AHB2APB BRIDGE RTL DESIGN USING VERILOG HDL

Mini Project Report

submitted in partial fulfillment for the award of the degree of

Bachelor of Technology

in

Electronics and Communication Engineering

Under the Esteemed Guidance of

Mrs.E. SUNEETHA

Guide Designation

by

KAMUJU SAI RAKESH 187Z1A0441

Department of Electronics and Communication Engineering

Nalla Narasimha Reddy Education Society's Group of Institutions

(Approved by AICTE, Affiliated to JNTUH & Accredited by NBA and NAAC)

Chowdariguda, Medchal-Malkajgiri (Dist.), Hyderabad –500088, Telangana

2021-2022

Nalla Narasimha Reddy Education Society's Group of Institutions

(Approved by AICTE, Affiliated to JNTUH & Accredited by NBA and NAAC) ***Chowdariguda, Medchal-Malkajgiri (Dist.), Hyderabad –500088, Telangana***

School of Engineering

Department of Electronics and Communication Engineering

CERTIFICATE

This is to certify that the project work entitled **"AHB2APB BRIDGE RTL DESIGN USING VERILOG HDL"** submitted by **KAMUJU SAI RAKESH-187Z1A0441** is in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Electronics and Communication** Engineering from **Jawaharlal Nehru Technological University**, **Hyderabad** during the academic year **2021-22**.

This mini project is a record of bonafide work carried out by them under the supervised guidance and has not been submitted to any other university or institution for the award of any degree.

Internal Guide | Head of the Department

Mrs.E.SUNEETHA
Associate Professor

Mr. P.S. SREENIVASA REDDY
Associate Professor

External Examiner

ACKNOWLEDGEMENT

I would like to thank**, Mrs.E.SUNEETHA,** Associate Professor, internal guide for his/her wonderful support and unforgettable help towards the project. Without his encouragement and constant guidance, we/I could not have finished this project and dissertation.

I would like to thank **Dr. P.MICHAEL PREETAM RAJ,** Associate Professor & Project Coordinator for the constant support and guidance throughout the course of the study.

I would like to thank **Dr.T.RAJASHEKAR,** Associate Professor & Project Coordinator for the constant support and guidance throughout the course of the study

I would like to thank **Mr. P.S. SREENIVASA REDDY,** Associate Professor & Head of the Department of Electronics and Communication Engineering for inspiring us/me to take up a project on this subject and successfully guiding us/me towards its completion.

I would like to thank **Dr. G.JANARDHANA RAJU**, Dean, School of Engineering for providing us/me by means of attaining our cherished goals.

We/I would like to thank Director **Dr. C.V.KRISHNA REDDY** whose sayings and teachings have been pivotal for us/me in all aspects of our/my academics. He has stood beside us/me in all the difficult times, and his love has helped us/me to overcome all the failures and success.

I express our/my gratitude to the institution **NALLA NARASIMHA REDDY EDUCATION SOCIETY'S GROUP OF INSTITUTIONS.**

Last but not the least, we/I would like to thank all the faculty members & lab instructors for the timely inputs and help for the successful completion of the project.

KAMUJJU SAI RAKESH - 187Z1A0441

DECLARATION

I hereby declare that the project dissertation entitled **"AHB2APB BRIDGE RTL DESIGN USING VERILOG HDL"** submitted by us/me in the partial fulfillment for the award of the degree of Bachelor of Technology in Electronics & Communication Engineering under Jawaharlal Nehru Technological University (JNTU), Hyderabad is our/my original work carried out during the academic year **2020-21** and has not been submitted earlier to any other university/institution for the fulfillment of the requirement for any degree or course of study.

We/I also declare that no chapter of this manuscript is whole or in part incorporated in this report has been copied or fetched from any earlier work done by others. However, the extract of literature which has been used for the study & reporting has been duly acknowledged and indexed by providing the details in the context of references.

KAMUJJU SAI RAKESH - 187Z1A0441

Chapter 1

Introduction to AMBA Architecture

1.1 Overview of the AMBA Specification

The Advanced Microcontroller Bus Architecture (AMBA) specification defines an on-chip communications standard for designing high-performance embedded microcontrollers.

Three distinct buses are defined within the AMBA Architecture

- Advanced High-Performance Bus (AHB)
- Advanced System Bus (ASB)
- Advanced Peripheral Bus (APB)

1.2 Advanced High-Performance Bus(AHB)

The AMBA AHB is for high-performance, high clock frequency system modules. The AHB acts as the high-performance system backbone bus.AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral microcell functions is also specified to ensure case of use in an efficient design flow using synthesis and automated test techniques.

1.3 Advanced Peripheral Bus (APB)

The AMBA APB is for low-power peripherals. AMBA APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions can be used in conjunction with either version of the system bus.

1.4 Objectives of the AMBA specification

The AMBA specification has been derived to satisfy four key requirements:

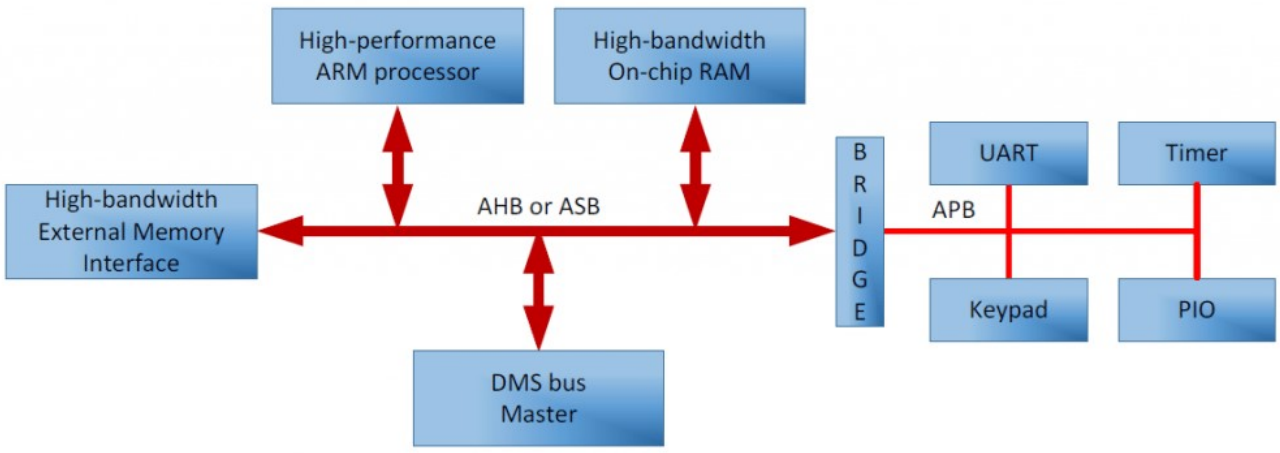
- To facilitate the right-first-time development of embedded microcontroller products with one or more CPU or signal processors
- To be technology-independent and ensure that highly reusable peripheral and system macrocells can be migrated across a diverse range of IC processes and be appropriate for full-custom, standard cell and gate array technologies

-Cont.

