Maseeh College of Engineering and Computer Science
PORTLAND STATE UNIVERSITY

ECE 593: SUMMER 2023 - CLASS PROJECT
FUNDAMENTALS OF PRE-SILICON VALIDATION

VERIFICATION OF AN AHB2APB BRIDGE

CHECKPOINT 1
INITIAL PROJECT PROPOSAL

MOHAMED GHONIM
GAYATRI VEMURI
07/05/2023

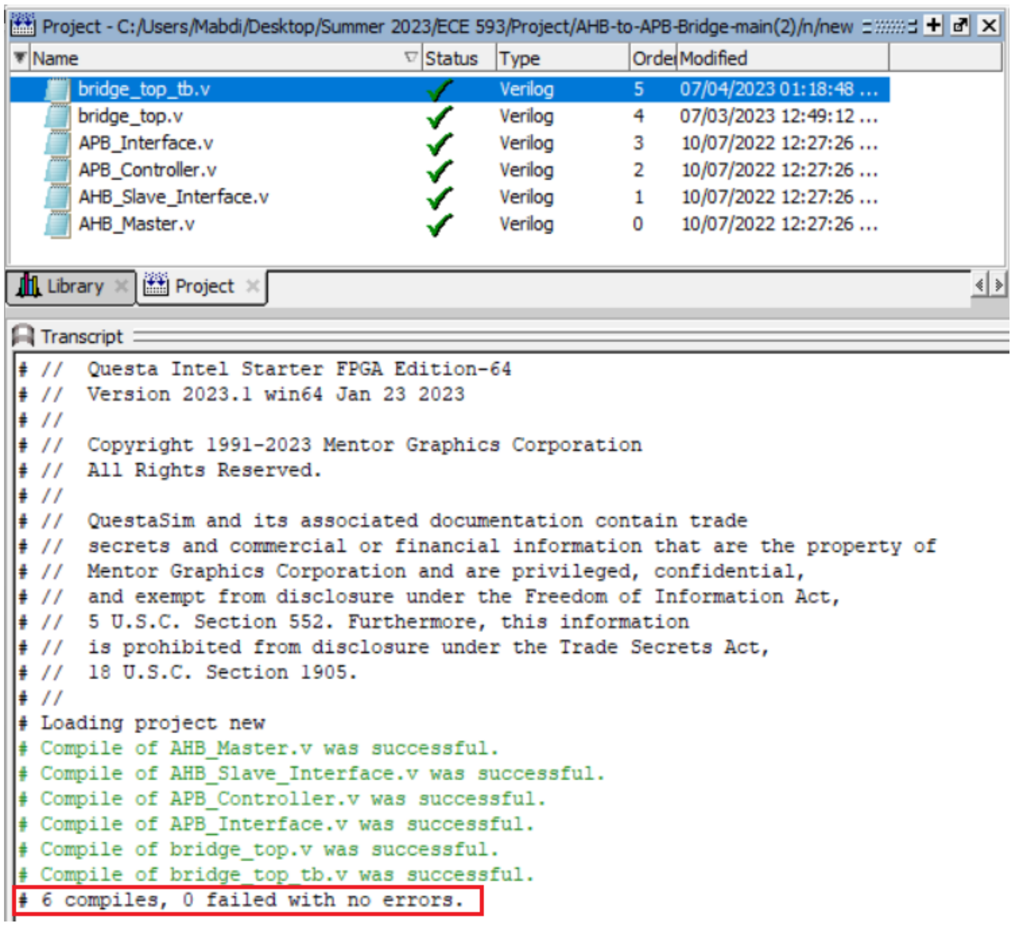
Table of Contents

Introduction

Our project focuses on the verification and validation of the AHB2APB bridge design. In order to achieve this objective, we plan to adopt System Verilog and UVM-based test bench development methodologies to create Verification IP (VIP) components for the design. The purpose of this initial phase is to establish the simulation environment and validate the basic functionality of the AHB2APB bridge.

To initiate our project, we selected an RTL code for the AHB2APB bridge design, which is included as an attachment to this submission and referenced in the appendix. The chosen RTL code can be found at: https://github.com/prajwalgekkouga/AHB-to-APB-Bridge

In order to ensure the basic correctness of chosen code, we utilized QuestaSim to compile the design files, and the compilation process was successful without errors or issues.

