VERIFICATION TEST PLAN

ECE-593: Fundamentals of Pre-Silicon Validation
Maseeh College of Engineering and Computer Science
Winter, 2025

Project Name: Design and Verification of AHB to APB Bridge using UVM

Members:
Maithreyi Venkatesan - 900119348
Raghavendra Davarapalli - 955118446
Neil Austin Tauro - 956372423
Suraj Vijay Shetty - 967611331

Table of Contents

1) Introduction ……..……2

a) Objective of verification plan…….2
b) Top level block diagram……2
c) Specifications for the design……..3

2) Verification requirements..……6

a) Verification levels……6
b) Controllability and Observability……6

3) Required Tools……..……7

4) Risks and Dependencies……...7

5) Functions…….……..……...8

6) Testing Methods..……..……...9

7)Verification structure……12

8) Verification Methodology……13

9) Verification Components……14

10) Coverage requirements…..……..14

11) Resource requirements...…….15

12) Schedule……...……..…….15

13) References/Citations……..……16

Introduction

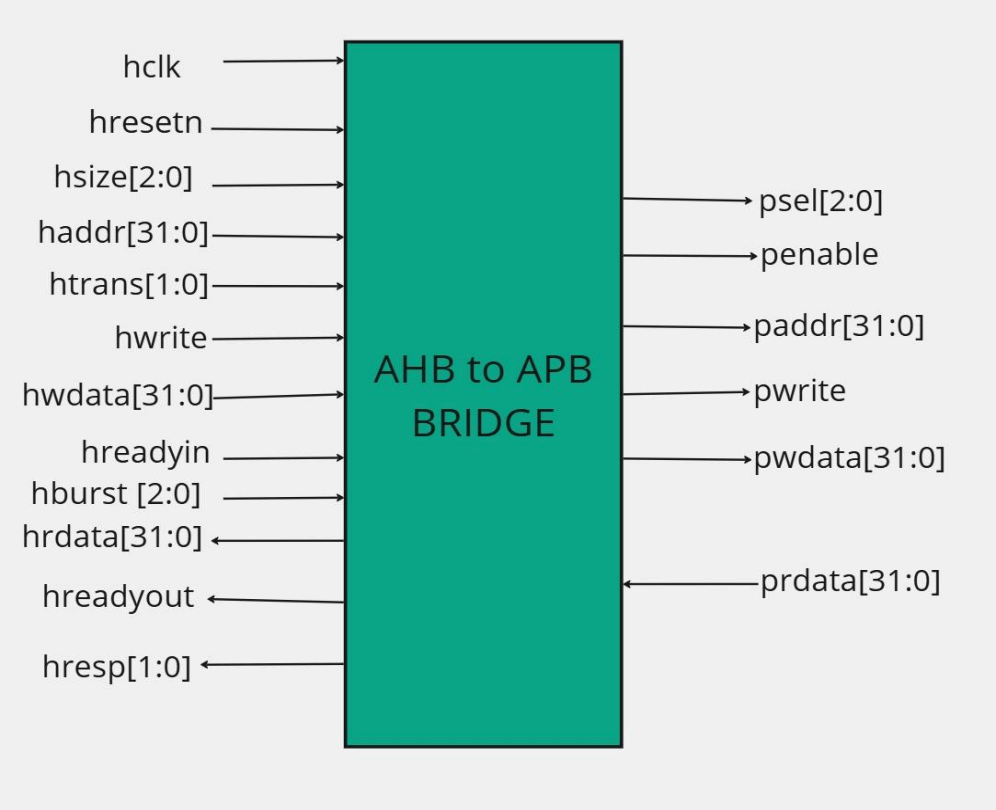
The goal of this verification plan is to specify the strategy and verification methodology that we incorporated to verify the functionality of the (Advanced High-Performance Bus) to APB (Advanced Peripheral Bus) bridge. The AHB to APB Bridge is a critical component in the AMBA (Advanced Microcontroller Bus Architecture) system, enabling seamless communication between high-performance master devices (on the AHP bus) and lower-bandwidth peripheral devices (on the APB bus).

a) Objective of the verification

The dummy (AHB Master) module includes two tasks, one dedicated to handling read operations and the other focused on managing write operations. These tasks are implemented to simplify the process of interacting with the design under test (DUT) by encapsulating the logic for reading and writing in reusable procedural blocks.

