VERIFICATION TEST PLAN

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

Group - 12

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

Members: 1. Daniel Jacobsen

2. Balaji Ginkal Harisha

3. Mohith Kumar Bennahatti Chikkegowda

4. Akshaya Kudumalakunte RaviKumar

Team GitHub Link: https://github.com/Mohithb19/TEAM_12_AHB2APB_bridge

Date: 02/18/2025

1 Table of Contents

2. Introduction:

The project of design and verification of AHB2APB is done using System Verilog and UVM based testbench development methodologies to create Verification IP components of the design. The APB2APB bridge is a crucial component in SOC architectures as it enables communication between the high-performance AHB (Advanced High-Performance Bus) and the low performance APB (Advanced Peripheral Bus).

The RTL code for our project is chosen from the github: https://github.com/prajwalgekkouga/AHB-to-APB-Bridge, The tool used to compile the design files and to verify is QuestaSim. QuestaSim was chosen as it provides full support for System Verilog and UVM.

2.1: Objective of the verification plan:

i. Functionality Verification: Ensure the bridge correctly transfers data and control signals between AHB and APB Buses.
ii. Protocol Compliance: Verify that AHB2APB bridge adheres to AHB and APB protocol Specifications.
iii. Wait State and Synchronization Handling: Check that bridge properly inserts wait states and synchronize transactions between pipelined AHB and non-pipelined APB.
iv. Scoreboard and Data Integrity Checks: Implement a scoreboard to compare expected vs actual data from AHB and APB monitors for correctness.
v. Corner Case and Stress Testing: Validate the bridge's operation under continuous transactions, bursts and multiple peripheral scenarios.
vi. Error and Exceptional Case Handling: Ensure the bridge correctly handles erroneous inputs, protocol violations and reset conditions.
vii. Coverage Metrics: Achieve functional and code coverage goals by implementing cover groups to measure verification completeness.

2.2 Top Level block diagram:

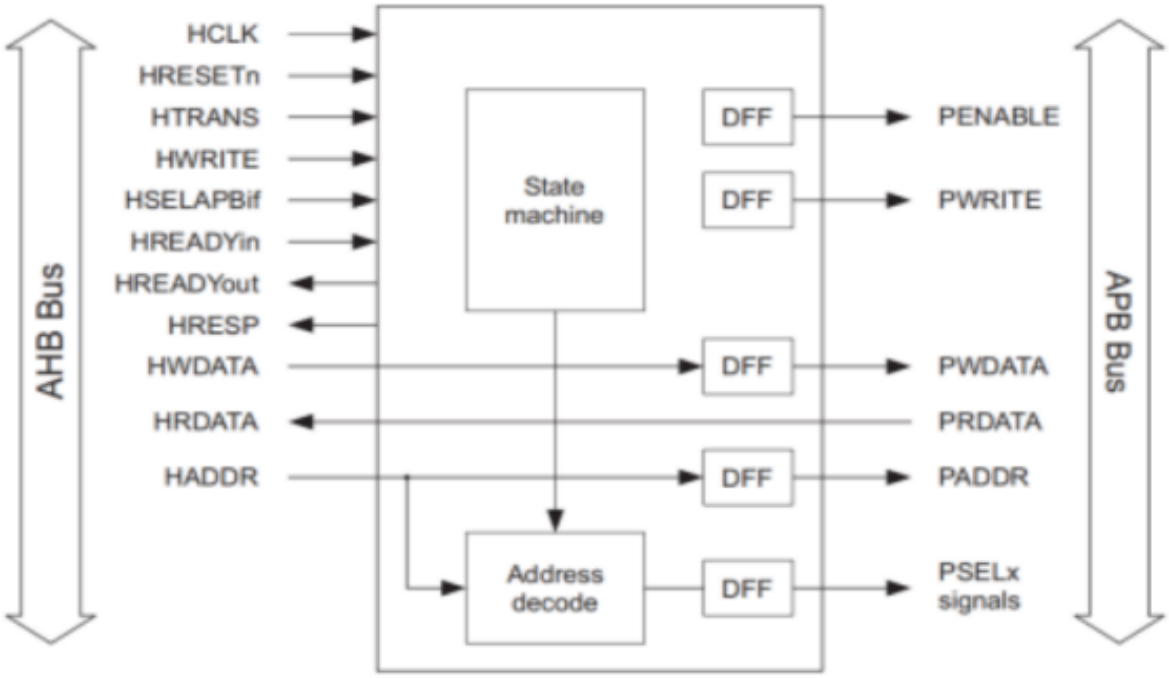


Fig1. AHB2APB Block Diagram

- AHB2APB bridge is an AHB SLAVE. It acts like an interface between high-speed AHB and low-power APB. The read and write transfers on AHB are converted into equivalent APB transfers and AHB is a pipelined protocol and APB is not pipelined. Therefore, for synchronization, this bridge adds WAIT states during the transfers to and from APB and AHB waits for APB.

