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

Date: 01/31/2025

1. 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.

1.1: 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.

1.2 Top Level block diagram:

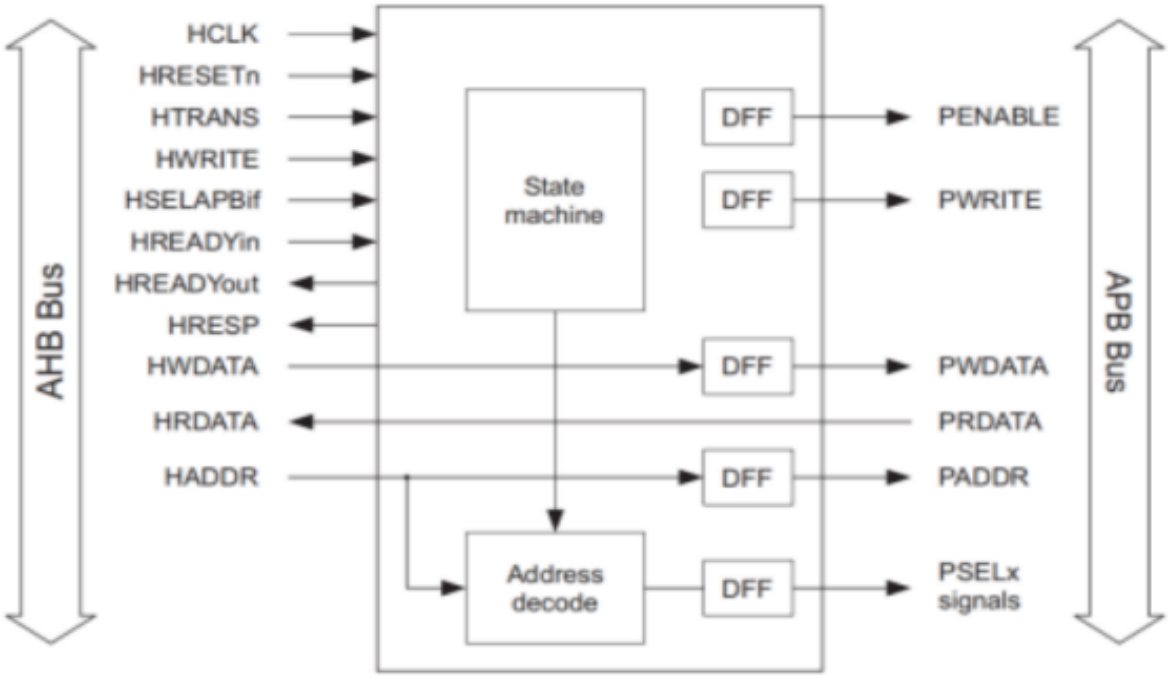


Fig1. AHB2APB Block Diagram

- 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.

2.Verification Test methods:

2.1.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.

2.2 AHB Slave:

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

2.3 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.

2.4. Test Scenarios:

- Implemented a directed testbench written in system Verilog using procedural test sequences to manually drive and verify transactions on the AHB2APB bridge.
- It is mainly verifying the functionality of the AHB to APB Bridge. It includes tasks for generating single read and write transactions, as well as checking the correctness of data flow between AHB and APB.
- Single Write: Verify that data is written to the APB peripheral correctly.
- Single Read: Verify that data is read back correctly from the APB peripheral. The testbench uses self-checking assertions to validate functionality.
- Basic Score boarding: It checks within single write and single read function as a simple scoreboard by validating read/write correctness.

