HREADY stays low, so AHB completion is held off until the peripheral is ready. When the APB transfer completes (PREADY = 1) the bridge drives HRESP = PSLVERR, which makes a slave error visible to the AHB master at the end of the transfer; for reads it also returns HRDATA = PRDATA.

4. UVM VERIFICATION ENVIRONMENT

4.1 Architecture

The testbench is organised as transaction, sequencer, sequences, driver, monitor, agent, scoreboard, coverage, environment and tests, all collected in a single package (ahb_uvm_pkg.sv). The environment instantiates an AHB agent, a scoreboard and a coverage subscriber, while the agent contains the sequencer, driver and monitor (Fig. 3). Sequences generate transactions that the sequencer passes to the driver; the driver applies them to the DUT through the AHB virtual interface. The monitor observes the interface and publishes the observed transactions through its analysis port to the scoreboard and the coverage subscriber.

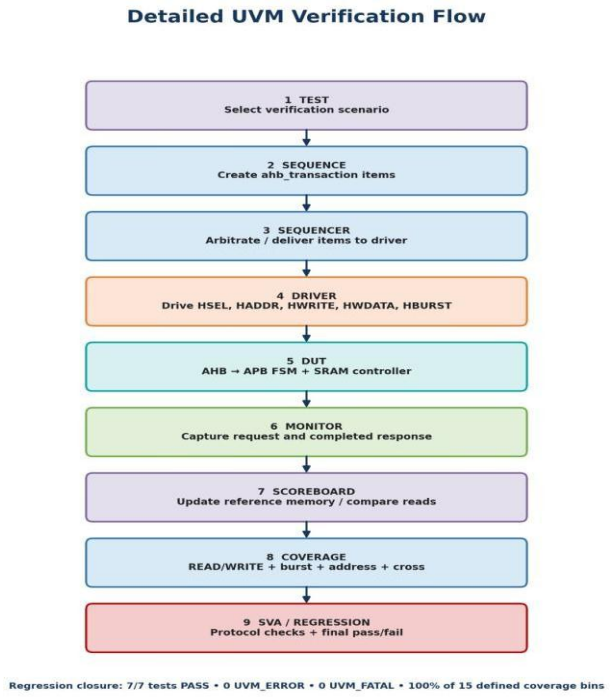


Fig. 3. UVM verification environment and data flow

4.2 Stimulus

Four groups of stimulus are provided: a basic write/read sequence, multiple write/read transactions, INCR burst traffic, and coverage-closure traffic designed to reach every bin of the coverage model. Three further tests target specific behaviours: an APB wait-state test, an APB error-response test and an SVA violation test.

4.3 Reference-Memory Scoreboard

The scoreboard implements the self-checking approach described in [8]. It maintains a reference memory that mirrors the SRAM behind the bridge: each observed write updates the model, and each observed read is compared against the model's expected value. Any mismatch is reported as a UVM error.

4.4 SVA Protocol Checking

SystemVerilog Assertions monitor the protocol behaviour of the bridge during simulation. [List the specific assertions implemented in the assertions/ directory, e.g. the properties checked on the APB SETUP/ACCESS sequence, PSEL/PENABLE relationship and wait-state behaviour.] A dedicated SVA violation test deliberately creates a protocol violation, and detection of that violation by the assertions is its PASS condition.

4.5 Regression Flow

The repository includes Questa/ModelSim scripts (sim/regression.sh) that compile the design and testbench and run the tests, each selected through the +UVM_TESTNAME plusarg. The regression collects the pass/fail status of every test, together with the UVM error and fatal counts and the functional coverage report.

5. FUNCTIONAL COVERAGE MODEL

The coverage subscriber (coverage/ahb_coverage.sv) defines exactly 15 bins, listed in Table I. The address space is divided into four equal ranges, and the cross of operation and burst type contributes six bins (three for reads and three for writes), giving 2 + 3 + 4 + 6 = 15 bins. Coverage is reported as with total_bins = 15.

Coverage (%) = (covered_bins / total_bins) × 100 (1)

TABLE I. FUNCTIONAL COVERAGE BINS

Category	Bins	Count
Operation	READ, WRITE	2
Burst type	SINGLE, INCR, INCR4	3
Address range	00–3F, 40–7F, 80–BF, C0–FF	4
Operation × burst (cross)	READ × 3 bursts, WRITE × 3 bursts	6
Total		**15**

6. RESULTS AND DISCUSSION

6.1 Regression Results

Table II summarises the seven-test regression. All seven tests pass, none fail, and the run reports zero UVM errors and zero UVM fatal errors. For the SVA violation test, detecting the intentionally injected protocol violation is the expected outcome and therefore counts as a pass.