3. Verification Requirements

3.1 Verification Levels

3.1.1 What hierarchy level are you verifying and why?

- **Hierarchy Level**: We are verifying the **top-level module** of the AHB-to-APB bridge design. This includes the integration of the AHB slave interface, APB FSM controller, and bridge logic.
- **Reason for Verification at This Level**:
  - The top-level module integrates all sub-modules (e.g., AHB slave interface, APB FSM controller) and ensures that they work together as intended.
  - Verifying at this level allows us to validate the end-to-end functionality of the AHB-to-APB bridge, including signal propagation, timing, and protocol compliance.
  - It ensures that the interactions between the AHB and APB protocols are correctly implemented and that the bridge meets the system-level requirements.

3.1.2 How is controllability and observability at the level you are verifying?

- **Controllability**:
  - At the top level, we have full control over the AHB signals (e.g., Hclk, Hresetn, Hwrite, Htrans, Haddr, Hwdata) and APB signals (e.g., Penable, Pwrite, Pselx, Paddr, Pwdata).
  - The testbench can drive all input signals to the DUT (Design Under Test) using a **driver** component, which allows us to create specific test scenarios (e.g., read/write transactions, error conditions).
  - The **generator** in the testbench can create randomized or directed transactions to cover various corner cases.

- **Observability**:
  - All output signals (e.g., Hrdata, Hresp, Hreadyout, Prdata) are monitored using a **monitor** component in the testbench.
  - The monitor captures the DUT's responses and forwards them to the **scoreboard** for comparison against expected results.
  - Internal signals (e.g., valid, Haddr1, Haddr2, Hwdata1, Hwdata2) can also be observed through the testbench to debug and verify intermediate states.

3.1.3 Are the interfaces and specifications clearly defined at the level you are verifying? List them.

- **Interfaces and Specifications**:
  The interfaces and specifications for the AHB-to-APB bridge are clearly defined at the top level. Below is a list of the key interfaces and their specifications:

1. **AHB Interface**:
   - **Signals**:
     - Hclk: AHB clock signal.
     - Hresetn: AHB active-low reset signal.
     - Hwrite: AHB write signal (1 = write, 0 = read).
     - Htrans: AHB transfer type (e.g., 2'b10 = NONSEQ, 2'b11 = SEQ).
     - Haddr: AHB address bus (32-bit).
     - Hwdata: AHB write data bus (32-bit).
     - Hrdata: AHB read data bus (32-bit).
     - Hresp: AHB response (e.g., 2'b00 = OKAY, 2'b01 = ERROR).
     - Hreadyout: AHB ready signal indicating the completion of a transfer.

- **Specifications**:
  - The AHB interface follows the AMBA AHB protocol specifications.
  - Supports single and burst transactions.
  - Handles address decoding and data transfer between AHB and APB.

2. **APB Interface**:
   - **Signals**:
     - Penable: APB enable signal.
     - Pwrite: APB write signal (1 = write, 0 = read).
     - Pselx: APB select signal (3-bit, selects the peripheral).
     - Paddr: APB address bus (32-bit).
     - Pwdata: APB write data bus (32-bit).
     - Prdata: APB read data bus (32-bit).
   - **Specifications**:
     - The APB interface follows the AMBA APB protocol specifications.
     - Supports single-cycle read/write transactions.
     - Handles data transfer between the bridge and APB peripherals.
3. **Internal Signals**:
   - **Signals**:
     - valid: Indicates a valid AHB transaction.
     - Haddr1, Haddr2: Pipeline stages for AHB address.
     - Hwdata1, Hwdata2: Pipeline stages for AHB write data.
     - Hwritereg: Registered version of Hwrite.
     - tempselx: Temporary select signal for APB peripherals.

- **Specifications**:
  - These signals are used internally for pipelining and synchronization between AHB and APB protocols.

4. **Functional Specifications**:
   - The bridge must correctly translate AHB transactions to APB transactions.
   - It must handle address decoding and peripheral selection.
   - It must ensure proper timing and protocol compliance for both AHB and APB interfaces.
   - It must handle error conditions (e.g., invalid addresses) and provide appropriate responses

4.Required Tools:

4.1Software Tools: Mentor graphics Questa-Sim, EDA Playground.

5. Risks and Dependencies

5.1:

1.Pipeline Logic Errors:

Risk: Incorrect propagation of address and data signals (Haddr1, Haddr2, Hwdata1, Hwdata2) within the AHB Slave Interface.
Mitigation: Implement assertions and functional coverage to validate pipeline behaviour. Conduct regression testing to ensure pipeline logic remains accurate.

2.State Machine Deadlocks:

Risk: The APB FSM Controller might enter a deadlock state due to faulty state transitions.
Mitigation: Introduce state transition checks and coverage points. Apply formal verification to validate critical transitions.