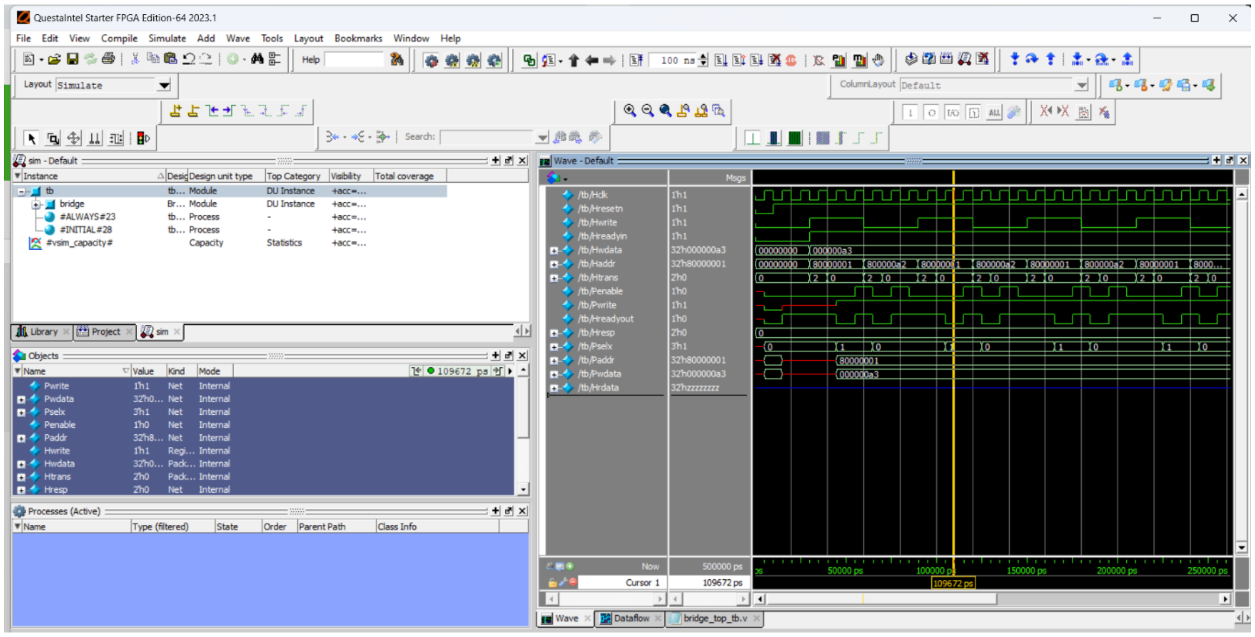


Project - C:/Users/Mabdi/Desktop/Summer 2023/ECE 593/Project/AHB-to-APB-Bridge-main(2)/n/new

Name	Status	Type	Order	Modified
bridge_top_tb.v	✓	Verilog	5	07/04/2023 01:18:48 ...
bridge_top.v	✓	Verilog	4	07/03/2023 12:49:12 ...
APB_Interface.v	✓	Verilog	3	10/07/2022 12:27:26 ...
APB_Controller.v	✓	Verilog	2	10/07/2022 12:27:26 ...
AHB_Slave_Interface.v	✓	Verilog	1	10/07/2022 12:27:26 ...
AHB_Master.v	✓	Verilog	0	10/07/2022 12:27:26 ...

```
# //  Questa Intel Starter FPGA Edition-64
# //  Version 2023.1 win64 Jan 23 2023
# //
# //  Copyright 1991-2023 Mentor Graphics Corporation
# //  All Rights Reserved.
# //
# //  QuestaSim and its associated documentation contain trade
# //  secrets and commercial or financial information that are the property of
# //  Mentor Graphics Corporation and are privileged, confidential,
# //  and exempt from disclosure under the Freedom of Information Act,
# //  5 U.S.C. Section 552. Furthermore, this information
# //  is prohibited from disclosure under the Trade Secrets Act,
# //  18 U.S.C. Section 1905.
# //
# Loading project new
# Compile of AHB_Master.v was successful.
# Compile of AHB_Slave_Interface.v was successful.
# Compile of APB_Controller.v was successful.
# Compile of APB_Interface.v was successful.
# Compile of bridge_top.v was successful.
# Compile of bridge_top_tb.v was successful.
# 6 compiles, 0 failed with no errors.
```

In addition, we designed our own basic testbench simulation to observe the waveform generated by the design, and this screenshot is attached below.


QuestaIntel Starter FPGA Edition-64 2023.1
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout Default
sim - Default
Library
Project
sim
Wave - Default
Wave
Dataflow
bridge_top_tb.v

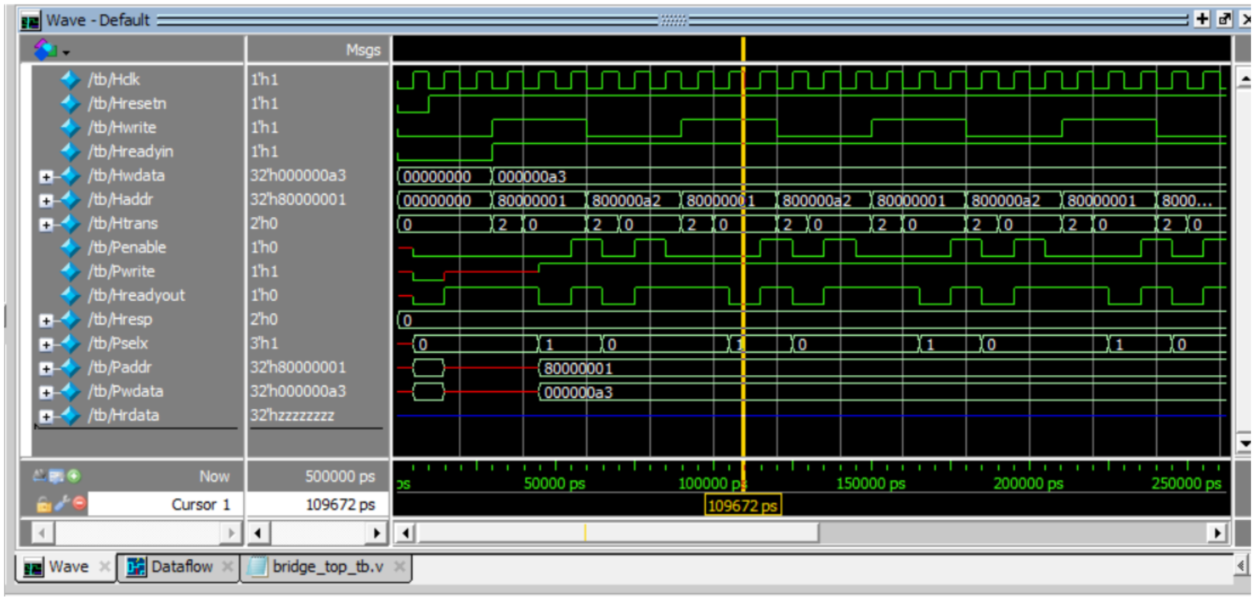
Instance	Desig	Design unit type	Top Category	Visibility	Total coverage
tb	tb…	Module	DU Instance	+acc=…	
bridge	Br…	Module	DU Instance	+acc=…	
#ALWAYS#23	tb…	Process	-	+acc=…	
#INITIAL#28	tb…	Process	-	+acc=…	
#vsim_capacity#		Capacity	Statistics	+acc=…	