- To encourage modular system design to assist in processor independence.
- To minimize the silicon infrastructure required to support efficient communication for both operation and manufacturing test.

1.5 Typical AMBA Architecture

An AMBA architecture consists of a high-performance system backbone bus (AMBA AHB or AMBA ASB), able to sustain the external memory bandwidth, on which the CPU, on-chip memory and other Direct Memory Access (DMA) devices reside. This bus provides a high-bandwidth interface between the elements that are involved in the majority of transfers. Also located on the high-performance bus is bridge to the lower bandwidth Abwehr most of the peripheral devices in the system are located in fig.1.0



1.6 Operation

The master starts a transfer by driving the address and control signals. These signals provide information about the address, direction, width of the transfer, and indicate if the transfer forms part of a burst. Transfers can be:

- Single
- Incrementing bursts that do not wrap at address boundaries
- Wrapping bursts that wrap at particular address boundaries

The write data bus moves data from the master to a slave, and the read data bus moves data from a slave to the master.

-Cont.

Every transfer consists of:

Address phase one address and control cycle.

Data phase one or more cycles for the data.

A slave cannot request that the address phase is extended and therefore all slaves must be capable of sampling the address during this time. However, a slave can request that the master extends the data phase by using HREADY. This signal, when LOW, causes wait states to be inserted into the transfer and enables the slave to have extra time to provide or sample data. The slave uses HRESP to indicate the success or failure of a transfer.

Chapter 2
AMBA SIGNALS

2.1 AMBA AHB Signals

AHB is a new generation of AMBA bus which is developed by ARM intended to address the requirements of high-performance synthesizable designs. It is a high-performance system bus that supports multiple bus masters and provides high-bandwidth operation

AMBA AHB implements the features required for high-performance, high clock frequency systems including:

- Burst transfers
- Split transactions
- Single-cycle bus master handover
- Single-clock edge operation
- Non-tristate implementation
- Wider data bus configurations (64/128 bits).

Bridging between this higher level of bus and the current ASB/APB can be done efficiently to ensure that any existing designs can be easily integrated AMBA AHB design may contain one or more bus masters, typically a system would common AHB slaves. Any other peripherals in the system could also be included as an AHB slave. However, Low-bandwidth peripherals typically reside on the APB.

A typical AMBA AHB system design contains

- AHB master
- AHB slave
- AHB arbiter
- AHB decoder

AHB master	A bus master is able to initiate read and write operations by providing an address and control information. Only one bus master is allowed to actively use the bus at any one time.
AHB slave	A bus slave responds to a read or write operation within a given address-space range. The bus slave signals back to the active master the success, failure or waiting of the data transfer.
AHB arbiter	the bus arbiter ensures that only one bus master at a time is allowed to intitate data transfers. Even though the arbitration protocol is fixed, any arbitration such as highest priority or fair access can be implemented depending on the application requirements.
AHB decoder	The AHB decoder is used to decode the address of each transfer and provide a select signal for the slave that is involved in the transfer.

AMBA AHB Signals:

Name	Destination	Description
HTRANS[1:0]	Slave	Indicates the transfer type of the current transfer. • IDLE • BUSY • NONSEQUENTIAL • SEQUENTIAL
HWDATA[31:0]	Slave	The write data bus transfers data from the master to the slaves during write operation. A minimum data bus width of 32bits is recommended. However, this can be extended to enable higher bandwidth operation.
HRDATA[31:0]	Slave	The read data bus is used to transfer data from bus master during read operations. A minimum data bus width of 32bits is recommended. This can extend to enable higher bandwidth operation
HWRITE	Slave	Indicates the transfer direction. When HIGH this Is signal indicates a write transfer and when LOW a read transfer. It has the same timing as the address signals, however.it must remain constant throughout a burst transfer.
HADDR[31:0]	Slave, Decoder	The 32-bit system address bus.
HBURST[2:0]	Slave	The burst type indicates if the transfer is a single transfers or forms part of a burst. Fixed length bursts of 4,8 and 16 beats are supported. The burst can be incrementing or wrapping. Incrementing bursts of undefined length are also supported.

-Cont.

Name	Destination	Description
HSIZE[2:0]	Slave	Indicates the size of the transfer, that is typically Byte, halfword or word. The protocol allows for larger transfer sizes up to a maximum of 1024bits.
HREADY	Slave	When HIGH the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. Slaves on the bus require HREADY as both an input and an output signal.
HSELx	Decoder	Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.

2.2 AMBA APB

The APB is part of the AMBA hierarchy of buses and is optimized for minimal power consumption and reduced interface complexity. The AMBA APB appears as a local secondary bus that is encapsulated as a single AHB or ASB slave device provides a low-power extension to the system bus which builds on AHB or ASB signals directly.

The APB bridge appears as a slave module which handles the bus handshake and control signal retiming on behalf of the local peripheral bus. By defining the APB interface from the starting point of the system bus, the benefits of the system diagnostics and test methodology can be exploited. The AMBA APB should be used to interface to any peripherals which are low bandwidth and do not require the high performance of a pipelined bus interface.

The latest revision of the APB is specified so that all signal transitions are only related to the rising edge of the clock. This improvement ensures the APB peripherals can be integrated easily into any Design flow.

APB Bus following advantages.

- High-frequency operation easier to achieve.
- Performance is independent of the mark-space ratio of the clock
- Static timing analysis is simplified by the use of a single clock edge
- Easy integration with cycle-based simulators.

A simple APB interface is recommended for simple register-mapped slave devices and for very low power interfaces where clocks cannot be globally routed.

AMBA APB Signals

Name	Description
PCLK	The rising edge of PCLK is used to time all transfers on the apb
PRESETn	Thee APB bus reset signal is active LOW and this signal will normally be connected directly to the system bus reset signal.
PADDR[31:0]	This is the APB address bus, which may be up to 32-bits wide and is driven by the peripheral bus bridge unit.
PSELx	A signal from the secondary decoder ,within the peripheral bus bridge unit, to each peripheral bus slave x. This signal indicated that the slave device is selected and a data transfer is required.
PENABLE	This strobe signal is used to time all accesses on the peripheral bus. The enable signal is used to indicate the second cycle of an APB transfer. The rising edge of PENALBE occurs in the middle of the APB transfer.
PWRITE	When HIGH this signal indicates an APB write access and when LOW a read access.
PRDATA	The read data bus is driven by the selected slave during read cycles(when PWRITE is LOW).
PWDATA	The write data bus is driven by the peripheral bus bridge unit during write cycles(when PWRITE is HIGH).The write data bus can be up to 32-bits wide.