3.Address Decoding Errors:

Risk: Improper address decoding (tempselx) could result in selecting the wrong peripheral.

Mitigation: Perform boundary testing for address ranges and validate the tempselx logic.

4.Concurrent Transaction Handling:

Risk: The bridge may not correctly manage simultaneous read and write transactions.
Mitigation: Develop complex test scenarios for concurrent operations and ensure data integrity through verification.

5.Error Handling:

Risk: The bridge may fail to properly respond to invalid transactions, such as incorrect Htrans or Hburst values.
Mitigation: Inject error scenarios and verify that the bridge generates the appropriate error response (Hresp = 2'b01).

Contingency Plans

1. **Simulation Debugging:** Utilize waveform analysis to pinpoint and resolve issues in pipeline logic or state machine transitions during simulation.
2. **Assertions:** Implement assertions to detect and flag errors in real-time as the simulation runs, ensuring immediate identification of issues.
3. **Functional Coverage:** Ensure comprehensive testing by covering all critical scenarios, such as address boundary conditions and burst mode transactions, to validate the design thoroughly.
4. **Regression Testing:** Conduct regression tests after each modification to confirm that existing functionalities remain intact and no new issues are introduced.

6. Functions to be Verified

6.1 Functions from Specification and Implementation

6.1.1 List of Functions That Will Be Verified

Below is a list of functions that will be verified, along with a description of each function:

1. **AHB-to-APB Transaction Translation**:
   - **Description**: The bridge must correctly translate AHB read/write transactions into APB read/write transactions.
   - **Verification**: Verify that AHB transactions (NONSEQ, SEQ) are correctly mapped to APB transactions, including address and data propagation.
2. **Address Decoding**:
   - **Description**: The bridge must decode the AHB address and select the appropriate APB peripheral using the Pselx signal.
   - **Verification**: Verify that the address decoding logic correctly selects the APB peripheral based on the AHB address range.
3. **Write Operation**:
   - **Description**: The bridge must handle AHB write transactions and propagate the write data to the selected APB peripheral.
   - **Verification**: Verify that the write data (Hwdata) is correctly transferred to the APB peripheral and that the Pwrite signal is asserted.
4. **Read Operation**:
   - **Description**: The bridge must handle AHB read transactions and return the read data from the selected APB peripheral.
   - **Verification**: Verify that the read data (Prdata) is correctly returned to the AHB master and that the Hrdata signal is updated accordingly.
5. **Pipeline Handling**:
   - **Description**: The bridge must handle pipelined AHB transactions, including address and data pipelining.

- **Verification**: Verify that the pipeline stages (Haddr1, Haddr2, Hwdata1, Hwdata2) correctly store and propagate address and data.

6. **Error Handling**:
   - **Description**: The bridge must handle error conditions, such as invalid addresses or unsupported transactions, and return the appropriate AHB response (Hresp).
   - **Verification**: Verify that the bridge responds with Hresp = 2'b01 (ERROR) for invalid transactions.
7. **Protocol Compliance**:
   - **Description**: The bridge must comply with the AMBA AHB and APB protocols, including timing and signal behavior.
   - **Verification**: Verify that all AHB and APB signals adhere to the protocol specifications.
8. **Reset Behavior**:
   - **Description**: The bridge must correctly initialize all signals and states upon reset (Hresetn).
   - **Verification**: Verify that all internal registers and signals are reset to their default values when Hresetn is asserted.

6.1.2 List of Functions That Will Not Be Verified

Below is a list of functions that will **not** be verified, along with the reason for exclusion:

1. **Power Management Features**:
   - **Description**: The bridge does not include power management features such as clock gating or power-down modes.
   - **Reason for Exclusion**: Power management is not part of the current specification or implementation.
2. **Multi-Layer AHB Support**:
   - **Description**: The bridge does not support multi-layer AHB architectures or advanced AHB features like split transactions.

- **Reason for Exclusion**: The current design is a simple AHB-to-APB bridge and does not require these features.

3. **APB Peripheral-Specific Logic**:
   - **Description**: The bridge does not include logic specific to individual APB peripherals (e.g., UART, SPI).
   - **Reason for Exclusion**: The bridge is designed to be generic, and peripheral-specific logic is verified separately in the peripheral's testbench.
4. **Performance Optimization**:
   - **Description**: The bridge does not include performance optimization features such as burst mode or data buffering.
   - **Reason for Exclusion**: Performance optimization is not a requirement for the current design.

6.1.3 List of Critical and Non-Critical Functions for Tapeout

Below is a list of **critical** and **non-critical** functions for tapeout:

1. **Critical Functions**:
   - **AHB-to-APB Transaction Translation**: Ensures correct data transfer between AHB and APB.
   - **Address Decoding**: Ensures the correct peripheral is selected for each transaction.
   - **Write Operation**: Ensures write data is correctly propagated to the APB peripheral.
   - **Read Operation**: Ensures read data is correctly returned to the AHB master.
   - **Error Handling**: Ensures the bridge responds correctly to invalid transactions.
   - **Protocol Compliance**: Ensures the bridge adheres to AMBA AHB and APB protocols.
   - **Reset Behavior**: Ensures the bridge initializes correctly after reset.