Objects

Name	Value	Kind	Mode
Pwrite	1'h1	Net	Internal
Pwdata	32'h0…	Net	Internal
Pselx	3'h1	Net	Internal
Penable	1'h0	Net	Internal
Paddr	32'h8…	Net	Internal
Hwrite	1'h1	Regi…	Internal
Hwdata	32'h0…	Pack…	Internal
Htrans	2'h0	Pack…	Internal
Hresp	2'h0	Net	Internal

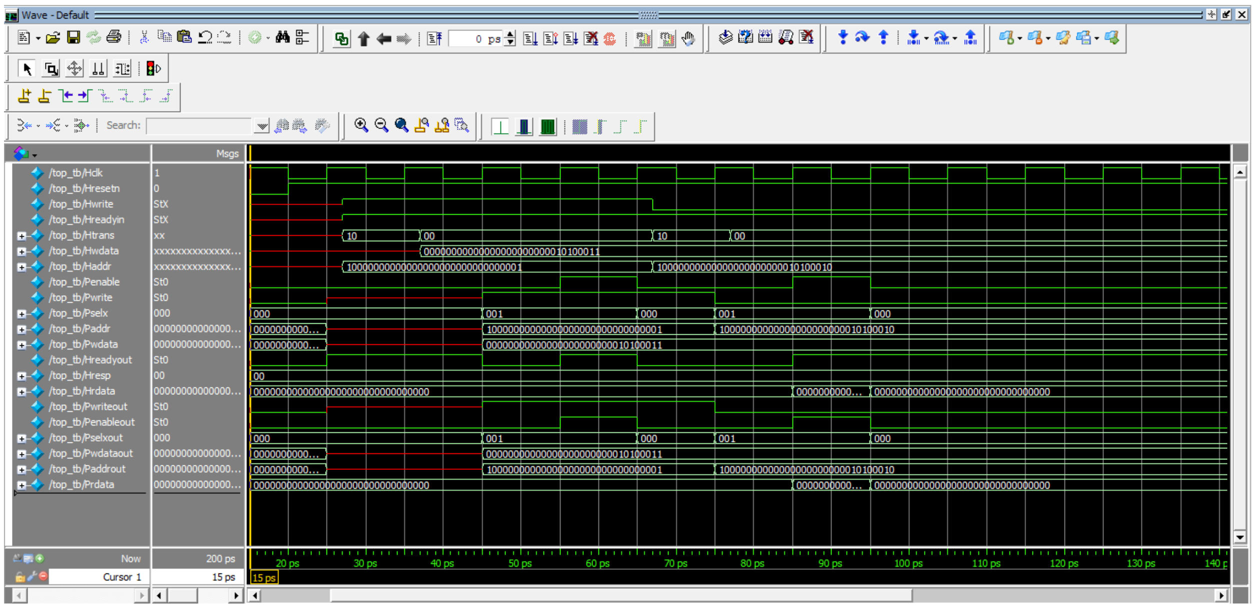
Processes (Active)

Name	Type (filtered)	State	Order	Parent Path	Class Info



	Msgs
/tb/Hclk	1'h1
/tb/Hresetn	1'h1
/tb/Hwrite	1'h1
/tb/Hreadyin	1'h1
/tb/Hwdata	32'h000000a3
/tb/Haddr	32'h80000001
/tb/Htrans	2'h0
/tb/Penable	1'h0
/tb/Pwrite	1'h1
/tb/Hreadyout	1'h0
/tb/Hresp	2'h0
/tb/Pselx	3'h1
/tb/Paddr	32'h80000001
/tb/Pwdata	32'h000000a3
/tb/Hrdata	32'hzzzzzzzz
Now	500000 ps
Cursor 1	109672 ps

The testbench above, and referenced in our appendix is our own basic design. However, there was a waveform generated from the bridge designer's himself, though we don't have a code for it, and this is the waveform below.



```
Wave - Default
                      Msgs
/top_tb/Hclk          1
/top_tb/Hresetn       0
/top_tb/Hwrite        StX
/top_tb/Hreadyin      StX
/top_tb/Htrans        xx
/top_tb/Hwdata        xxxxxxxxxxxxxx...
/top_tb/Haddr         xxxxxxxxxxxxxx...
/top_tb/Penable       St0
/top_tb/Pwrite        St0
/top_tb/Pselx         000
/top_tb/Paddr         00000000000000...
/top_tb/Pwdata        00000000000000...
/top_tb/Hreadyout     St0
/top_tb/Hresp         00
/top_tb/Hrdata        00000000000000...
/top_tb/Pwriteout     St0
/top_tb/Penableout    St0
/top_tb/Pselxout      000
/top_tb/Pwdataout     00000000000000...
/top_tb/Paddrout      00000000000000...
/top_tb/Prdata        00000000000000...
              Now     200 ps
         Cursor 1     15 ps
```

Below is the finite state machine from the ARM AHB System Technical Reference Manual.