In addition to the tasks, a self-checking testbench has been developed to validate the functionality of the design. By combining the read and write tasks with the self-checking mechanism, the testbench ensures efficient and reliable verification of the design's behavior under different scenarios

b) Block Diagram



c) <u>Specifications for the design</u>

- The AHB to APB Bridge is the only synthesizable module in our project and the signals are all present exactly as in the block diagram. Since the AHB is pipelined and the APB is non-pipelined, we need to convert the AHB signals to APB signals by inserting wait states. This protocol will be according to the Finite State Machine (FSM) employed in the design specification.
- We are only verifying the single read/write and a simple incrementing by 4 burst transfer and this will be the scope of this project.

The signals that are present in the modules are provided in the table below,

Signals	Type	Direction	Description
HCLK	Bus clock	Input	This clock times all bus transfers.
HRESETn	Reset	Input	The bus reset signal is active LOW, and is used to reset the system and the bus.
HSIZE [2:0]	Data size	Input	Specifies the size of the data transfer.
HADDR [31:0]	Address bus	Input	The 32-bit system address bus.
HTRANS [1:0]	Transfer type	Input	This indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE	Transfer direction	Input	When HIGH this signal indicates a write transfer, and when LOW, a read transfer.

HWDATA [31:0]	Write data bus	Input	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HBURST [2:0]	Burst signals	Input	The hburst is a 3 bit signal.
HREADYIN/ HREADYout	Transfer done	Input/Output	When HIGH the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer
HRDATA [31:0]	Read data bus	Output	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation
HRESP [1:0]	Transfer response	Output	The transfer response gives further details about the status of a transfer. This module will consistently generate the OKAY response.
DATA READ [31:0]	Data read from hrdata for AHB Master	Output	The 32-bit data bus for read operations on the AHB Master side.

PSEL [2:0]	Peripheral slave select	Output	There is one of these signals for each APB peripheral present in the system. The signal indicates that the slave device is selected, and that a data transfer is required. It has the same timing as the peripheral address bus. It becomes HIGH at the same time as PADDR, but will be set LOW at the end of the transfer
PENABLE	Peripheral enable	Output	This enable signal is used to time all accesses on the peripheral bus. PENABLE goes HIGH on the second clock rising edge of the transfer, and LOW on the third (last) rising clock edge of the transfer.
PADDR [31:0]	Peripheral address bus	Output	This is the APB address bus, which may be up to 32 bits wide and is used by individual peripherals for decoding register accesses to that peripheral. The address becomes valid after the first rising edge of the clock at the start of the transfer. If there is a following APB transfer, then the address will change to the new value, otherwise it will hold its current value until the start of the next APB transfer.

PWRITE	Peripheral transfer direction	Output	This signal indicates a write to a peripheral when HIGH, and a read from a peripheral when LOW. It has the same timing as the peripheral address bus.
PWDATA [31:0]	Peripheral write data bus	Output	The peripheral write data bus is continuously driven by this module, changing during write cycles (when PWRITE is HIGH).
PRDATA [31:0]	Peripheral read data bus	Input	The peripheral read data bus is driven by the selected peripheral bus slave during read cycles (when PWRITE is LOW).

Verification Requirements

a) Verification levels

The verification process will be carried out at two levels: block (module) level and top level. At the block level, individual components/modules—such as the AHB Master, APB FSM Controller, and APB Slave—will be verified independently. At the top level, the integrated AHB-APB Bridge will be verified as a complete system.

b) Controllability and Observability

Controllability is typically high at this level since individual modules can be directly stimulated with specific input values to exercise all possible scenarios. Observability is also high because the outputs of each module (e.g., hreadyout, hrdata, prdata) can be monitored without interference from other components. Debugging is straightforward as there are fewer dependencies.