Reason for Criticality: These functions are essential for the correct operation of the bridge and must be fully verified before tapeout.

2. **Non-Critical Functions**:
    - **Pipeline Handling**: While important, minor issues in pipelining may not prevent the bridge from functioning.
    - **Performance Optimization**: Not required for basic functionality.
    - **Power Management Features**: Not part of the current specification.

Reason for Non-Criticality: These functions are either optional or have a lower impact on the overall functionality of the bridge.

7. Test and Methods:

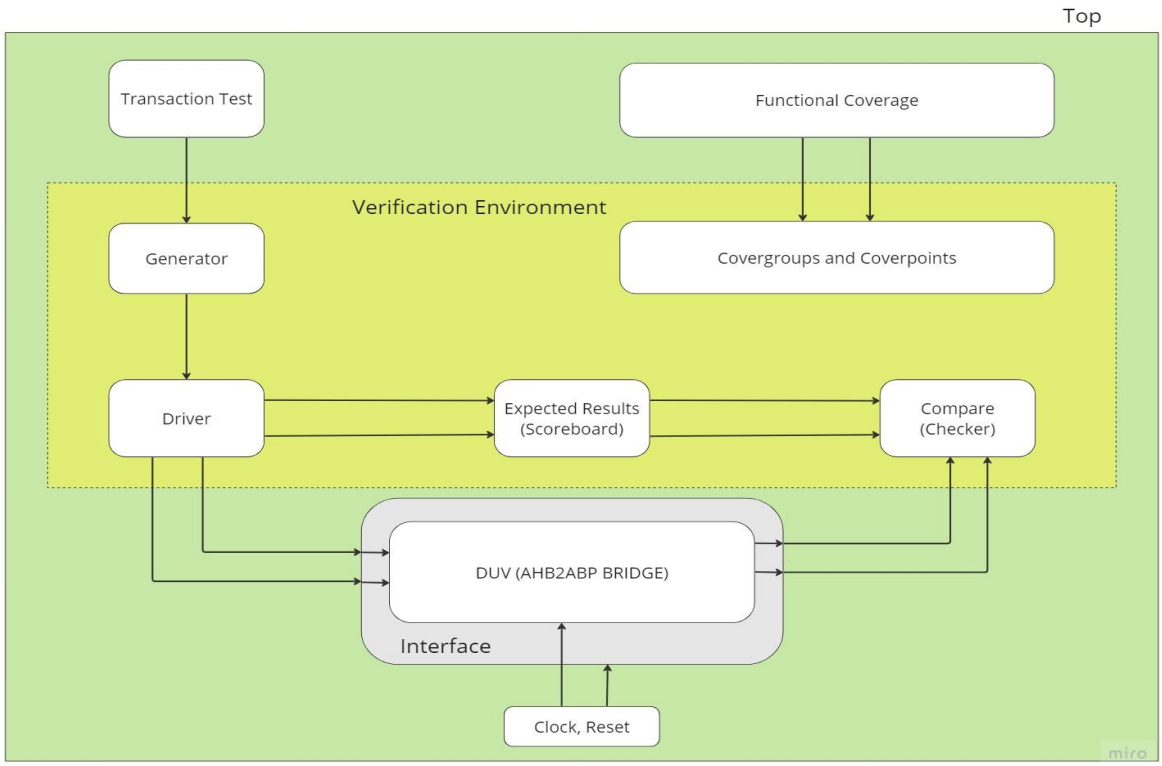
7.1.1 Testing methods to be used: Gray Box

7.1.2 Pro's and Con's why we have chosen this method:

- **Pro's:**
  - Partial Knowledge of Internal Logic: The testbench utilizes internal signals like valid, Hwritereg, and tempselx to validate interactions between modules such as the AHB Slave Interface and APB FSM Controller, ensuring thorough testing of critical paths.
  - Balanced Approach: Combines the benefits of black box and white box testing, enabling both system-level functionality checks and targeted internal validation.
- **Con's:**
  - Limited Visibility: Only specific internal signals are accessible, meaning some aspects, such as complete state machine transitions, may not be fully covered.
  - Higher Complexity: Requires an understanding of internal structures, making test case development more intricate compared to black box testing.

Gray box testing is the most suitable approach for this DUV as it enables the testbench to validate complex interactions between the AHB and APB interfaces, such as pipeline logic and state transitions, while also verifying overall bridge functionality. This method strikes a balance between in-depth testing and system-level validation, making it the ideal choice for this project.