CHAPTER 3

Project Overview

3.1 Abstract

The AMBA AHB is for high-performance, high clock frequency system modules. The AHB acts as the high-performance backbone system bus. AHB supports the efficient connection of processors. The AMBA APB is optimized for low power consumption and interface reduced complexity to support peripheral functions. In this project functions of the AHB2APB Bridge protocol by writing the code in VERILOG and simulating it in XILINX ISE. In this project, we verify the all functions of Bridge protocol by writing verification code in UVM with different test cases. The code coverage and functional coverage and functional verification of the Bridge RTL design is 100% covered by using Intel Quartus Prime Tool.

3.2 Background and Motivation

As part of Maven Silicon internship, I was introduced with AHB2APB bridge module.

I started on the module and learned how to mitigate problems, how to subdivide tasks and create universal blocks to work in sync.in order to learn these I have familiarized with all the required skills in the previous assessments of internship.

3.3 Methodology

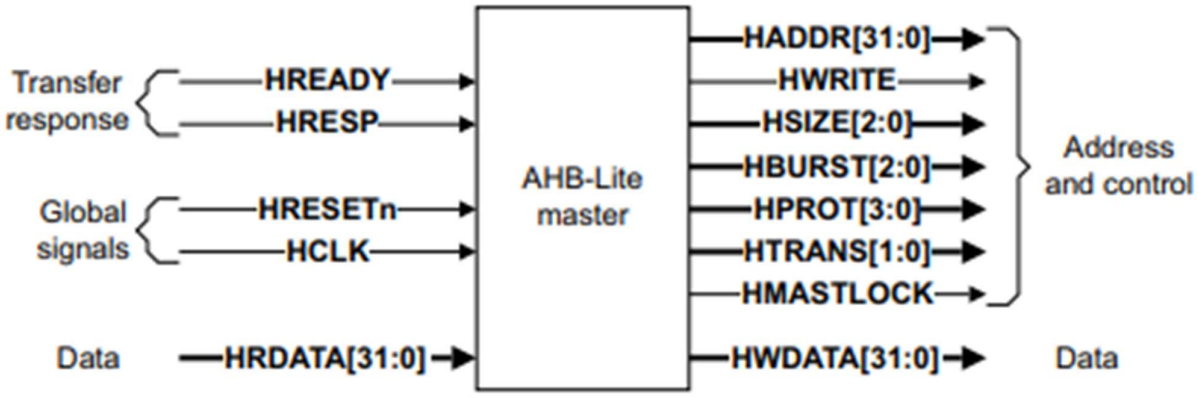
The bridge in our internship creates interface which facilitates single write and single read data transfers. The global signals utilized in this project are HCLK and HRESETn. The project is subdivided into 4 modules each module serves unique role. They are

- AHB master
- AHB slave
- APB FSM controller
- APB Interface

AHB Master

AHB master drives and loads all the necessary signal required for testing and verifying outputs.it has three 1-bit, one 2-bit and one 32-bit input lines namely HCLK, HRESEn, HREADYout, HRESP and HRDATA respectively.

It handles and tests all data transfer types like single write, single read, burst transfer.



AHB Slave

An AHB-Lite slave responds to transfers initiated by masters in the system. The slave uses the HSELx select signal from the decoder to control when it responds to a bus transfer. The slave signals back to the master:

- success
- failure
- waiting of the data transfer

APB FSM Controller

APB Finite State Machine Controller has 7 states. They are

ST_IDLE

During this state the APB buses and PWRITE are driven with the last values they had, and PSEL and PENABLE lines are driven LOW. The ST_IDLE state is entered from:

- reset, when the system is initialized.
- ST_RENABLE,ST_WENABLE or ST_IDLE when there are no peripheral transfers to perform

The next state is:

- ST_READ,for a read transfer, when the AHB contains a valid APB read transfer
- ST_WWAIT for a write transfer, when the AHB contains a valid APB write transfer.

ST_READ

During this state the address is decoded onto PADDR ,the relevant PSEL line is driven HIGH, and PWRITE is driven LOW.A wait state is always inserted to ensure that the data phase of current AHB transfer does not complete until the APB read data has been driven onto HRDATA

The ST_READ state is entered from ST_IDLE,ST_RENABLE,ST_WENABLE or ST_WENABLEP during a valid read transfer.

The next state will always be ST_RENABLE.

ST_WWAIT

This state is need due to the pipelined structure of AHB transfers, to allow the AHB side of the write transfer to complete so that the write data becomes available on HWDATA.The APB write transfer is then started in the next clock cycle.

The ST_WWAIT state is entered from ST_IDLE,ST_RENABLE or ST_WENABLE

during a valid write transfer

The next state will always be ST_WRITE.

ST_WRITE

During this state the address is decoded and driven onto PADDR,the relevant PSEL line is driven HIGH, and PWRITE is driven HIGH.

A wait state is not inserted, as a single write transfer can complete without affecting the AHB.

The ST_WRITE state is entered from:

- ST_WWAIT, when there are no further peripheral transfers to perform
- ST_WENABLEP,when the currently pending peripheral transfer is a write, and there are no further transfers to perform.

The next state is:

- ST_WENABLE, when there are no further peripheral transfers to perform.
- ST_WENABLEP, when there is one further peripheral write transfer to perform.

ST_WRITEP

During this state the address is decoded and driven onto PADDR,the relevant PSEL line is driven HIGH, and PWRITE is driven HIGH.A wait state is always inserted as there must only ever be one pending transfer between the currently performed APB transfer and the currently driven AHB transfer.

The ST_WRITEP state is entered from:

- ST_WWAIT, when there is a further peripheral transfer to perform.
- ST_WENABLEP,when the currently pending peripheral transfer is a write and there is a further transfer to perform

The next state will always be ST_WENABLEP.

ST_RENABLE

During this state the PENABLE output is driven HIGH, Enabling the current APB transfer. All other APB outputs remain the same as the previous cycle.

The ST_RENABLE state is always entered from ST_READ.

The next state is:

- ST_READ, when there is a further peripheral read transfer to perform

- ST_WWAIT,when there is a further peripheral write transfer to perform
- ST_IDLE, when there are no further peripheral transfers to perform

ST_WENABLE

During this state the PENABLE output is driven HIGH, enabling the current APB transfer. All other APB outputs remain the same as the previous cycle.