Required Tools

The verification environment for the AHB-APB bridge design will be built using several key tools and technologies:

a) Simulation Tool and Language*:*

QuestaSim serves as our primary simulation platform. It's particularly well-suited for our needs as it offers comprehensive verification capabilities and supports SystemVerilog, which is our chosen language. The tool provides robust debugging features that will be essential for our verification process.

For testbench development, we've selected SystemVerilog as our primary language. This choice leverages SystemVerilog's advanced features that extend beyond basic Verilog, including object-oriented programming concepts like classes, as well as randomization and functional coverage capabilities. These features will enable us to create thorough and sophisticated test scenarios.

b) Version Control System:

To ensure proper project management and collaboration, we'll implement version control using Git. This will help us track changes throughout the development process and maintain a clear history of our design and testbench modifications.

c) Coverage Analysis Tool:

Coverage analysis will be handled through QuestaSim's built-in capabilities. This will allow us to measure both code coverage and functional coverage, providing insights into the thoroughness of our verification efforts.

d) Documentation Tool:

For project documentation, we'll use a combination of Microsoft Word and Google Docs to maintain detailed records of our verification plan, test cases, and results.

This comprehensive toolset will enable us to effectively verify that the AHB-APB bridge meets all design specifications and functions correctly.

Risks and Dependencies

Risks: While pushing the code to GitHub, there is a high chance of merge conflicts to the main branch. We also have a very tight schedule. To mitigate these risks, proper branch management practices should be followed, including frequent merging, resolving conflicts promptly, and ensuring clear communication within the team. Understanding the complexity of the AHB to APB design takes time due to the intricacies of the bus protocols and signal mapping. It's important to thoroughly review the documentation and collaborate with the team to address any challenges as they arise.

Dependencies: Our project relies on the availability of the QuestaSim simulator, the completion of the RTL design, and access to the ARM documentation for the AHB-APB bridge. We will need to collaborate closely with the design team to ensure the design is completed on time and that any design changes are communicated to us promptly.

Functions

While our design includes distinct modules (AHB Master, AHB Slave Interface, APB FSM Controller, and APB Interface), our verification strategy will be driven by the AMBA specification document rather than module boundaries. We'll extract the required bridge functionalities from this specification and verify them against our RTL implementation. Each functionality will undergo thorough testing across multiple scenarios, encompassing normal operating conditions, boundary cases, and error situations. This specification-driven approach ensures we verify the design's actual requirements rather than just testing individual modules in isolation.

Functions to be verified:

1. Single read:
    - AHB master initiates a single read transaction
    - Bridge converts it to APB protocol
2. Single write:
    - AHB master initiates a single write transaction
    - Bridge converts the write command to APB protocol
3.
4. Reset:
    - Synchronous active low reset behavior verification
    - All state machines return to idle state
    - All control signals return to default values
    - Proper resumption of operation after reset
5. Address Decoding:
    - Correct slave selection based on address range
    - Proper address translation between AHB and APB domains
    - Valid/invalid address range handling
    - Address alignment checking

Functions not verified:

1. Burst transfers beyond INC by 4:
    - Current implementation limited to INCR4 bursts due to timing constraints and design complexity considerations
    - Other burst types (WRAP4, WRAP8, WRAP16, INCR8, INCR16, undefined length bursts) are not implemented
    - Future scope exists to expand burst support once core functionality is thoroughly verified
2. Error Handling:

- Basic error handling excluded from initial design to focus on core protocol conversion functionality
- Features like slave error responses, timeout conditions, and invalid address handling are currently not implemented
- Error handling mechanisms can be integrated in future versions

Critical Functions:

- Reset, Data transfer and addressing are critical functions that are mandatory to be correct for the basic functionality of the AHB to APB Bridge.