TABLE II. REGRESSION RESULTS

#	Test	Purpose	Result
1	Basic Write/Read	Write followed by read-back	PASS
2	Multiple Write/Read	Several write/read transactions	PASS
3	INCR Burst	INCR burst traffic	PASS
4	Coverage Closure	Directed traffic to hit all bins	PASS
5	SVA Violation	Intentional violation must be detected	PASS
6	APB Wait State	PREADY held low before completion	PASS
7	APB Error Response	PSLVERR propagated to HRESP	PASS

6.3 Wait-State and Error-Response Behaviour

In the wait-state test the APB peripheral holds PREADY low and the bridge remains in ACCESS; three wait cycles were observed before the transfer completed, and AHB completion was withheld until the peripheral became ready. In the error-response test, the PSLVERR asserted by the peripheral at completion is propagated to HRESP on the AHB side, as specified in Section 3.3.

6.2 Coverage Closure

The coverage-closure test drives all 15 bins of Table I, giving 15 / 15 = 100% functional coverage. This figure applies only to the bins explicitly defined in coverage/ahb_coverage.sv. It shows that the specified operation, burst, address-range and cross scenarios were exercised; it is not a claim that the design is mathematically free of bugs.

6.3 Discussion

Taken together, the results indicate that the bridge behaves as specified for the scenarios that were exercised: data written through the bridge is read back correctly according to the reference-memory scoreboard, wait states stall the AHB side, and slave errors reach HRESP. The assertions add an independent protocol-level check, and the deliberate violation test confirms that those checks are able to fail when a violation occurs.

7. LIMITATIONS AND FUTURE WORK

The verification scope has limits that should be stated clearly. The bridge models a simplified AHB subset (HSEL, HADDR, HWRITE, HWDATA and HBURST) and a single APB peripheral. The coverage model contains 15 bins, which is modest, and the burst coverage is limited to SINGLE, INCR and INCR4. Stimulus is limited to the sequences described in Section 4.2. Code coverage and formal analysis are not reported.

Future work includes extending the stimulus with constrained-random sequences run over multiple seeds, further burst types (for example WRAP4 and INCR8), back-to-back and mid-transfer scenarios, a richer cross-coverage model, code-coverage closure, multiple APB slaves with address decoding, and formal verification of the key assertions.

8. CONCLUSION

This paper presented an AHB-to-APB bridge implemented in SystemVerilog together with a UVM verification environment containing a reference-memory scoreboard, SVA protocol checking and a 15-bin functional coverage model. The automated regression of seven tests passes with zero UVM errors and zero UVM fatal errors, the wait-state and error-response behaviours are verified, and all 15 defined coverage bins are covered. The results demonstrate a complete, reproducible design-and-verification flow, while the stated limitations define a clear path for extending the verification in future work.

ACKNOWLEDGEMENT

The author thanks [Jesmitha S, Assistant Professor, MTech, Bangalore Institute of Technology] for guidance during this major project.

REFERENCES

[1] ARM Ltd., "AMBA 3 AHB-Lite Protocol Specification," ARM IHI 0033A, 2006.
[2] ARM Ltd., "AMBA APB Protocol Specification, Version 2.0," ARM IHI 0024C, 2010.
[3]Accellera Systems Initiative, "Universal Verification Methodology (UVM) 1.2 User's Guide," 2015.
[4] IEEE Standard for SystemVerilog—Unified Hardware Design, Specification, and Verification Language, IEEE Std 1800-2017, 2018.
[5] S. Vijayaraghavan and M. Ramanathan, A Practical Guide for SystemVerilog Assertions. New York, NY, USA: Springer, 2005.
[6] C. Spear and G. Tumbush, SystemVerilog for Verification: A Guide to Learning the Testbench Language Features, 3rd ed. New York, NY, USA: Springer, 2012.
[7] R. Salemi, The UVM Primer: An Introduction to the Universal Verification Methodology. Boston Light Press, 2013.
[8]J. Bergeron, Writing Testbenches Using SystemVerilog. New York, NY, USA: Springer, 2006.

BIOGRAPHIES



"Jesmitha S is an Assistant Professor in the Department of Electronics and Communication Engineering at Bangalore Institute of Technology, Bengaluru, Karnataka, India.



"Sahil Jangra (1BI23EC135) is an undergraduate student in the Department of Electronics and Communication Engineering at Bangalore Institute of Technology, Bengaluru, Karnataka, India."



"Sachin Kumar (1BI23EC133) is an undergraduate student in the Department of Electronics and Communication Engineering at Bangalore Institute of Technology, Bengaluru, Karnataka, India."



"Sanskar Dubey(1BI24EC143) is an undergraduate student in the Department of Electronics and Communication Engineering at Bangalore Institute of Technology, Bengaluru, Karnataka, India."



"Shikhar Sinha(1BI23EC146) is an undergraduate student in the Department of Electronics and Communication Engineering at Bangalore Institute of Technology, Bengaluru, Karnataka, India."