The ST_WENABLE state is always entered from ST_WRITE

The next state is :

- ST_READ, when there is a further peripheral read transfer to perform
- ST_WWAIT,when there is further peripheral write transfer to perform
- ST_IDLE, when there are no further peripheral transfers to perform

ST_WENABLEP

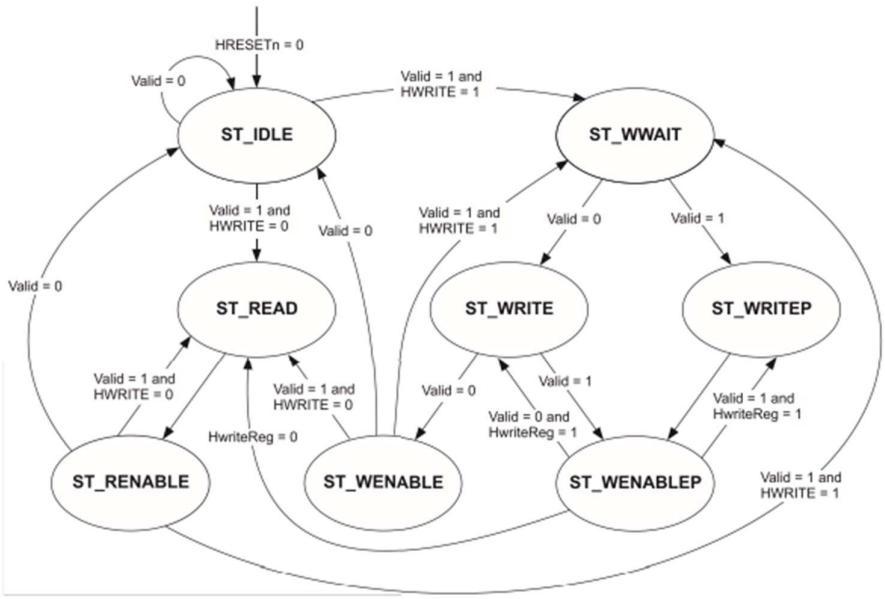
A wait state is inserted if the pending transfer is a read because, when a read follows a write, and extra wait state must be inserted to allow the write transfer to complete on the APB before the read is started.

The ST_WENABLEP state is entered from:

- ST_WRTIE,when the currently driven AHB transfer is a peripheral transfer
- ST_WRITEP,when there is a pending peripheral transfer following the current write.

The next state is:

- ST_READ, when the pending transfer is a read
- ST_WRITE,when the pending transfer is a write and there are no further transfers to perform.
- ST_WRITEP, when the pending transfer is a write and there is a further transfer to perform.



APB INTERFACE

APB interface include slave select is used to switch corresponding slave module and pipeline the address and data lines from the bridge module.

3.4 Objective

The main objectives to design and implement AHB2APB bridge are

- To facilitate synchronization between High-performance bus and Low-speed peripheral bus.
- To facilitate burst transfers
- Single-cycle/clock edge operation, bus master handover
- Synchronization of microcontrollers and slave devices.

Chapter 4

Tools Utilized

4.1 Intel Quartus prime tool

Intel Quartus Prime is programmable logic device design software produced by Intel; prior to Intel's acquisition of Altera the tool was called Altera Quartus Prime, earlier Altera Quartus II. Quartus Prime enables analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer. Quartus Prime includes an implementation of VHDL and Verilog for hardware description, visual editing of logic circuits, and vector waveform simulation.

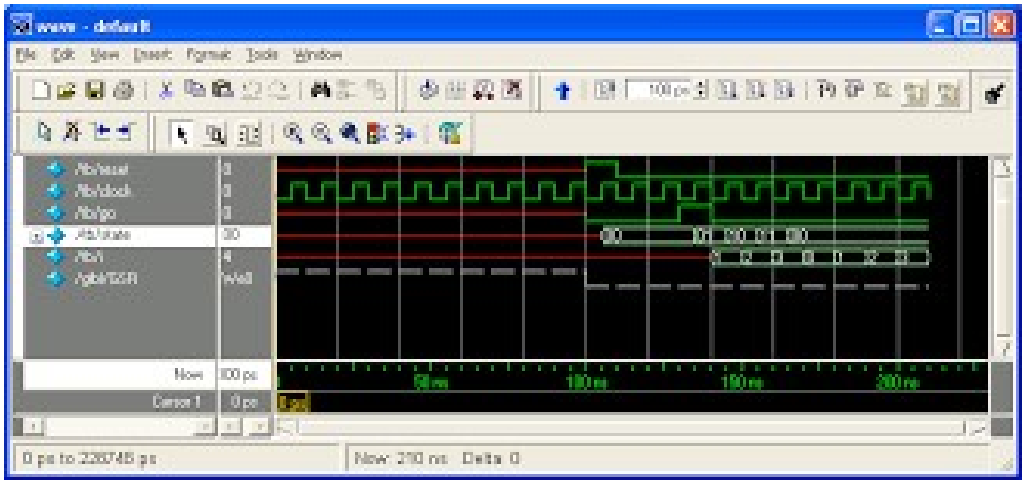
Quartus Prime software features include:

- Platform Designer (previously QSys, previously SOPC Builder), a tool that eliminates manual system integration tasks by automatically generating interconnect logic and creating a testbench to verify functionality.
- SoCEDS, a set of development tools, utility programs, run-time software, and application examples to help you develop software for SoC FPGA embedded systems.

-Cont.

- DSP Builder, a tool that creates a seamless bridge between the MATLAB/Simulink tool and Quartus Prime software, so FPGA designers have the algorithm development, simulation, and verification capabilities of MATLAB/Simulink system-level design tools.

4.2 Model Sim



ModelSim is a multi-language environment by Mentor Graphics, for simulation of hardware description languages such as VHDL, Verilog and SystemC, and includes a built-in C debugger. ModelSim can be used independently, or in conjunction with Intel Quartus Prime, PSIM, Xilinx ISE or Xilinx Vivado. Simulation is performed using the graphical user interface (GUI), or automatically using scripts.

4.3 Xilinx Design Suite

Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of hardware description language (HDL) designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis. Vivado represents a ground-up rewrite and re-thinking of the entire design flow (compared to ISE).