Testing Methods

In the initial stages, we will employ black box verification, focusing on the system's inputs and outputs to ensure it behaves as expected for given inputs. This approach, complemented by assertions for sanity checks and finite state machine transition verification, is well-suited for preliminary testing. As we progress, we will transition to grey box verification, leveraging our knowledge of the system's internal workings. This hybrid method is particularly beneficial for our AHB-APB bridge design, as it provides deeper insights into the interactions between components like the AHB Master, AHB Slave Interface, APB FSM Controller, and APB Interface, and their contributions to the system's overall functionality.

At this point, we are implementing directed test cases using the AHB Master module that sanity checks our design using single read/write and burst (INC by 4) operations. Moving forward, we will be moving onto constrained random testing with a class based testbench and finally moving onto a full fledged UVM based testbench.

Checking Methodology:

Our verification strategy incorporates two key approaches:

1. Comprehensive Output Verification: We'll validate all system outputs to ensure correct behavior under various input conditions. This includes:
    - Monitoring FSM state transitions
    - Verifying accuracy of read data operations
    - Confirming proper APB bus control signal timing and values
2. Multi-Level Design Verification: Our verification focuses on both functional and structural aspects:
    - Design Context: Verifying the bridge operates correctly within its intended use case
    - Microarchitecture: Ensuring internal components work together properly

- Component Interaction: Validating proper communication between the bridge's internal blocks
- Protocol Compliance: Confirming adherence to both AHB and APB specifications

Testcase Scenarios:

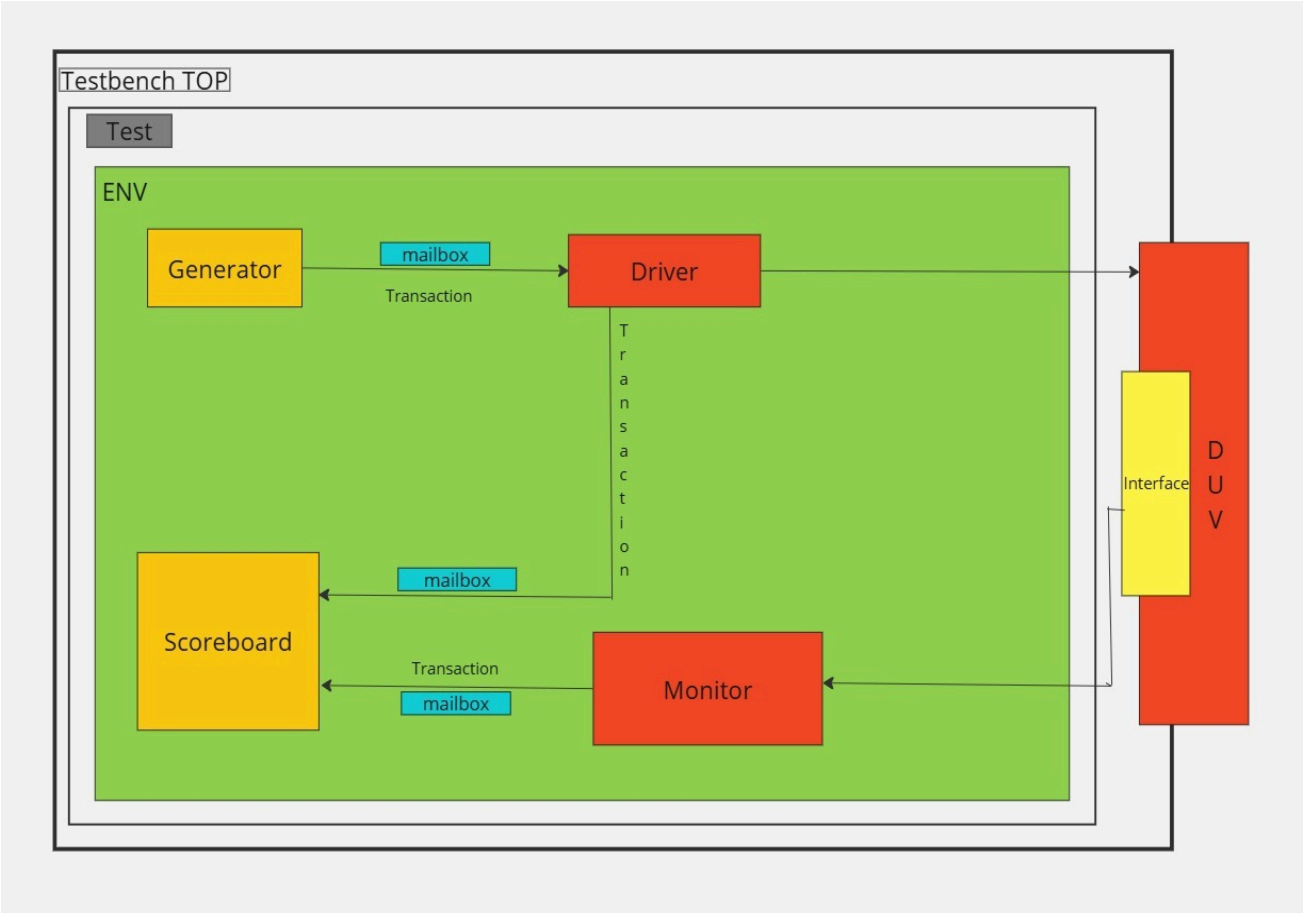
For now, simple test cases have been used for the standard conventional testbench.