7.1.3 Test Bench Architecture:



➢ **AHB_Master:**

- AHB is a new generation of AMBA bus which is intended to address the requirements of high-performance synthesizable designs.
- It is a new level of bus that sits above APB and implements the features required for high-performance and high-clock frequency systems such as:
  i. Burst functions.
  ii. Split transactions.
  iii. Single-cycle bus MASTER handover.
  iv. Single clock edge operation.
  v. Non-tristate implementation and Wider dat.
  vi. Larger overhead of approximately 27 control signals, up to 75 MASTERs are allowed.
  vii. Split transaction phases: Address, data (Pipelined), HREADY signal allows insertion of wait-states.
- It generates read and write transactions for the AHB-to-APB bridge. It provides tasks for single read and write operations.

- **AHB Slave:**
  It implements the AHB Slave Interface, responsible for handling AHB transactions and forwarding them to the APB FSM Controller.
- APB:

- It is a low-power, low-bandwidth bus in the AMBA protocol suite, designed for connecting peripherals in a system on chip.
- It implements the APB FSM Controller, responsible for controlling the APB signals (PSELx, PWRITE, PENABLE) based on inputs from the AHB Slave Interface and ensuring proper handshaking with the APB peripheral.
- APB Interface handles the interaction between the APB FSM Controller and the APB peripherals. It includes logic for read and write operations on the APB side.

Components Used: Transaction, Generator, Driver, Monitor, Scoreboard, Environment, Bridge, Interface.

i. **Transaction:** The Transaction class encapsulates the characteristics and behavior of transactions within the AHB-APB Bridge. It includes essential details such as addresses, data, transaction type, and other parameters required for AHB and APB protocols. Constraints are applied to ensure the generation of valid transactions. Additionally, the class provides methods to determine the transaction type based on whether it is a read or write operation, display transaction details, and define a cover-group for coverage collection. This coverage monitors various operations, sizes, and burst types, ensuring thorough verification.

ii. **Generator:** The generator class in System Verilog generates and transmits randomized transactions to a driver using a mailbox. It handles both **write** and **read** operations, with Hsize, Hburst, and Htrans being randomized. In the write task, a random address and data are assigned, whereas the read task assigns only a random address. Debug messages help identify whether the transaction is **sequential or non-sequential**, assisting in verification.

iii. **Driver:** The System Verilog driver class receives transactions from the generator and drives them into the Design Under Verification (DUV) using a virtual interface (vif) for seamless interaction. It manages transaction handles and multiple mailboxes to regulate data flow: gen2driv receives transactions from the generator, driv2sb forwards them to the scoreboard for verification, and driv2cor sends them directly to the DUV. The core

functionality resides in the drive task, which retrieves transactions via gen2driv.get(tx), transmits them to both the scoreboard and the DUV through driv2sb.put(tx) and driv2cor.put(tx), and then assigns transaction values to the DUV signals via the vif. To ensure synchronization, the driver waits for a clock edge before processing the next transaction, maintaining accurate and reliable communication between verification components.

iv. **Monitor:** The monitor class plays a vital role in the SystemVerilog testbench by observing the interface, capturing transactions on the bus, and forwarding them to the scoreboard for comparison with expected results. As a passive component, It acts as a listener, ensuring non-intrusive monitoring of the Design Under Test (DUT). In this setup, the monitor continuously observes the signals on the AHB-APB bridge interface, creates transaction objects based on the detected activity, and sends them to the scoreboard for verification. Operating in an infinite loop, it constantly monitors the interface for new transactions. To maintain synchronization, the monitor leverages a clocking block (mon_cb), which ensures that signals are sampled accurately with the clock. The captured values are then used to construct a transaction object, which is subsequently transmitted to the scoreboard via a mailbox for further processing.

v. **Scoreboard:** The Scoreboard class plays a crucial role in verification process as it validates the correctness of the design. It contains a memory model that mimics the behavior of the design under test (DUT). The Scoreboard receives transactions from both the driver and the monitor, allowing it to compare the expected and actual responses. This class has methods to handle both data write and read operations. For a write operation, it verifies that the data has been correctly written into the memory model. For a read operation, it checks if the data read from the DUT matches with the data stored in the memory model. Any discrepancy in data would result in an assertion error, flagging a failure in the verification process.

vi. **Environment:** The environment class plays a crucial role in the System Verilog testbench, serving as the core of the verification process. It integrates key verification components such as the generator, driver, monitor, and scoreboard, ensuring seamless interaction between them. In this setup, the environment establishes mailboxes for communication,

initializes all components, and oversees the execution of specific test cases. These test cases simulate various transactions that the AHB-APB bridge must handle, enabling thorough testing and verification of the design under test (DUT).

vii. **Interface:** The System Verilog interface encapsulates the AHB-APB Bridge protocol signals, providing a centralized handle for efficient management and facilitating communication between verification components like the driver, monitor, and DUT. It defines two clocking blocks, drv_cb for the driver and mon_cb for the monitor ensuring precise control over signal driving and sampling, which is crucial for accurate design verification. The drv_cb clocking block enables the driver to correctly transmit signals to the DUT, while mon_cb allows the monitor to sample signals from the DUT, ensuring synchronized operation. Additionally, the interface includes modports, with the master modport assigned to the driver for signal driving and the slave modport designated for the monitor to sample signals, enabling structured and efficient interaction.