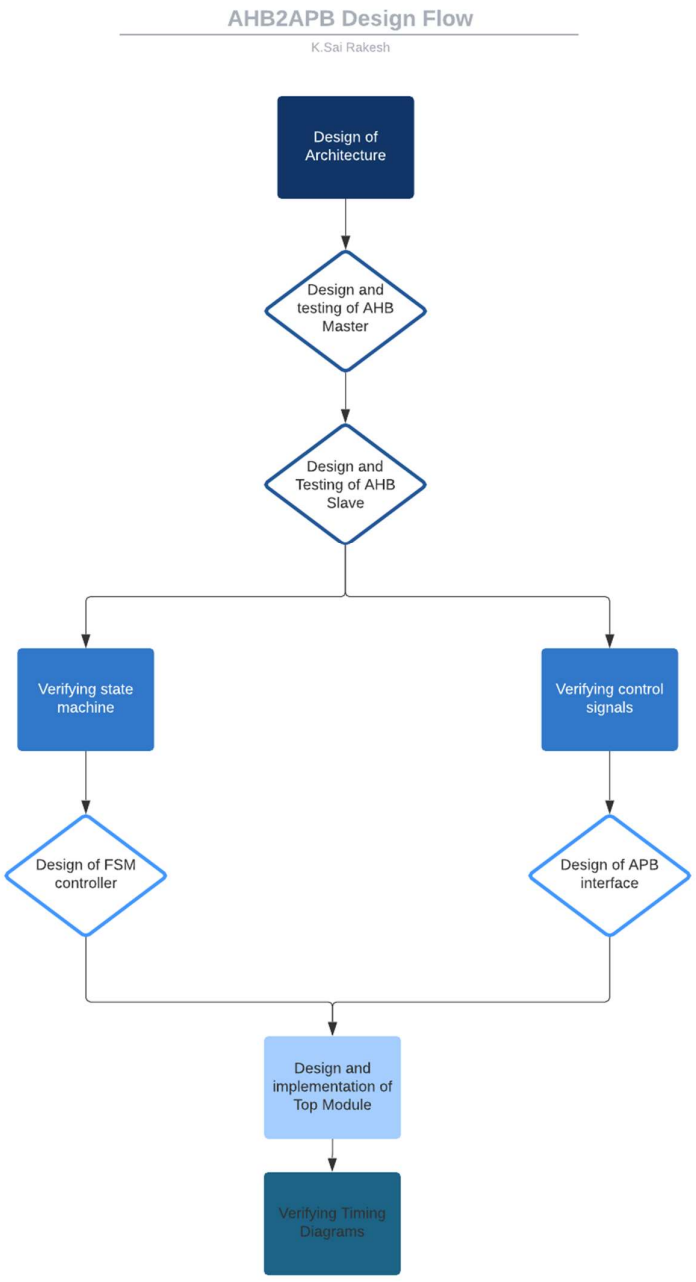
The Vivado-High-Level-Synthesis compiler enables C, C++ and SystemC programs to be directly targeted into Xilinx devices without the need to manually create RTL. Vivado HLS is widely reviewed to increase developer productivity, and is

confirmed to support C++ classes, templates, functions and operator overloading. Vivado 2014.1 introduced support for automatically converting OpenCL kernels to IP for Xilinx devices. OpenCL kernels are programs that execute across various CPU, GPU and FPGA platforms.

Chapter-5

Project Flow

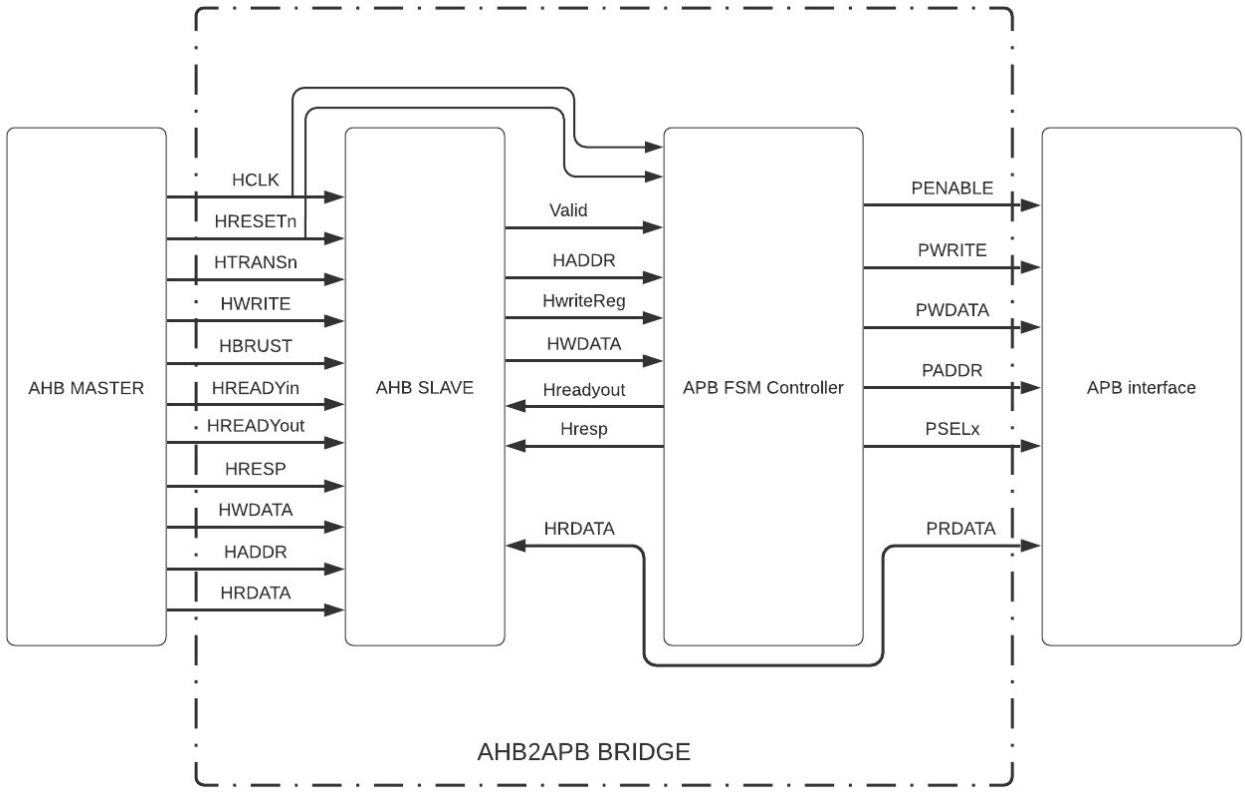
Flow Chart:



5.1 Designing of AHB2APB Required Modules

Complete Architecture including testing Modules

(Only SLAVE and FSM Controller are synthesizable)

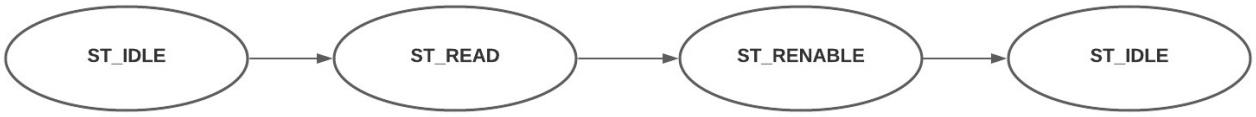


The above modules are coded in verilog vhdl and compiled in modelsim.The compiled modules are then individually tested with appropriate stimulus in the testbench codes.In this project we have implementing this bridge which can perform:

- Single Read
- Single Write

Single Read

During the read mode the FSM transverse its order in the following pattern

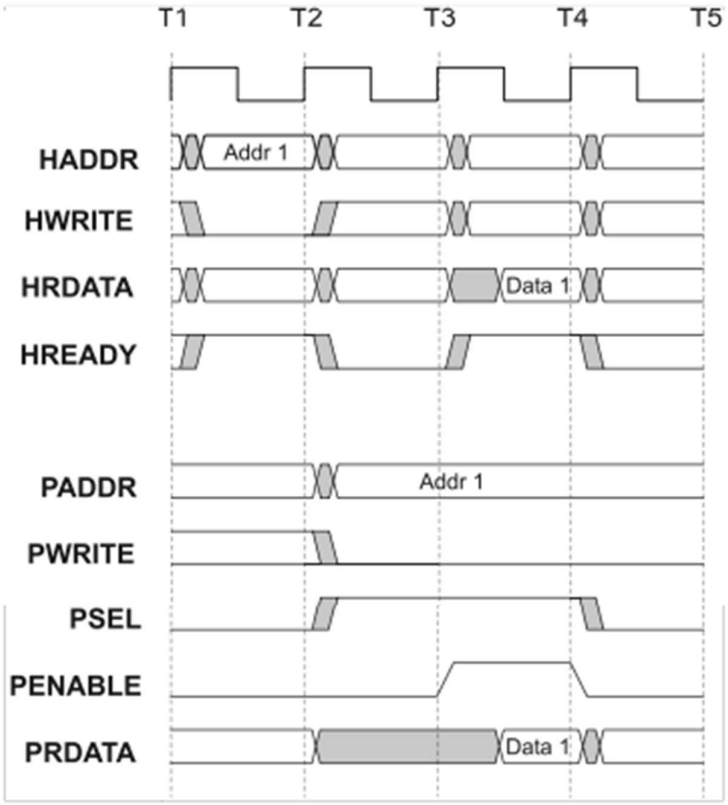


Single Write

During the read mode the FSM transverse its order in the following pattern

5.2 Timing Diagrams

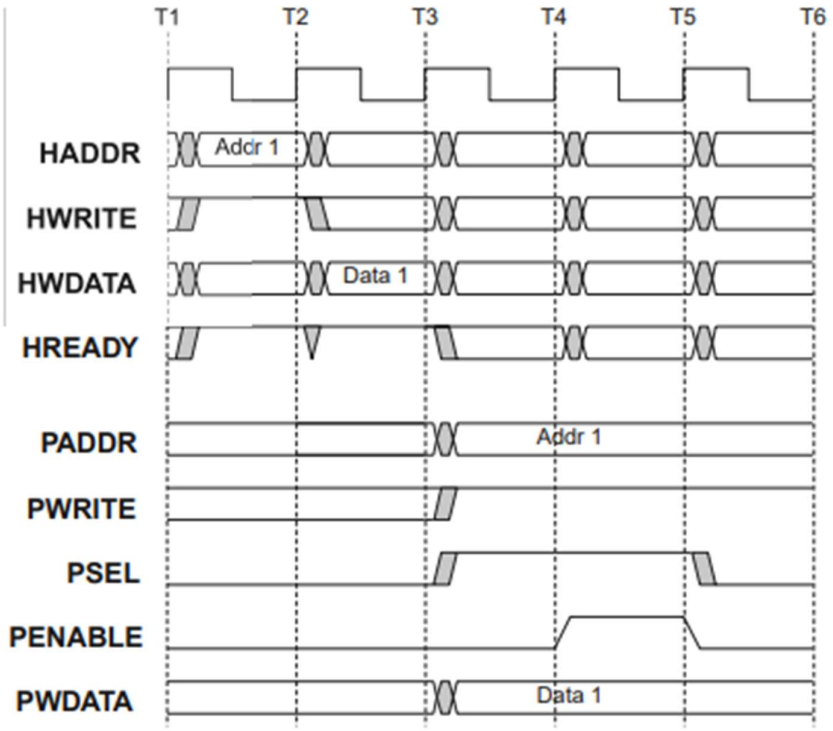
Read transfer



the transfer starts on the AHB at time T1 and the address is sampled by the Bridge at T2.If the transfer is for the peripheral bus then this address is broadcast and the appropriate peripheral select signal is generated. This first cycle on the peripheral bus is called the SETUP cycle, this is followed by the ENABLE cycle, when the PENABLE signal is asserted.

During the ENABLE cycle the peripheral must provide the read data. Normally it will be possible to route this read data directly back to the AHB,where the bus master can sample it on the rising edge of the clock at the end of the Enable cycle, which is at time T4

Write Transfer



Single write transfers to the APB can occur with zero wait states. The bridge is responsible for sampling the Address and data of the transfer and then holding these values for the duration of the write transfer on the APB.