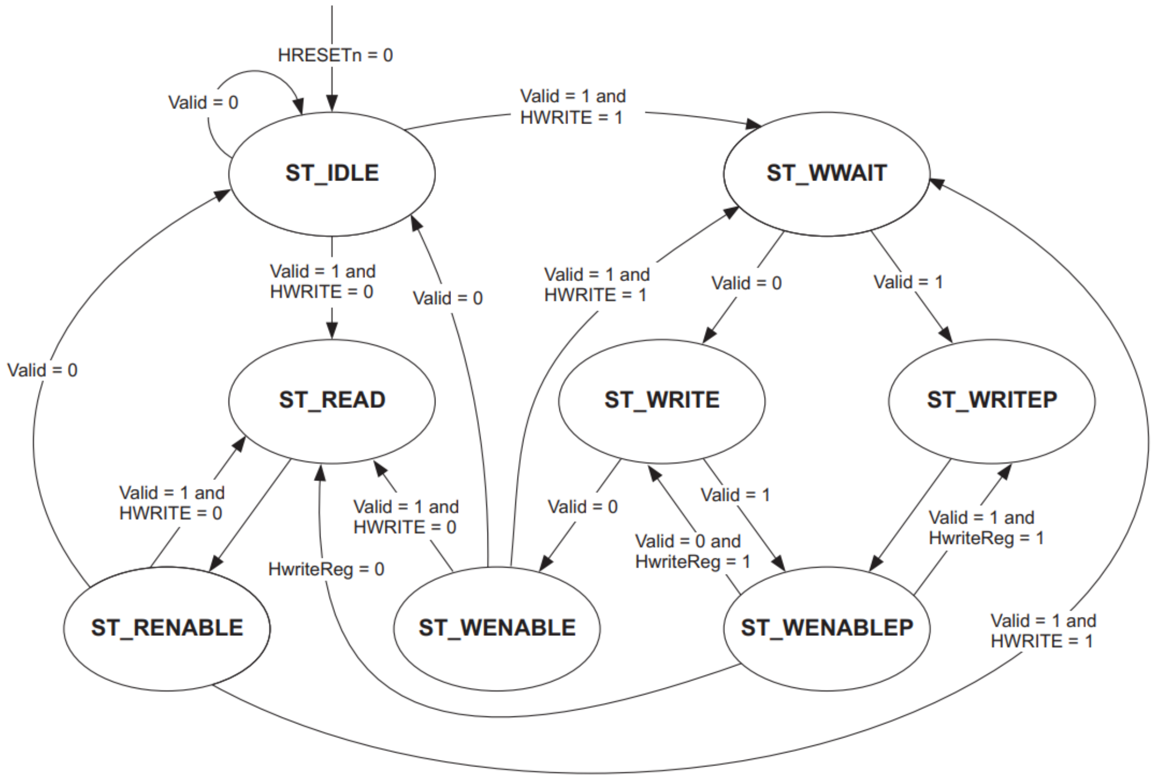


Figure 4-3 State machine for AHB to APB interface

Initial Project Proposal

1. Project Objective:
   - Develop Verification IP (VIP) for the AHB2APB bridge design.
   - Verify the functionality of the AHB2APB bridge design using SystemVerilog and UVM-based testbench development.
   - Implement directed tests, random tests, SystemVerilog assertions, and coverage analysis to thoroughly verify the design.

2. Design Selection:
   - We have selected a customized AHB2APB bridge design based on the AHB2APB standard. The RTL code we are utilizing for our project is attached to this submission (referenced in the appendix) and sourced from: [https://github.com/prajwalgekkouga/AHB-to-APB-Bridge]
   - Simulation Environment: Questa Sim.

3. VIP Development:
   - Master BFM: We plan to develop a Master Bus Functional Model (BFM) to generate AHB transactions and drive them to our customized AHB2APB bridge.
   - Slave BFM: We plan to create a Slave BFM to respond to APB transactions from the AHB2APB bridge and emulate the behavior of APB slaves.
   - Monitor: We plan to create a Monitor module to capture and monitor signals and transactions between the AHB and APB interfaces in our customized design.
   - Checker: We plan to implement a Checker module to verify the correctness of the AHB2APB bridge behavior and detect any anomalies.
   - Scoreboard: We plan to develop a Scoreboard module to compare the expected and observed transactions and flag any discrepancies for analysis.
   - SVA: We plan to write SystemVerilog Assertions (SVA) to capture and verify specific design properties and behaviors, tailored to our customized AHB2APB bridge.
   - Coverpoints and Covergroups: We plan to define coverpoints and covergroups specific to our customized AHB2APB bridge design to achieve functional coverage goals.
   - Testbench: We plan to design a comprehensive testbench with multiple masters and multiple slaves, specifically tailored to our customized AHB2APB bridge design, to verify its functionality.
   - Tests: We plan to develop directed tests and random tests targeting different functionalities and corner cases of our customized AHB2APB bridge design to ensure its robustness.

4. Verification of the Design:
   - Directed Tests: We plan to develop specific test scenarios to exercise various functionalities and corner cases of our customized AHB2APB bridge design, ensuring its compliance with our design requirements.
   - Random Tests: We plan to generate random test stimuli to stress-test our customized AHB2APB bridge design and uncover any potential issues or corner case bugs.
   - SystemVerilog Assertions: We plan to write assertions to specify and verify expected behaviors and properties unique to our customized AHB2APB bridge design.
   - Coverage: We plan to define coverage metrics and track the coverage progress using coverage analysis tools to ensure comprehensive verification of our customized AHB2APB bridge design.