Test Name/Number	Test Description/Features
Single write halfword non sequential	Performs a single non-sequential write transaction by randomizing and setting the necessary signals before sending the transaction to the driver.
Single read halfword non sequential	Performs a single non-sequential read transaction by randomizing and setting the necessary signals before sending the transaction to the driver.
Single write non sequential error	Performs a single non-sequential write transaction with an error response (`hresp=1`), randomizes and sets the necessary signals, then sends the transaction to the driver after sampling coverage.
Single read byte okay	Performs a single read transaction for a byte-sized transfer (`Hsize = 3'b000`), sets the necessary signals, samples coverage, and sends the transaction to the driver.
Single write halfword seq okay	Performs a single sequential write transaction by setting the necessary signals (`Hwrite = 1`, `Htrans = 2'b11`), samples coverage, and sends the transaction to the driver.
Single read word okay	Performs a single read transaction for a word-sized transfer (`Hsize = 3'b010`), sets the necessary signals, samples coverage, and sends the transaction to the driver.

Single write byte sequential error	Performs a sequential byte-sized write transaction (`Hsize=3'b000`) with an error response (`hresp=1`), samples coverage, and sends the transaction to the driver.
Single read byte nonsequential reset	Performs a single non-sequential byte-sized read transaction (`Hsize=3'b000`) with a reset condition (`hreset=1`), samples coverage, and sends the transaction to the driver.
Single write halfword nonsequential reset	Performs a single non-sequential halfword write transaction (`Hsize=3'b001`) with a reset condition (`hreset=1`), then sends the transaction to the driver.
Single read word nonsequential reset	Performs a single non-sequential word-sized read transaction (`Hsize=3'b010`) with a reset condition (`hreset=1`), then sends the transaction to the driver.
Single read byte sequential reset	Performs a sequential byte-sized read transaction (`Hsize=3'b000`) with a reset condition (`hreset=1`), then sends the transaction to the driver.
Single write halfword sequential reset	Performs a sequential byte read transaction with a reset condition (`hreset=1`) and sends it to the driver.
Single read word sequential reset	Performs a sequential word-sized read transaction (`Hsize=3'b010`) with a reset condition (`hreset=1`), randomizes the transaction, and sends it to the driver.

Single write byte sequential error reset	Performs a sequential byte-sized write transaction (`Hsize=3'b000`) with an error response (`hresp=1`) and a reset condition (`hreset=1`), randomizes the transaction, and sends it to the driver.
Single write byte idle error	Performs a byte-sized write transaction (`Hsize=3'b000`) with an idle transaction (`Htrans=2'b00`) and an error response (`hresp=1`), samples coverage, and sends it to the driver.

