Table of Contents

[Introduction 2](#_Toc139451590)

[Initial Project Proposal 5](#_Toc139451591)

[Design Specifications 7](#_Toc139451592)

[Appendix 1 (RTL Design Code) 9](#_Toc139451593)

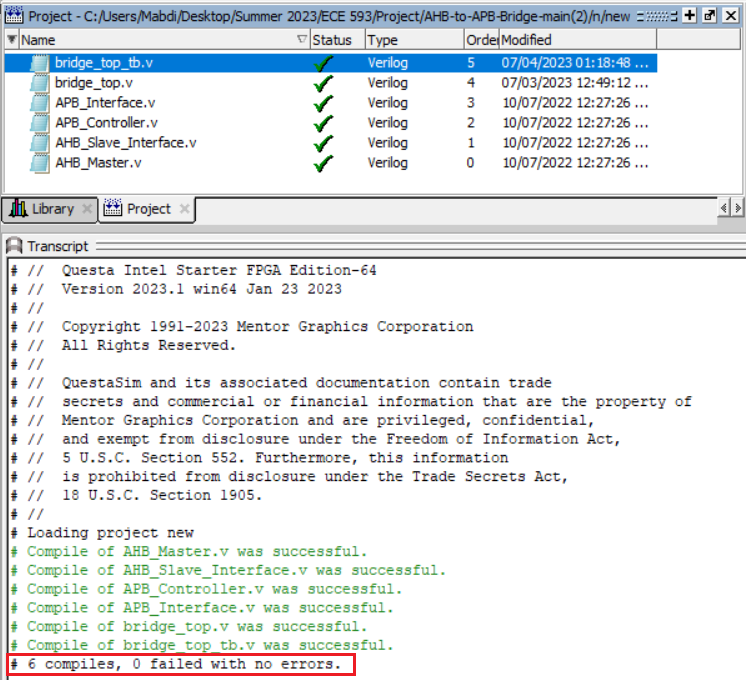
[Appendix 2 (Our basic testbench) 28](#_Toc139451594)

## Introduction

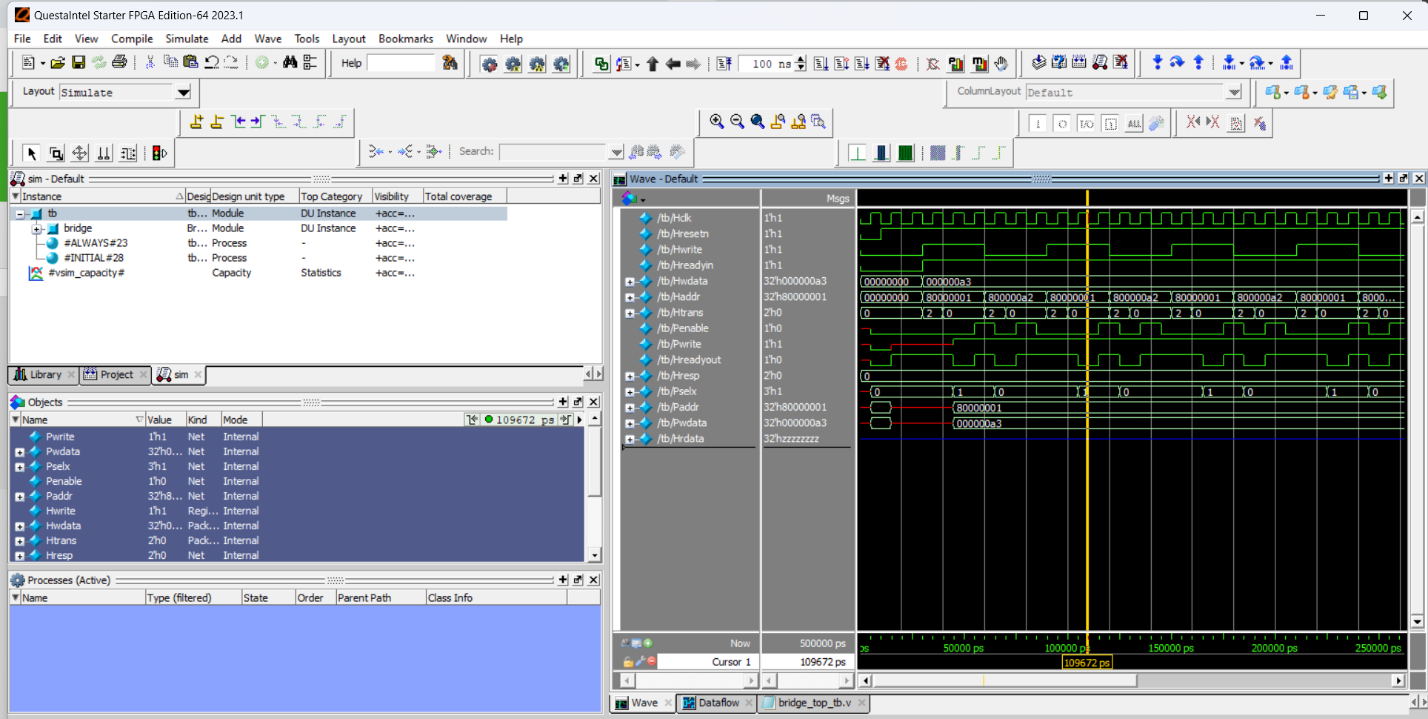
Our project focuses on the verification and validation of the AHB2APB bridge design. In order to achieve this objective, we plan to adopt System Verilog and UVM-based test bench development methodologies to create Verification IP (VIP) components for the design. The purpose of this initial phase is to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