5. Documentation and Submission:
   - Detailed Design Code: We plan to prepare a detailed design code for all the VIP components specific to our customized AHB2APB bridge design.
   - Simulation Waveforms: We plan to generate simulation waveforms to demonstrate the correctness and functionality of our customized AHB2APB bridge design.
   - Specifications and References: We plan to compile a comprehensive list of specifications and references to websites used during the design process of our customized AHB2APB bridge.
   - Submission: We plan to submit the design code, simulation waveforms, and a document containing specifications and references for our customized AHB2APB bridge design.

Design Specifications

Design Overview:

- Our customized AHB2APB bridge design provides a bridge between the Advanced High-performance Bus (AHB) and the Advanced Peripheral Bus (APB) interfaces, tailored to meet our specific requirements.
- The design supports AHB masters and APB slaves, allowing seamless transactions between the two interfaces while ensuring compliance with our design specifications.

Design Input Signals (AHB side):

- HCLK: Clock signal for the AHB interface.
- HRESETn: Active-low reset signal for the AHB interface.
- HSIZE[2:0]: Size of the transfer on the AHB interface.
- HADDR[31:0]: Address for the AHB transfer.
- HTRANS[1:0]: Transfer type on the AHB interface.
- HWRITE: Write enable signal for the AHB interface.
- HWDATA[31:0]: Write data on the AHB interface.
- HREADYin: Ready signal indicating the availability of the AHB interface.

Design Output Signals (AHB side):

- HRDATA[31:0]: Read data from the AHB interface.
- HREADYout: Ready signal indicating the readiness of the AHB interface.
- HRESP[1:0]: Response indicating the status of the AHB transfer.

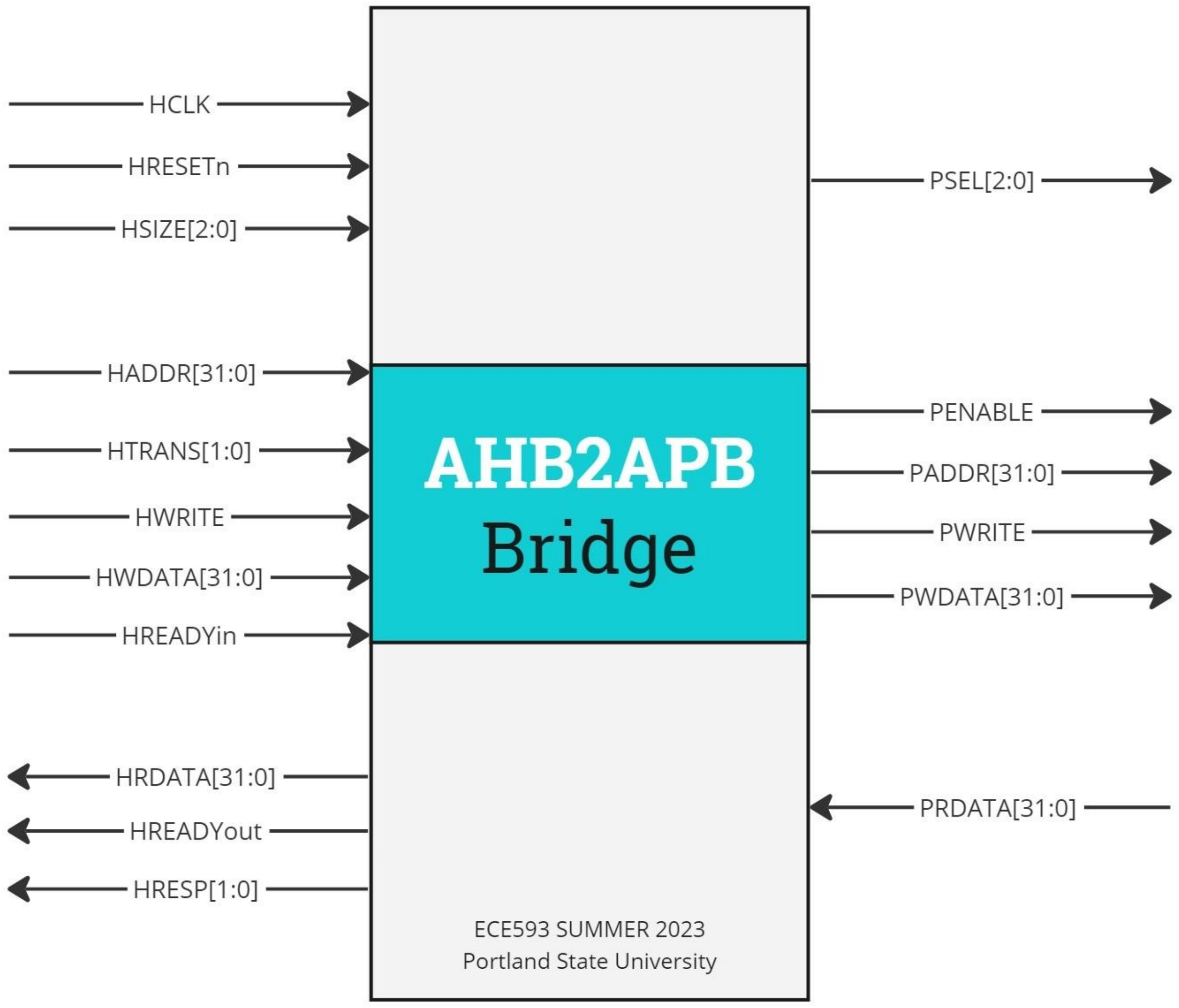
Design Input Signals (APB side):

- PRDATA[31:0]: Read data on the APB interface.