3.Required Tools:

3.1 Software Tools: Mentor graphics Questasim, EDA Playground.

4. Resource Requirements:

4.1. People

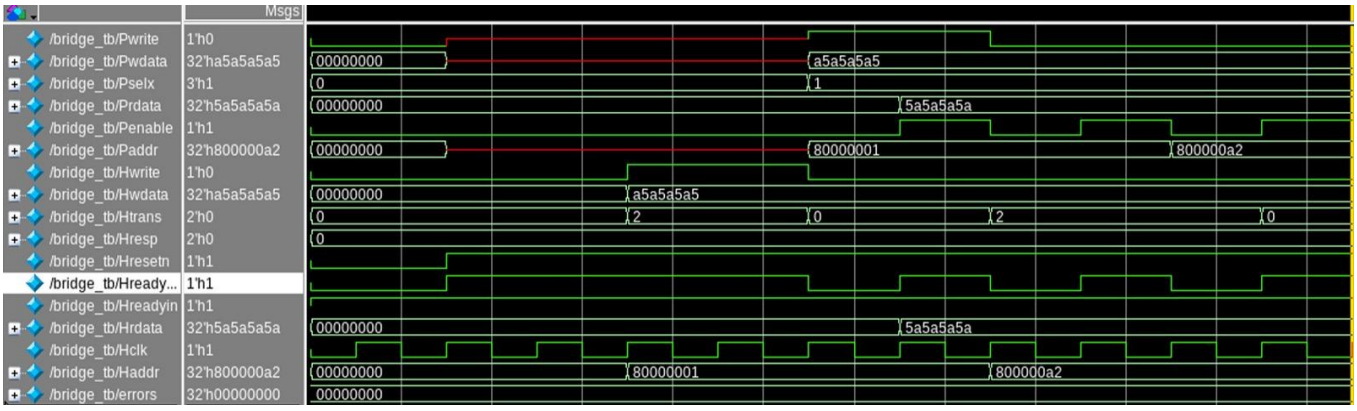
Daniel Jacobsen	Compiled the code and ran the simulation.
Balaji Ginkal Harisha	Designed the verification plan.
Mohith Kumar Bennahatti Chikkegowda	Documented the specifications of the AHB2APB bridge.
Akshaya Kudumalakunte RaviKumar	Obtained the resources and chose the code for the project.

4.2: 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.

5. Simulation Results:

5.1: Waveform:



5.2: Transcript:

```
run -all
#
# Starting Bridge Tests...
#
#                    65 Write matched: Pwrite=1, Paddr=0x80000001, Pwdata=0xa5a5a5a5
#                   115 Read matched: Hrdata=0x5a5a5a5a
#
# Testbench PASSED with no errors!
#
# ** Note: $finish    : /u/dj23/ECE593/Final Project/Testbench.sv(218)
#    Time: 115 ns  Iteration: 1  Instance: /bridge_tb
# 1
# Break in Module bridge_tb at /u/dj23/ECE593/Final Project/Testbench.sv line 218
```