7.1.4 Verification Strategy: Dynamic Simulation

Dynamic Simulation is the chosen verification strategy for this project as it enables real-time testing of the AHB to APB bridge through clock-driven transactions, aligning with both the testbench structure and the DUV's operation. This approach supports randomized testing and functional coverage, ensuring validation across a diverse set of scenarios. It is particularly effective for module-level verification, as it focuses on analyzing the DUV's behavior over time in response to dynamic inputs, making it the most suitable method for this project.

7.1.5 Driving Methodology: Constraint Random Verification

7.1.5.1 Testing Methods:

1. **Randomized Test Generation:**

   The testbench generator produces randomized AHB transactions, including random values for addresses (Haddr), data (Hwdata), burst types (Hburst), and transfer types (Htrans). This serves as a fundamental aspect of constrained random verification, ensuring diverse test scenarios.

2. **Constraints:**

   Randomization is controlled to generate valid and meaningful test cases. For instance:

- Addresses are restricted to specific ranges to ensure access to valid memory regions.
- Burst types are constrained to permissible AHB burst modes (e.g., INCR, WRAP4, INCR4, etc.).
- Transfer types are limited to valid AHB transfer types (e.g., NON-SEQ, SEQ) to maintain protocol compliance.

3. **Functional coverage:**

   The testbench incorporates functional coverage points to monitor the scenarios that have been tested.

4. **Driver & Monitor:**

   The driver sends the randomized transactions to the DUV, while the monitor monitors the responses on the APB interface. This closed-loop feedback guarantees that the randomized tests are properly validated.

7.1.6 Checking Methodology: Implementation based checking

1. **Scoreboard and Memory Module:**

The scoreboard utilizes a memory model (mem_tb) to track both write and read transactions. It compares the data written to the memory model with the data read from the DUV, verifying correctness according to the implementation.

2.Monitor and Driver:

The monitor watches the DUV's responses and forwards them to the scoreboard for comparison. The driver applies transactions to the DUV, ensuring the implementation functions as intended

3.Functional Coverage:

The testbench incorporates functional coverage points (e.g., Hwrite, Htrans, Hburst) to verify that the implementation is tested across all defined scenarios.

7.1.7 Test Scenarios:

7.1.7.1: Basic Test

Test Name / Number	Test Description/ Features
1.1.1	**Check basic read operation:** Ensure that the AHB Master can successfully perform a single read from the APB peripheral. The test checks that the correct data is returned from the given address (Haddr = 32'h8000_00A2).
1.1.2	**Check basic write operation:** Ensure that the AHB Master can successfully perform a single write to the APB peripheral. The test confirms that the data (Hwdata = 8'hA3) is accurately written to the specified address (Haddr = 32'h8000_0001).

7.1.7.2 Complex Test:

Test Name / Number	Test Description/ Features
1.2.1	**Concurrent events (R+W):** Ensure that the AHB Master can handle simultaneous read and write operations. The test verifies that the bridge processes consecutive read and write transactions correctly, without data corruption or protocol violations
1.2.2	**Burst mode transactions:** Ensure that the AHB Master can handle burst transactions (e.g., INCR4, WRAP8) and that the APB FSM Controller processes these transactions correctly. The test confirms that all data within the burst is transferred accurately.

7.1.7.3 Regression Test:

Test Name / Number	Test Description/Features
1.3.1	**Single read and write operations:** Ensure that basic read and write operations consistently pass. This serves as a sanity check to confirm the bridge is working properly after any modifications.
1.3.2	**Pipeline functionality**: Ensure that the pipeline logic in the AHB Slave Interface properly propagates address and data signals (Haddr1, Haddr2, Hwdata1, Hwdata2) without any errors.

7.1.7.4 Special or Corner Cases:

Test Name / Number	Test Description
1.4.1	Address boundary testing: Ensure that the bridge correctly handles transactions at the boundaries of valid address ranges (e.g., Haddr = 32'h8000_0000 and Haddr = 32'h8BFF_FFFF). The test confirms that the valid signal is properly generated for boundary addresses.
1.4.2	Error injection testing: Introduce errors (e.g., invalid Htrans or Hburst values) and verify that the bridge generates the correct error signal (Hresp = 2'b01 for

	ERROR). This ensures the bridge handles invalid transactions properly.

8.Coverage Requirements:

Coverage makes sure all the key features of he AHB-to-APB bridge are tested and are working to the best of their ability. Below are the main coverage areas.