Design Output Signals (APB side):

- PSEL[2:0]: Slave select signal on the APB interface.
- PENABLE: Enable signal for the APB interface.
- PADDR[31:0]: Address for the APB transfer.
- PWRITE: Write enable signal for the APB interface.
- PWDATA[31:0]: Write data on the APB interface.

These input and output signals form the communication interface between the AHB and APB sides of the bridge design. They facilitate the transfer of data and control signals between the two interfaces, ensuring the proper functioning of the AHB2APB bridge.



Appendix 1 (RTL Design Code)

This code is referenced from: https://github.com/prajwalgekkouga/AHB-to-APB-Bridge

```
// Bridge Top

module
Bridge_Top(Hclk,Hresetn,Hwrite,Hreadyin,Hreadyout,Hwdata,Haddr,Htrans,Prdata,Penable
,Pwrite,Pselx,Paddr,Pwdata,Hreadyout,Hresp,Hrdata);

input Hclk,Hresetn,Hwrite,Hreadyin;
input [31:0] Hwdata,Haddr,Prdata;
input[1:0] Htrans;
output Penable,Pwrite,Hreadyout;
output [1:0] Hresp;
output [2:0] Pselx;
output [31:0] Paddr,Pwdata;
output [31:0] Hrdata;

/////////////////////////////////////////////////////////////////INTERMEDIATE
SIGNALS

wire valid;
wire [31:0] Haddr1,Haddr2,Hwdata1,Hwdata2;
wire Hwritereg;
wire [2:0] tempselx;

///////////////////////////////////////////////////////////////// MODULE
INSTANTIATIONS
```

```
AHB_slave_interface AHBSlave
(Hclk,Hresetn,Hwrite,Hreadyin,Htrans,Haddr,Hwdata,Prdata,valid,Haddr1,Haddr2,Hwdata1
,Hwdata2,Hrdata,Hwritereg,tempselx,Hresp);


APB_FSM_Controller APBControl (
Hclk,Hresetn,valid,Haddr1,Haddr2,Hwdata1,Hwdata2,Prdata,Hwrite,Haddr,Hwdata,Hwriter
eg,tempselx,Pwrite,Penable,Pselx,Paddr,Pwdata,Hreadyout);


endmodule


module
APB_Interface(Pwrite,Pselx,Penable,Paddr,Pwdata,Pwriteout,Pselxout,Penableout,Paddrout
,Pwdataout,Prdata);


input Pwrite,Penable;

input [2:0] Pselx;

input [31:0] Pwdata,Paddr;


output Pwriteout,Penableout;

output [2:0] Pselxout;

output [31:0] Pwdataout,Paddrout;

output reg [31:0] Prdata;


assign Penableout=Penable;

assign Pselxout=Pselx;

assign Pwriteout=Pwrite;

assign Paddrout=Paddr;

assign Pwdataout=Pwdata;


always @(*)
```

```
 begin
  if (~Pwrite && Penable)
   Prdata=($random)%256;
  else
   Prdata=0;
 end

endmodule

module APB_FSM_Controller(
Hclk,Hresetn,valid,Haddr1,Haddr2,Hwdata1,Hwdata2,Prdata,Hwrite,Haddr,Hwdata,Hwriter
eg,tempselx,
                                          Pwrite,Penable,Pselx,Paddr,Pwdata,Hreadyout);

input Hclk,Hresetn,valid,Hwrite,Hwritereg;
input [31:0] Hwdata,Haddr,Haddr1,Haddr2,Hwdata1,Hwdata2,Prdata;
input [2:0] tempselx;
output reg Pwrite,Penable;
output reg Hreadyout;
output reg [2:0] Pselx;
output reg [31:0] Paddr,Pwdata;

//////////////////////////////////////////////////////PARAMETERS

parameter ST_IDLE=3'b000;
parameter ST_WWAIT=3'b001;
parameter ST_READ= 3'b010;
parameter ST_WRITE=3'b011;
parameter ST_WRITEP=3'b100;
parameter ST_RENABLE=3'b101;
parameter ST_WENABLE=3'b110;
parameter ST_WENABLEP=3'b111;


////////////////////////////////////////////////////// PRESENT STATE LOGIC

reg [2:0] PRESENT_STATE,NEXT_STATE;

always @(posedge Hclk)
 begin:PRESENT_STATE_LOGIC
  if (~Hresetn)
    PRESENT_STATE<=ST_IDLE;
  else
    PRESENT_STATE<=NEXT_STATE;
 end


///////////////////////////////////////////////////////// NEXT STATE LOGIC

always @(PRESENT_STATE,valid,Hwrite,Hwritereg)
 begin:NEXT_STATE_LOGIC
  case (PRESENT_STATE)

            ST_IDLE:begin
                     if (~valid)
                      NEXT_STATE=ST_IDLE;
                     else if (valid && Hwrite)
```

```
               NEXT_STATE=ST_WWAIT;
              else
               NEXT_STATE=ST_READ;
             end

ST_WWAIT:begin
              if (~valid)
               NEXT_STATE=ST_WRITE;
              else
               NEXT_STATE=ST_WRITEP;
             end

ST_READ: begin
                NEXT_STATE=ST_RENABLE;
              end

ST_WRITE:begin
               if (~valid)
                NEXT_STATE=ST_WENABLE;
               else
                NEXT_STATE=ST_WENABLEP;
              end

ST_WRITEP:begin
                NEXT_STATE=ST_WENABLEP;
               end

ST_RENABLE:begin
```

```
                 if (~valid)
                  NEXT_STATE=ST_IDLE;
                 else if (valid && Hwrite)
                  NEXT_STATE=ST_WWAIT;
                 else
                  NEXT_STATE=ST_READ;
                end

ST_WENABLE:begin
                 if (~valid)
                  NEXT_STATE=ST_IDLE;
                 else if (valid && Hwrite)
                  NEXT_STATE=ST_WWAIT;
                 else
                  NEXT_STATE=ST_READ;
                end

ST_WENABLEP:begin
                  if (~valid && Hwritereg)
                   NEXT_STATE=ST_WRITE;
                  else if (valid && Hwritereg)
                   NEXT_STATE=ST_WRITEP;
                  else
                   NEXT_STATE=ST_READ;
                 end

default: begin
                NEXT_STATE=ST_IDLE;
```

```
                        end
  endcase
 end



///////////////////////////////////////////////////////OUTPUT
LOGIC:COMBINATIONAL


reg Penable_temp,Hreadyout_temp,Pwrite_temp;

reg [2:0] Pselx_temp;

reg [31:0] Paddr_temp, Pwdata_temp;


always @(*)

 begin:OUTPUT_COMBINATIONAL_LOGIC

   case(PRESENT_STATE)


        ST_IDLE: begin

                          if (valid && ~Hwrite)

                           begin:IDLE_TO_READ

                               Paddr_temp=Haddr;

                                   Pwrite_temp=Hwrite;

                                   Pselx_temp=tempselx;

                                   Penable_temp=0;

                                   Hreadyout_temp=0;

                           end


                          else if (valid && Hwrite)

                           begin:IDLE_TO_WWAIT
```

```
                    Pselx_temp=0;
                        Penable_temp=0;
                        Hreadyout_temp=1;
                end

                else
        begin:IDLE_TO_IDLE
                    Pselx_temp=0;
                        Penable_temp=0;
                        Hreadyout_temp=1;
                end
            end

ST_WWAIT:begin
        if (~valid)
                begin:WAIT_TO_WRITE
                 Paddr_temp=Haddr1;
                        Pwrite_temp=1;
                        Pselx_temp=tempselx;
                        Penable_temp=0;
                        Pwdata_temp=Hwdata;
                        Hreadyout_temp=0;
                end

                else
                begin:WAIT_TO_WRITEP
                 Paddr_temp=Haddr1;
                        Pwrite_temp=1;
                    Pselx_temp=tempselx;
                    Pwdata_temp=Hwdata;
                    Penable_temp=0;
                    Hreadyout_temp=0;
                end

            end

ST_READ: begin:READ_TO_RENABLE
                Penable_temp=1;
                Hreadyout_temp=1;
            end

ST_WRITE:begin
 if (~valid)
                begin:WRITE_TO_WENABLE
                    Penable_temp=1;
                    Hreadyout_temp=1;
                end