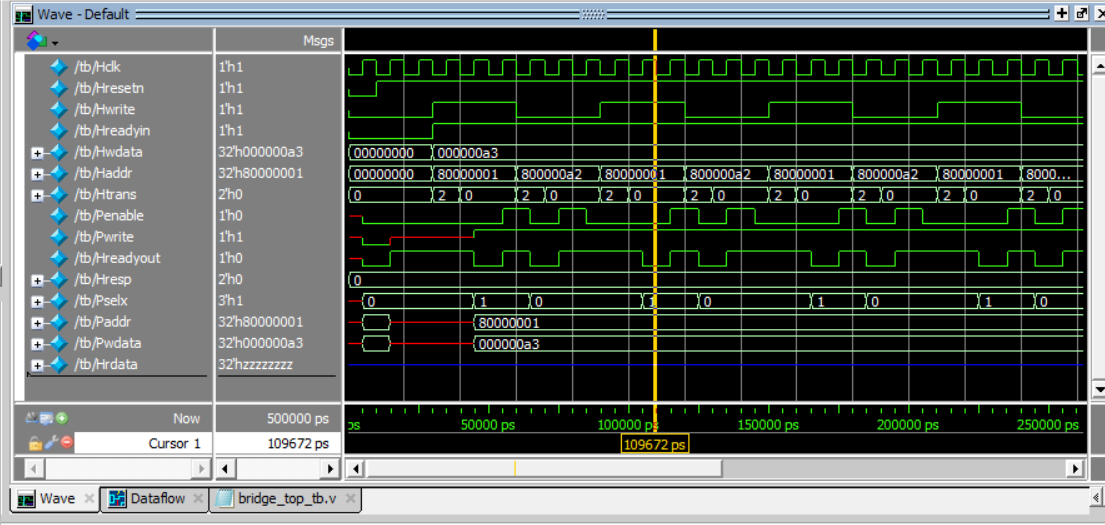
To initiate our project, we selected an RTL code for the AHB2APB bridge design, which is included as an attachment to this submission and referenced in the appendix. The chosen RTL code can be found at: <https://github.com/prajwalgekkouga/AHB-to-APB-Bridge>

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.