8.1 Assertion Coverage

- Protocol Compliance to ensure that the transaction follow AHB and APB rules
- Response timing to check if transactions complete within an expected timeframe
- Error detection to ensure incorrect transaction trigger proper error responses

8.2 Functional Coverage

- Transaction coverage for read and write operations
- Address Ranges to ensure al valid address can be accessed
- Test data patterns for various formats like all 1’s or 0’s
- Pipeline handling to ensure correct pipelining of address and data
- Test invalid commands for error handling

8.3 Code Coverage

- Statement coverage to make sure all the lines of the code execute
- Branch coverage for the paths
- Toggle coverage to ensure all signals change states

8.4 Cross Coverage

- Transaction types vs Htrans to check read/write operations
- Transaction types vs Hsize to ensure all data sizes are used
- Transaction types vs Hbursts to cover different burst types

8.5 Validation Environment Coverage

- Test sequence execution to ensure all test scenarios run at least once
- Checker and scoreboard usage to verify expected vs actual results are compared correctly.
- Randomization coverage to see if test stimulus covers diverse cases

9. Resource Requirements:

9.1 **Team members and who is doing what and expertise:**

Daniel Jacobsen	M1: Compiled the code and ran the simulation. M2 & M3: Worked on Code and Functional Coverage.
Balaji Ginkal Harisha	M1: Designed the verification plan. M2 & M3: Worked on Transaction and Generator.
Mohith Kumar Bennahatti Chikkegowda	M1: Documented the specifications of the AHB2APB bridge. M2 & M3: Worked on Driver and Monitor.
Akshaya Kudumalakunte RaviKumar	M1: Obtained the resources and chose the code for the project. M2 & M3: Worked on Environment and Scoreboard.

9.2: Computing:

We need a sufficient computing resource to run our project's simulations, like a computer with sufficient processing power and a memory to simulate in Questasim and which should be handle our design and testbenches.

10. Simulation Results:

10.1: Transcript:

```
[SCOREBOARD] - Input Address: 48d91
[SCOREBOARD] - Input Write Data: 61ca2891
[SCOREBOARD] - Data Stored: 61ca2891
[SCOREBOARD] - WRITE = DATA MATCHED

[GENERATOR] - Generated Transaction:
[GENERATOR] - Haddr: 647815cf, Hwdata: 3b515c3d
[GENERATOR] - Hburst: 001 (INCR)
[GENERATOR] - Htrans: 11 (SEQ)
[SCOREBOARD] - Checking Write Transaction:
[GENERATOR] - Generated Transaction:
[GENERATOR] - Haddr: d48d898d, Hwdata: 17868ba7
[GENERATOR] - Hburst: 010 (WRAP4)
[GENERATOR] - Htrans: 11 (SEQ)
[SCOREBOARD] - Checking Write Transaction:
[MONITOR] - Observed Transaction:
[MONITOR] - Haddr: 27248d91, Hwdata: 61ca2891, Prdata: 00000000
[MONITOR] - Hburst: 000 (UNKNOWN)
[MONITOR] - Htrans: 10 (NON-SEQ)
[MONITOR] - Observed Transaction:
[MONITOR] - Haddr: 27248d91, Hwdata: 61ca2891, Prdata: 00000000
[MONITOR] - Hburst: 000 (UNKNOWN)
[MONITOR] - Htrans: 10 (NON-SEQ)
[MONITOR] - Observed Transaction:
[MONITOR] - Haddr: 27248d91, Hwdata: 61ca2891, Prdata: 00000000
[MONITOR] - Hburst: 000 (UNKNOWN)
[MONITOR] - Htrans: 10 (NON-SEQ)
[SCOREBOARD] - Input Address: d898d
[SCOREBOARD] - Input Write Data: 17868ba7
[SCOREBOARD] - Data Stored: 17868ba7
[SCOREBOARD] - WRITE = DATA MATCHED

[SCOREBOARD] - Input Address: d898d
[SCOREBOARD] - Input Write Data: 17868ba7
[SCOREBOARD] - Data Stored: 17868ba7
[SCOREBOARD] - WRITE = DATA MATCHED

[GENERATOR] - Generated Transaction:
[GENERATOR] - Haddr: e24cc359, Hwdata: b23263ca
[GENERATOR] - Hburst: 011 (INCR4)
[GENERATOR] - Htrans: 10 (NON-SEQ)
[SCOREBOARD] - Checking Read Transaction:
[SCOREBOARD] - Temp address = cc359
[SCOREBOARD] - Read data from DUT: 00000000
[SCOREBOARD] - Data from TB memory: 00000000
[SCOREBOARD] - READ = DATA MATCHED
```