               else
                begin:WRITE_TO_WENABLEP ///DOUBT
                    Penable_temp=1;
                    Hreadyout_temp=1;
                end
            end

ST_WRITEP:begin:WRITEP_TO_WENABLEP
```

```
 Penable_temp=1;
                         Hreadyout_temp=1;
                end

ST_RENABLE:begin
          if (valid && ~Hwrite)
                                   begin:RENABLE_TO_READ
                                              Paddr_temp=Haddr;
                                              Pwrite_temp=Hwrite;
                                              Pselx_temp=tempselx;
                                              Penable_temp=0;
                                              Hreadyout_temp=0;
                                   end

                        else if (valid && Hwrite)
                          begin:RENABLE_TO_WWAIT
                           Pselx_temp=0;
                                   Penable_temp=0;
                                   Hreadyout_temp=1;
                          end

                        else
  begin:RENABLE_TO_IDLE
                           Pselx_temp=0;
                                   Penable_temp=0;
                                   Hreadyout_temp=1;
                          end
```

```
                    end

ST_WENABLEP:begin
   if (~valid && Hwritereg)
                              begin:WENABLEP_TO_WRITEP
                               Paddr_temp=Haddr2;
                                        Pwrite_temp=Hwrite;
                                        Pselx_temp=tempselx;
                                        Penable_temp=0;
                                        Pwdata_temp=Hwdata;
                                        Hreadyout_temp=0;
                                       end



                            else
                             begin:WENABLEP_TO_WRITE_OR_READ /////DOUBT
                              Paddr_temp=Haddr2;
                                       Pwrite_temp=Hwrite;
                                       Pselx_temp=tempselx;
                                       Pwdata_temp=Hwdata;
                                       Penable_temp=0;
                                       Hreadyout_temp=0;
                             end
                    end

ST_WENABLE :begin
            if (~valid && Hwritereg)
                              begin:WENABLE_TO_IDLE
```

```
                                     Pselx_temp=0;
                                     Penable_temp=0;
                                     Hreadyout_temp=0;
                                    end


                              else
                               begin:WENABLE_TO_WAIT_OR_READ /////DOUBT
                                    Pselx_temp=0;
                                    Penable_temp=0;
                                    Hreadyout_temp=0;
                               end

                          end

 endcase
end


////////////////////////////////////////////////////////////OUTPUT
LOGIC:SEQUENTIAL

always @(posedge Hclk)
 begin

  if (~Hresetn)
   begin
    Paddr<=0;
```

```
            Pwrite<=0;
            Pselx<=0;
            Pwdata<=0;
            Penable<=0;
            Hreadyout<=0;
  end

 else
  begin
       Paddr<=Paddr_temp;
            Pwrite<=Pwrite_temp;
            Pselx<=Pselx_temp;
            Pwdata<=Pwdata_temp;
            Penable<=Penable_temp;
            Hreadyout<=Hreadyout_temp;
  end
 end

endmodule

module AHB_slave_interface(Hclk,Hresetn,Hwrite,Hreadyin,Htrans,Haddr,Hwdata,

Prdata,valid,Haddr1,Haddr2,Hwdata1,Hwdata2,Hrdata,Hwritereg,tempselx,Hresp);
input Hclk,Hresetn;
input Hwrite,Hreadyin;
input [1:0] Htrans;
input [31:0] Haddr,Hwdata,Prdata;
output reg valid;
output reg [31:0] Haddr1,Haddr2,Hwdata1,Hwdata2;
output [31:0] Hrdata;
output reg Hwritereg;
output reg [2:0] tempselx;
output  [1:0] Hresp;



/// Implementing Pipeline Logic for Address,Data and Control Signal

        always @(posedge Hclk)
                begin

                        if (~Hresetn)
                                begin
                                        Haddr1<=0;
                                        Haddr2<=0;
                                end
                        else
                                begin
                                        Haddr1<=Haddr;
                                        Haddr2<=Haddr1;
                                end

                end

        always @(posedge Hclk)
                begin
                        if (~Hresetn)
                                begin
                                        Hwdata1<=0;
                                        Hwdata2<=0;
                                end
                        else
                                begin
                                        Hwdata1<=Hwdata;
                                        Hwdata2<=Hwdata1;
                                end

                end

        always @(posedge Hclk)
                begin
                        if (~Hresetn)
                                Hwritereg<=0;
                        else
                                Hwritereg<=Hwrite;
                end



/// Implementing Valid Logic Generation

        always @(Hreadyin,Haddr,Htrans,Hresetn)
                begin
                        valid=0;
```

```
                        if (Hresetn && Hreadyin && (Haddr>=32'h8000_0000 &&
Haddr<32'h8C00_0000) && (Htrans==2'b10 || Htrans==2'b11) )

                                valid=1;


                end


/// Implementing Tempselx Logic


        always @(Haddr,Hresetn)

                begin

                        tempselx=3'b000;

                        if (Hresetn && Haddr>=32'h8000_0000 && Haddr<32'h8400_0000)

                                tempselx=3'b001;

                        else if (Hresetn && Haddr>=32'h8400_0000 &&
Haddr<32'h8800_0000)

                                tempselx=3'b010;

                        else if (Hresetn && Haddr>=32'h8800_0000 &&
Haddr<32'h8C00_0000)

                                tempselx=3'b100;


                end




assign Hrdata = Prdata;

assign Hresp=2'b00;


endmodule
```

```
module
AHB_Master(Hclk,Hresetn,Hresp,Hrdata,Hwrite,Hreadyin,Hreadyout,Htrans,Hwdata,Haddr
);


input Hclk,Hresetn,Hreadyout;

input [1:0]Hresp;

input [31:0] Hrdata;

output reg Hwrite,Hreadyin;

output reg [1:0] Htrans;

output reg [31:0] Hwdata,Haddr;


reg [2:0] Hburst;

reg [2:0] Hsize;




task single_write();

 begin

  @(posedge Hclk)

  #2;

   begin

    Hwrite=1;

    Htrans=2'b10;

    Hsize=3'b000;

    Hburst=3'b000;

    Hreadyin=1;

    Haddr=32'h8000_0001;

   end
  @(posedge Hclk)
  #2;
   begin
    Htrans=2'b00;
    Hwdata=8'hA3;
   end
 end
endtask



task single_read();
 begin
  @(posedge Hclk)
  #2;
   begin
    Hwrite=0;
    Htrans=2'b10;
    Hsize=3'b000;
    Hburst=3'b000;
    Hreadyin=1;
    Haddr=32'h8000_00A2;
   end

  @(posedge Hclk)
  #2;
   begin
    Htrans=2'b00;
   end
 end
endtask


endmodule
```