5.3 Merging all modules under Top Module

Verilog HDL code Top Module

D: > Projects > Live Projects > Projects > project > bridgetop.v

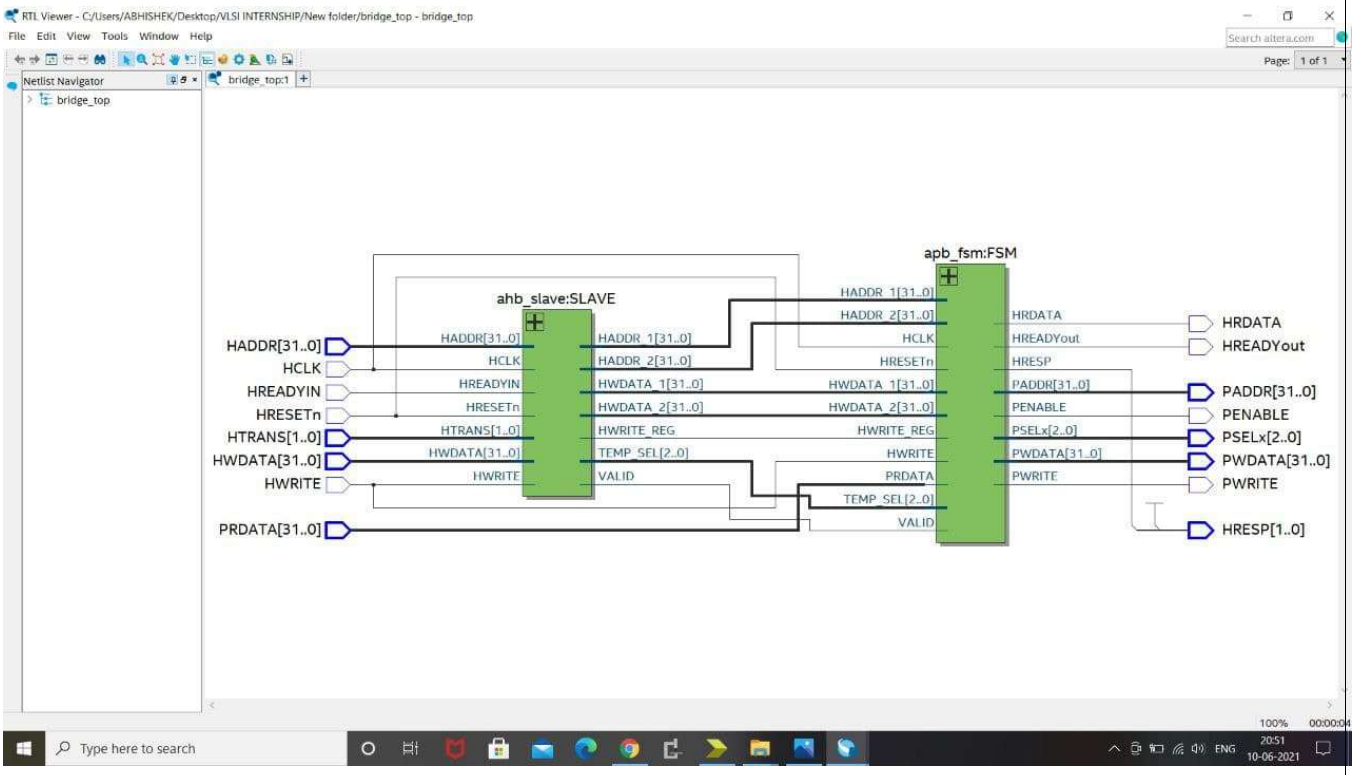
```
module bridgetop(   HCLK,HRESETn,HADDR,HWDATA,HWRITE,HTRANS,HREADYin,
                    HREADYout,HRDATA,HRESP,PENABLE,PWRITE,PSEL,PWDATA,
                    PRDATA,PADDR);

input HCLK,HRESETn;
input HWRITE,HREADYin;
input [1:0] HTRANS;
input [31:0] HADDR,HWDATA,PRDATA;

output PWRITE,PENABLE,HREADYout;
output [2:0]PSEL;
output  [1:0]HRESP;
output [31:0] PADDR,PWDATA,HRDATA;

wire [31:0] addr1,addr2,data1,data2;
wire writereg,vld;
wire [2:0] sel;

ahbslave AHBSLAVE(  .HCLK(HCLK),.HRESETn(HRESETn),.HADDR(HADDR),
                    .HADDR_1(addr1),.HADDR_2(addr2),.HWDATA(HWDATA),
                    .HWDATA_1(data1),.HWDATA_2(data2),.HWRITE(HWRITE),
                    .HWRITEreg(writereg),.HRESP(HRESP),.HRDATA(HRDATA),
                    .PRDATA(PRDATA),.TEMP_SEL(sel),.valid(vld),
                    .HTRANS(HTRANS),.HREADYin(HREADYin)
                 );
fsm FSM(  .HCLK(HCLK),.HRESETn(HRESETn),.HADDR_1(addr1),
          .HADDR_2(addr2),.HWDATA_1(data1),.HWDATA_2(data2),
          .HWRITE(HWRITE),.HWRITEreg(writereg),.HTRANS(HTRANS),
          .valid(vld),.TEMP_SEL(sel),.PADDR(PADDR),.PSEL(PSEL),
          .PWRITE(PWRITE),.PENABLE(PENABLE),.PWDATA(PWDATA),
          .HREADYout(HREADYout)
          );

endmodule
```

Chapter 6

Project Simulation Outputs

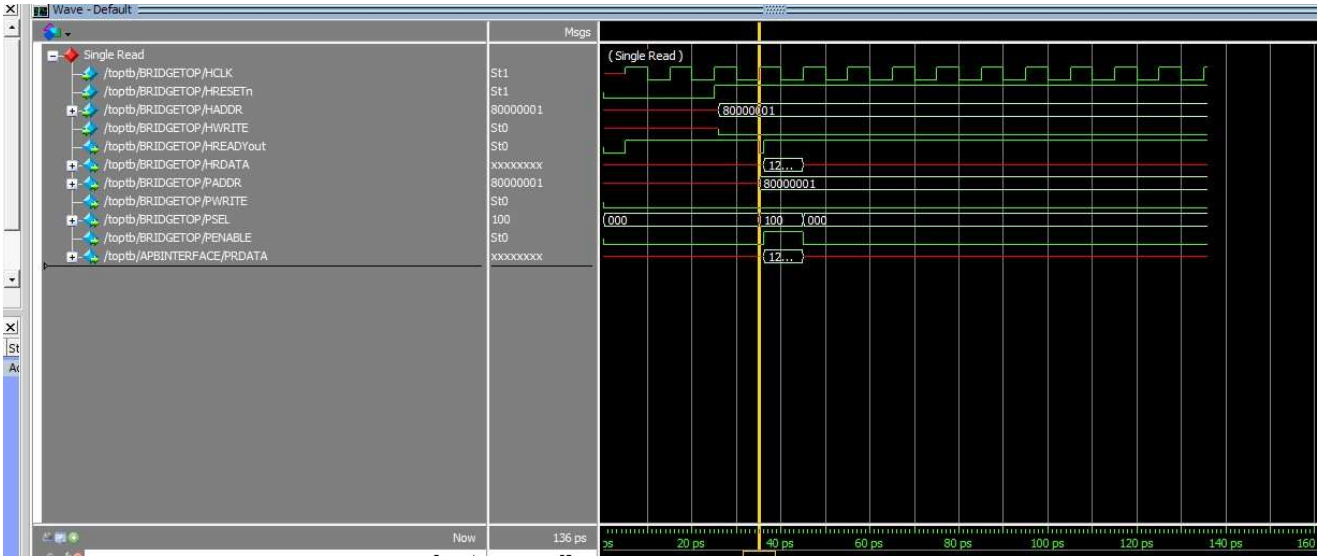
6.1 Synthesis

Synthesis of AHB2APB bridge is done in Intel Quartus prime tool, only slave and FSM controller are synthesizable and AHB MASTER, APB interface are used in testing of the bridge.