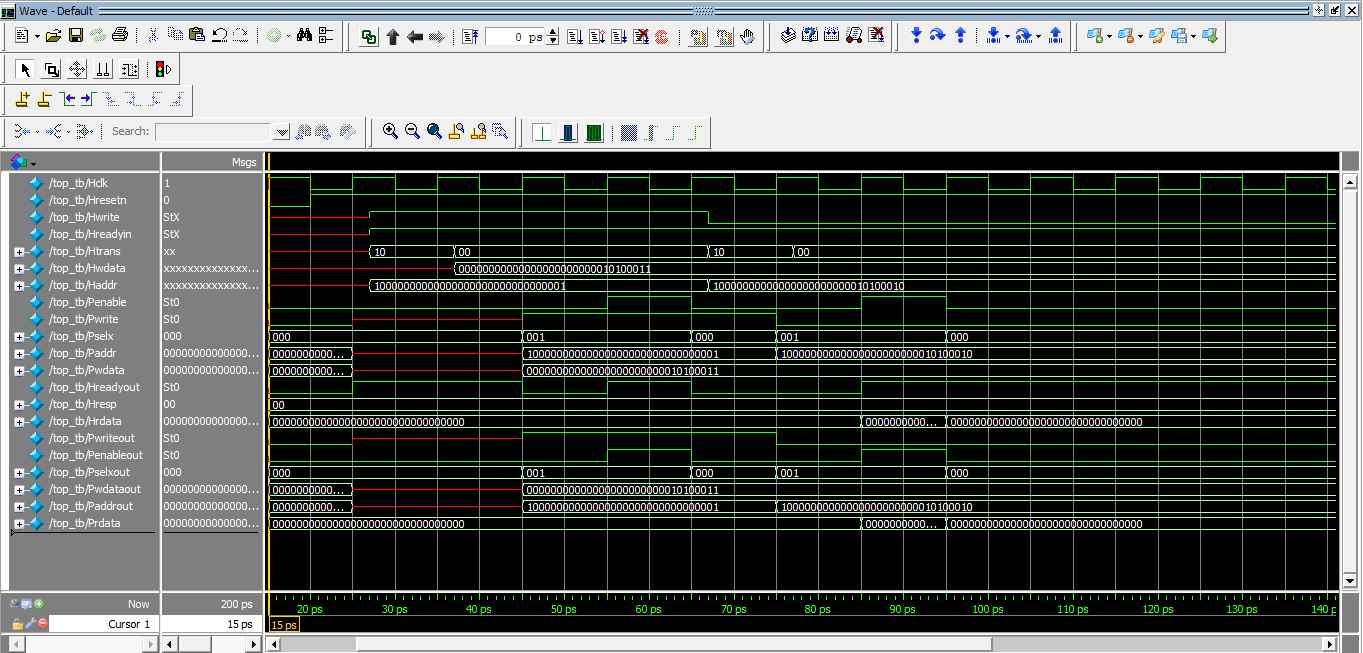


In addition, we designed our own basic testbench simulation to observe the waveform generated by the design, and this screenshot is attached below.

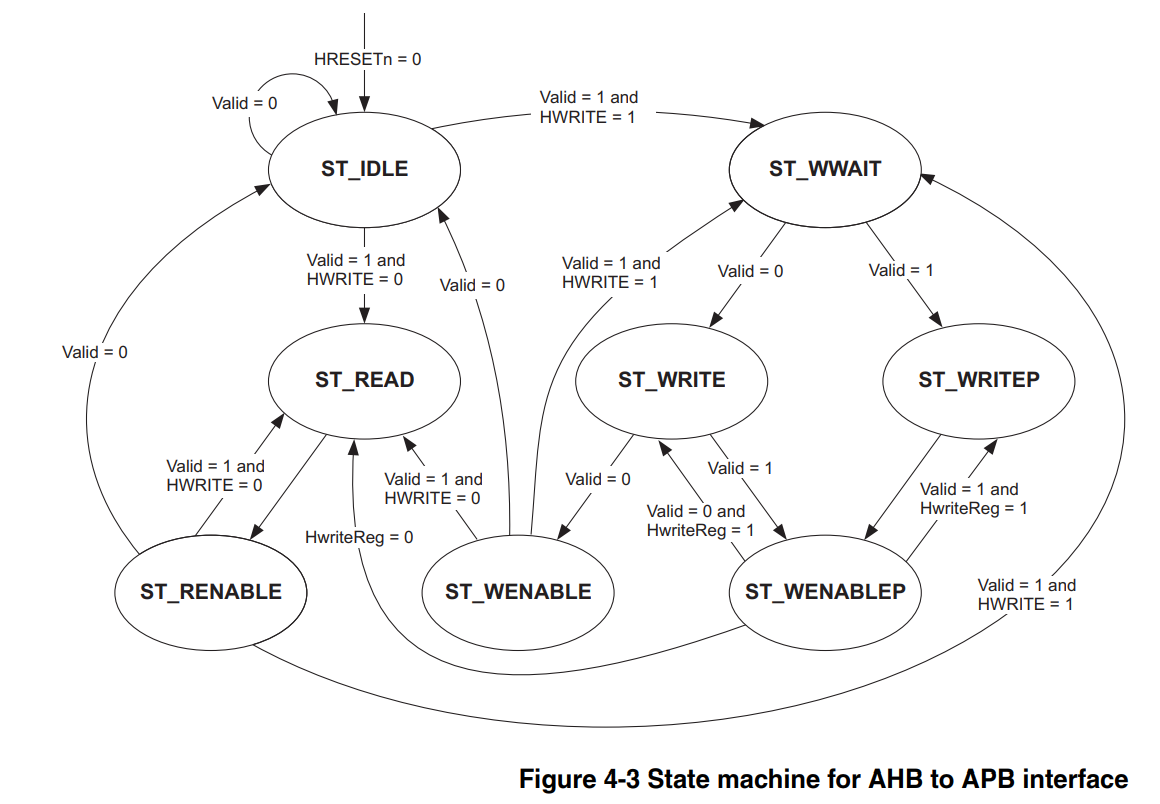




The testbench above, and referenced in our appendix is our own basic design. However, there was a waveform generated from the bridge designer’s himself, though we don’t have a code for it, and this is the waveform below.