Appendix 2 (Our basic testbench)

We created this testbench.

```
`timescale 1ns / 1ps


module tb;
  // Declare signals
  reg Hclk, Hresetn, Hwrite, Hreadyin;
  reg [31:0] Hwdata, Haddr;
  reg [1:0] Htrans;
  wire Penable, Pwrite, Hreadyout;
  wire [1:0] Hresp;
  wire [2:0] Pselx;
  wire [31:0] Paddr, Pwdata, Hrdata;

  // Instantiate the Bridge_Top module
  Bridge_Top bridge (
    .Hclk(Hclk), .Hresetn(Hresetn), .Hwrite(Hwrite), .Hreadyin(Hreadyin),
    .Hwdata(Hwdata), .Haddr(Haddr), .Htrans(Htrans),
    .Penable(Penable), .Pwrite(Pwrite), .Hreadyout(Hreadyout),
    .Hresp(Hresp), .Pselx(Pselx), .Paddr(Paddr), .Pwdata(Pwdata), .Hrdata(Hrdata)
  );

  // Clock generator
  always begin
    #5 Hclk = ~Hclk;
  end

  // Testbench stimulus
  initial begin
    // Initialize signals
    Hclk = 0; Hresetn = 0; Hwrite = 0; Hreadyin = 0;
    Hwdata = 0; Haddr = 0; Htrans = 0;

    // Apply reset
    #10 Hresetn = 1;

    // Infinite loop for read and write operations
    forever begin
      // Write operation
      #20 Hwrite = 1; Hreadyin = 1; Htrans = 2'b10; Haddr = 32'h8000_0001; Hwdata =
8'hA3;
      #10 Htrans = 2'b00;

      // Read operation
      #20 Hwrite = 0; Hreadyin = 1; Htrans = 2'b10; Haddr = 32'h8000_00A2;
      #10 Htrans = 2'b00;
    end
  end

endmodule
```