6. The conventional testbench created to check the basic read and write operations is:

```
module bridge_tb;

  // Clock and reset
  logic Hclk;
  logic Hresetn;

  // AHB signals
  logic Hwrite;
  logic Hreadyin;
  logic [31:0] Hwdata;
  logic [31:0] Haddr;
  logic [1:0]  Htrans;
  logic [1:0]  Hresp;
  logic [31:0] Hrdata;
  logic Hreadyout;

  // APB signals
  logic Penable;
  logic Pwrite;
  logic [2:0] Pselx;
  logic [31:0] Paddr;
  logic [31:0] Pwdata;
  logic [31:0] Prdata;

  // DUT instantiation
  Bridge_Top bridge_top (
     .Hclk       (Hclk),
     .Hresetn    (Hresetn),
     .Hwrite     (Hwrite),
     .Hreadyin   (Hreadyin),
     .Hwdata     (Hwdata),
     .Haddr      (Haddr),
     .Htrans     (Htrans),
     .Prdata     (Prdata),
     .Hreadyout  (Hreadyout),
     .Hresp      (Hresp),
     .Hrdata     (Hrdata),
     .Penable    (Penable),
     .Pwrite     (Pwrite),
     .Pselx      (Pselx),
     .Paddr      (Paddr),
     .Pwdata     (Pwdata)
```

```
  );

  // Variables for the testbench
  integer errors = 0;

  // Clock generation (10 ns period)
  initial begin
    Hclk = 0;
    forever #5 Hclk = ~Hclk;
  end

  // Reset generation
  initial begin
    Hresetn = 0;
    #15;      // Hold reset low for 15 ns
    Hresetn = 1;
  end

  /* USED IN EDA PLAYGROUND TESTING
  // Waveform dump
  initial begin
    $dumpfile("bridge_tb.vcd");
    $dumpvars(0, bridge_tb);
  end
*/

  // Tasks to model AHB transactions


  // Single WRITE transaction
  task automatic do_single_write(
    input [31:0] addr,
    input [31:0] data
  );
    begin
      // 1) Drive address & data, set Htrans=NONSEQ, set Hwrite=1
      @(posedge Hclk);
      Haddr    <= addr;
      Hwdata   <= data;
      Hwrite   <= 1;
      Htrans   <= 2'b10; // NONSEQ
      Hreadyin <= 1;

      // 2) Wait for the bridge to respond (if the bridge deasserts Hreadyout, wait until it is
re-asserted)
      //    Some designs keep Hreadyout=1 all the time; if so, you can just wait 1 or 2 cycles.
```

```
    @(posedge Hclk);
    while (!Hreadyout) @(posedge Hclk);

    // 3) Return to idle
    @(posedge Hclk);
    Htrans <= 2'b00;
    Hwrite <= 0;
  end
endtask

// Check the APB signals for that single WRITE
// (Call this after do_single_write finishes, plus maybe 1 cycle.)
task automatic check_single_write(
  input [31:0] expectedAddr,
  input [31:0] expectedData
);
  begin
    // Wait a cycle so the pipeline can update APB signals
    @(posedge Hclk);
    // Compare with Paddr, Pwdata, Pwrite
    if (Pwrite !== 1'b1 || Paddr !== expectedAddr || Pwdata !== expectedData) begin
      $display($time, " *** ERROR: Write mismatch. ");
      $display("    Pwrite=%b, Paddr=0x%08h, Pwdata=0x%08h",
               Pwrite, Paddr, Pwdata);
      errors++;
    end else begin
      $display($time, " Write matched: Pwrite=%b, Paddr=0x%08h, Pwdata=0x%08h",
               Pwrite, Paddr, Pwdata);
    end
  end
endtask

// Single READ transaction
task automatic do_single_read(
  input [31:0] addr
);
  begin
    @(posedge Hclk);
    Haddr    <= addr;
    Hwrite   <= 0;
    Htrans   <= 2'b10;  // NONSEQ
    Hreadyin <= 1;

    // Wait for Hreadyout
    @(posedge Hclk);
    while (!Hreadyout) @(posedge Hclk);
```

```
    // Return to IDLE
    @(posedge Hclk);
    Htrans <= 2'b00;
  end
endtask

// Check the read data returned on Hrdata
task automatic check_single_read(
  input [31:0] expectedData
);
  begin
    // Wait a cycle or two for pipeline
    @(posedge Hclk);

    if (Hrdata !== expectedData) begin
      $display($time, " *** ERROR: Read mismatch. Hrdata=0x%08h (expected
0x%08h)",
               Hrdata, expectedData);
      errors++;
    end else begin
      $display($time, " Read matched: Hrdata=0x%08h", Hrdata);
    end
  end
endtask


// Main Test Sequence

initial begin
  // Initialize signals
  Hwrite   = 0;
  Hreadyin = 1;
  Haddr    = 32'b0;
  Hwdata   = 32'b0;
  Htrans   = 2'b00;
  Prdata   = 32'b0;

  // Wait until reset is de-asserted
  @(posedge Hclk);
  wait (Hresetn === 1);

  @(posedge Hclk);
  $display("\nStarting Bridge Tests...\n");

  // TEST 1: Single Write
```

```
    do_single_write(32'h8000_0001, 32'hA5A5A5A5);
    check_single_write(32'h8000_0001, 32'hA5A5A5A5);

    // TEST 2: Single Read
    // We'll set Prdata to be returned by the APB side:
    Prdata = 32'h5A5A5A5A;
    do_single_read(32'h8000_00A2);
    check_single_read(32'h5A5A5A5A);

    // End of tests
    if (errors == 0)
      $display("\nTestbench PASSED with no errors!\n");
    else
      $display("\nTestbench FAILED with %0d errors!\n", errors);

    $finish;
  end

endmodule
```