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

****

## 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.

1. 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.

1. 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.

1. 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.

1. 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.

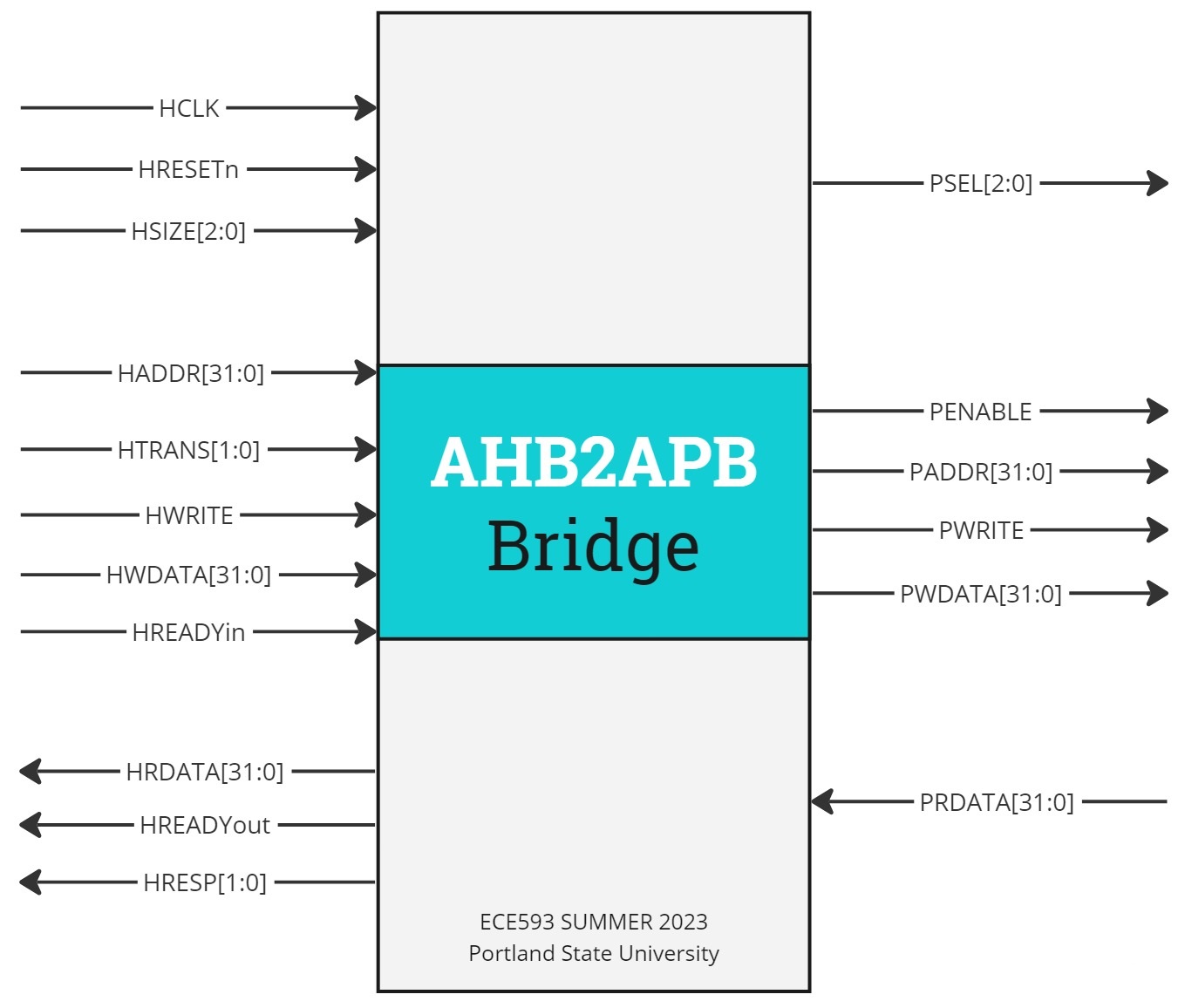
Design Input Signals (APB side):

* 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,Hwritereg,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,Hwritereg,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