Verification IP structure



Verification Methodology

The testbench follows a layered architecture and consists of the following components:

- **Testbench Top**: Instantiates and connects all components.
- **Test**: Configures and starts the environment.
- **Environment (ENV)**: Contains the key verification components.
- **Generator**: Generates random or constrained stimulus.
- **Driver**: Drives the stimulus into the DUT through the interface.
- **Monitor**: Observes DUT outputs and forwards them to the scoreboard.
- **Scoreboard**: Compares expected vs. actual results for verification.
- **DUT (Design Under Test)**: The module being verified.

We employed a bottom-up testing approach, first verifying individual modules before integrating and testing larger subsystems. Starting with the transaction generator, we ensured its basic functionality using simple test scenarios before progressing to more complex cases. This method helped identify and resolve issues early.

We verified the transaction generator, which creates and drives transactions into the DUT. Initial tests covered basic functionality with simple scenarios. Once validated, we introduced more variables and conditions to explore the AHB APB Bridge's operational limits. Next, we integrated the monitor to observe DUT transactions and the scoreboard to validate results.

We began with basic write and read tests, gradually increasing complexity and coverage. As more test cases were introduced, we thoroughly assessed the DUT's robustness against various scenarios and edge cases. This structured approach ensured comprehensive validation of the AHB APB Bridge, confirming its compliance with specifications and functional correctness. By methodically verifying its reliability, we ensured a production-ready design.

Verification Components

- **Generator**: The generator is responsible for creating test transactions that simulate real-world scenarios. It defines the input data, control signals, and conditions to be applied to the DUT. By varying the transaction parameters, it ensures comprehensive functional verification.

- **Driver**: The driver receives transactions from the generator and converts them into signal-level interactions compatible with the DUT. It applies these signals through the interface, ensuring correct timing and protocol adherence.

- **DUT Interface**: This component serves as the communication bridge between the testbench and the DUT. It manages signal interactions, ensuring that the DUT receives properly formatted inputs and transmits outputs correctly.

- **Monitor**: The monitor passively observes all transactions occurring in the DUT without influencing its behavior. It records input and output data, helping in debugging and ensuring that the DUT operates as expected.

- **Scoreboard**: The scoreboard is responsible for comparing the expected outputs with the actual results produced by the DUT. It detects mismatches, logs errors, and ensures functional correctness, playing a crucial role in validation and debugging.

Coverage Requirements

Our verification strategy focuses on comprehensive testing of the design through multiple coverage metrics. We will use QuestaSim's code coverage tools to verify that all code lines have been exercised during testing. Beyond basic code coverage, we'll implement functional coverage to ensure we've tested all relevant bus operation scenarios, including edge cases and critical corner conditions. This dual approach of code coverage and functional coverage will help ensure that our verification effort is thorough and that the design behaves correctly across all possible usage scenarios.

Resource Requirements

a) Project Members
   - Neil Tauro - Design Specification and first half of Bridge controller
   - Suraj Shetty - Design Specification and APB slave interface
   - Maithreyi Venkatesan - Verification plan and Testbench development
   - Raghavendra Davarapalli - Verification plan, Interface for modules and AHB Master module

b) Computing resources
   - Remote Systems from Portland State University.

c) Simulator license
   - QuestaSim provided by PSU remote systems.

Schedule

Milestones	Date	Objectives
Milestone 1	01/31/2025	Defining the design specification and verification plan, along with developing the APB slave interface, simple testbench, module interfaces, and the AHB master module for seamless communication along with the bridge controller.
Milestone 2	02/18/2025	Class based testbench with all components and interfaces. Should verify ~50 random bursts of data.
Milestone 3	02/18/2025	Finalize changes in RTL. code and functional coverage. Update verification plan document.
Milestone 4	TBA	UVM testbench along with updated verification plan. UVM reporting macros to log reports/data to the console.
Final Deliverables	03/04/2025	Complete UVM architecture, environment and testbench. Bug injection. Finalized documents, paper and presentations.

References and Citations

- Utilized ClaudeAI for adding comments into the code and paraphrasing the document.
- RTL Design code referenced from "https://github.com/prajwalgekkouga/AHB-to-APB-Bridge"
- ARM Documentation "https://developer.arm.com/documentation/ihi0033/latest/"