10.2: Coverage Results:

```
================================================================================
=== Instance: /top_sv_unit
=== Design Unit: work.top_sv_unit
================================================================================

Assertion Coverage:
    Assertions                       2         2         0   100.00%
----------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
----------------------------------------------------------------------
/top_sv_unit/ahb_apb_scoreboard/data_write/immed__58
                     scoreboard.sv(58)                  0          1
/top_sv_unit/ahb_apb_scoreboard/data_read/immed__79
                     scoreboard.sv(79)                  0          1

Covergroup Coverage:
    Covergroups                      1        na        na    80.95%
        Coverpoints/Crosses          7        na        na        na
            Covergroup Bins         38        26        12    68.42%
----------------------------------------------------------------------------------------------------
Covergroup                                          Metric       Goal       Bins    Status

----------------------------------------------------------------------------------------------------
 TYPE /top_sv_unit/Transaction/cov_cg                80.95%        100          -    Uncovered
    covered/total bins:                                 26         38          -
    missing/total bins:                                 12         38          -
    % Hit:                                          68.42%        100          -
    Coverpoint Hwrite_cp                           100.00%        100          -    Covered
        covered/total bins:                               2          2          -
        missing/total bins:                               0          2          -
        % Hit:                                      100.00%        100          -
        bin read                                         52          1          -    Covered
        bin write                                        48          1          -    Covered
    Coverpoint Htrans_cp                           100.00%        100          -    Covered
        covered/total bins:                               3          3          -
        missing/total bins:                               0          3          -
        % Hit:                                      100.00%        100          -
        bin non_seq                                      28          1          -    Covered
        bin idle                                         22          1          -    Covered
        bin seq                                          16          1          -    Covered
    Coverpoint Hsize_cp                            100.00%        100          -    Covered
        covered/total bins:                               3          3          -
        missing/total bins:                               0          3          -
```

.....

10k lines of other code

.....

```
    Cross Hwrite_x_hburst                              0.00%        100          -    ZERO
        covered/total bins:                                0         12          -
        missing/total bins:                               12         12          -
        % Hit:                                         0.00%        100          -
        Auto, Default and User Defined Bins:
            bin <*,*>                                      0          1         12    ZERO

TOTAL COVERGROUP COVERAGE: 80.95%  COVERGROUP TYPES: 1

ASSERTION RESULTS:
--------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
--------------------------------------------------------------------
/top_sv_unit/ahb_apb_scoreboard/data_write/immed__58
                     scoreboard.sv(58)                    0          1
/top_sv_unit/ahb_apb_scoreboard/data_read/immed__79
                     scoreboard.sv(79)                    0          1

Total Coverage By Instance (filtered view): 90.47%

End time: 21:57:04 on Feb 18,2025, Elapsed time: 0:00:00
Errors: 0, Warnings: 0
```

10.3: Top Level Module:

```
`include "transactions.sv"
`include "generator.sv"
`include "interface.sv"
`include "driver.sv"
`include "monitor.sv"
`include "scoreboard.sv"
`include "coverage.sv"
`include "environment.sv"
`include "test.sv"
`include "AHB_Slave_Interface.sv"
`include "APB_Controller.sv"
`include "Bridge_Top.sv"

module ahb_apb_top;

  logic clk, reset;

  // Generates clk with a time period of 5 ns
  always
  begin
    forever begin
```

```
    #5 clk = ~clk;
  end
 end



 ahb_apb_bfm_if bfm(clk, reset); // Connect clock and reset


  // Connecting DUT signals with signals present on the interface
// Connecting DUT signals with signals present on the interface
Bridge_Top dut(
   .Hclk(bfm.clk),
   .Hresetn(bfm.resetn),
   .Hwrite(bfm.Hwrite),
   .Hreadyin(bfm.Hreadyin),
   .Htrans(bfm.Htrans),
   .Hwdata(bfm.Hwdata),
   .Haddr(bfm.Haddr),
   .Hrdata(bfm.Hrdata),
   .Hresp(bfm.Hresp),
   .Hreadyout(bfm.Hreadyout),
   .Prdata(bfm.Prdata),
   .Pwdata(bfm.Pwdata),
   .Paddr(bfm.Paddr),
   .Pselx(bfm.Pselx),
   .Pwrite(bfm.Pwrite),
   .Penable(bfm.Penable)
);


  // test ahb_apb_test(bfm); // -> not initialized
   test test_h;
   Transaction trans;
 initial begin
   $display("in top");
   trans = new();
   //trans.cov_cg.sample();  // -> to get the coverage
 test_h = new(bfm);
 test_h.run();
 end

 // Initialize clk and reset
 initial begin
   clk = 1;
   reset = 0;
```

```
    #10
    reset = 1;
    #100000;
    $stop; // Stops simulation
  end

endmodule
```

11. References Used/Citation:

1. https://github.com/prajwalgekkouga/AHB-to-APB-Bridge
2. https://github.com/Ghonimo/Pre_Silicon-AHB-to_APB-Verification
3. https://github.com/Siddhi-95/AHB-to-APB-Bridge-Verification/tree/main