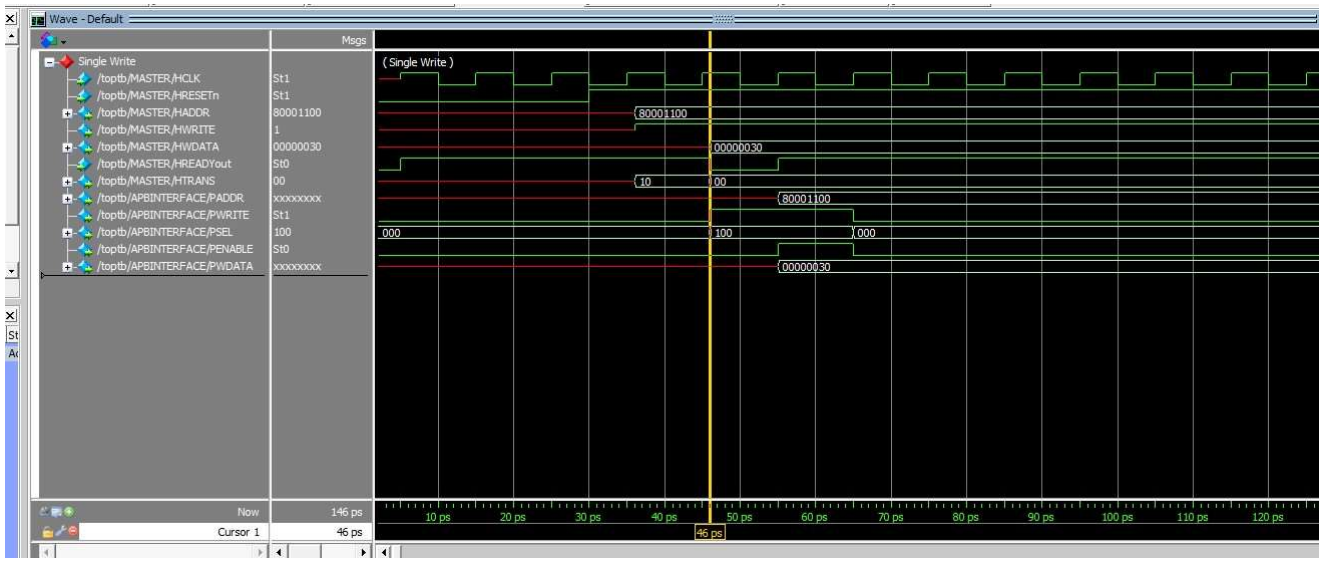


6.2 Timing diagrams

Single Read



Single Write



7.Appendix

7.1.Verilog code of FSM Controller

```
module fsm(
        HCLK,
        HRESETn,
        HADDR_1,
        HADDR_2,
        HWDATA_1,
        HWDATA_2,
        HWRITE,
        HWRITEreg,
        HTRANS,
        valid,
        TEMP_SEL,
        PADDR,
        PSEL,
        PWRITE,
        PENABLE,
        PWDATA,
        HREADYout
       );
//parameter declaration
parameter ST_IDLE=3'b000,          ST_WWAIT=3'b001,
        ST_READ=3'b010,          ST_WRITE=3'b011,
        ST_WRITEP=3'b100,        ST_RENABLE=3'b101,
        ST_WENABLE=3'b110,       ST_WENABLEP=3'b111
//input declaration
input HCLK,HRESETn;
input valid,HWRITE,HWRITEreg;
input [1:0] HTRANS;
input [2:0] TEMP_SEL;
input [31:0] HADDR_1,HADDR_2,HWDATA_1,HWDATA_2;
//output declaration

output reg PWRITE,PENABLE,HREADYout;
output reg [31:0] PADDR,PWDATA;
output reg [2:0] PSEL;

//TEMP variables
reg [2:0] state,nstate;
reg [31:0] tmpaddr;
//reset state
always@(posedge HCLK,negedge HRESETn)
begin
   if(~HRESETn)
   begin
      state <= ST_IDLE;
   end
   else
   begin
      state<=nstate;
   end
end
```

```
// state transistion conditions
always@(*)
 begin
  //state=nstate;
    nstate=ST_IDLE;
  PSEL = 0;
  PENABLE = 0;
  HREADYout = 0;
     PWRITE = 0;
  case(state)

       ST_IDLE:begin
       //    PSEL=0;
              //  PENABLE=0;
                HREADYout=1'b1;

                if(~valid)
                  nstate=ST_IDLE;
                else if(valid && HWRITE)
                  nstate=ST_WWAIT;
                else if (valid == 1 && HWRITE == 0)
                  nstate=ST_READ;
              end

   ST_WWAIT:begin
                HREADYout=1'b1;
                if(valid)
                  nstate=ST_WRITE;
                else if(~valid)
                  nstate=ST_WRITEP;
              end
  ST_READ: begin
                PSEL=TEMP_SEL;
                //PENABLE=1'b0;
                PADDR=HADDR_1;
                //PWRITE=1'b1;
               HREADYout=1'b0;
                nstate=ST_RENABLE;
              end
  ST_WRITE:begin
                PSEL=TEMP_SEL;
               // PADDR=HADDR_1;
                PWRITE=1'b1;
                PWDATA=HWDATA_1;
                PENABLE=1'b0;
                 if(~valid)
                    nstate=ST_WENABLE;
                 else if(valid)
                    nstate=ST_WENABLEP;
                end
```

7.2 Verilog code for AHB MASTER

```
Task singleread();
begin
  @(posedge HCLK);
  #1;
   HADDR=32'h8000_0001;
   HWRITE=1'b0;
   HTRANS=NONSEQ;
   HREADYin=1'b1;
   @(posedge HCLK);
   #1;
   begin
     // HWRITE=1'b1;
      HTRANS=IDLE;
   end
end
endtask


//single write

task singlewrite();
begin
   @(posedge HCLK)
   #1;
   begin
   HADDR=32'h8000_1100;
   HWRITE=1'b1;
   HTRANS=NONSEQ;
   HREADYin=1'b1;
   end
   @(posedge HCLK)
   #1;
   begin
      HTRANS=IDLE;
      HWDATA=$random()%100;
   end
end
endtask

endmodule
```

7.3 Verilog code for AHB SLAVE

```
module ahbslave(
   HCLK,   HRESETn,   HADDR,   HADDR_1,
   HADDR_2,   HWDATA,   HWDATA_1,   HWDATA_2,
   HWRITE,   HWRITEreg,   HRESP,   HRDATA,
   PRDATA,   TEMP_SEL,   valid,   HTRANS,
   HREADYin
);
//slave memory map
parameter slave0_0=32'h8000_0000,slave0_1=32'h8400_0000;
parameter slave1_0=32'h8400_0001,slave1_1=32'h8800_0000;
parameter slave2_0=32'h8800_0001,slave2_1=32'h8c00_0000;
parameter   IDLE=2'b00,WAIT=2'b01,
        NONSEQ=2'b10,SEQ=2'b11;
//Input declaration
input HCLK,HRESETn;
```

```
input HREADYin,HWRITE;
input [1:0] HTRANS;
input [31:0] HADDR,HWDATA,PRDATA;
//Output declaration
//reg internal register
output reg HWRITEreg,valid;
output reg [1:0]HRESP;
output reg [2:0] TEMP_SEL;
output reg [31:0] HRDATA,HADDR_1,HADDR_2,HWDATA_1,HWDATA_2;
//assigning prdata = hr data
assign HRDATA=PRDATA;
//pipelining
always@(posedge HCLK,negedge HRESETn)
begin
   HRESP=0;
   if(~HRESETn)
   begin
      HADDR_1<=0;
      HADDR_2<=0;
      HWDATA_1<=0;
      HWDATA_2<=0;
   end
   else
   begin
      HADDR_1<=HADDR;
      HADDR_2<=HADDR_1;
      HWDATA_1<=HWDATA;
      HWDATA_2<=HWDATA_1;
   end
end
//hwritereg transfer
always@(*)
begin
    if((HADDR>=slave0_0 && HADDR<= slave2_1) && (HTRANS!=IDLE && HTRANS!=WAIT) &&
(HREADYin==1'b1))
       valid = 1'b1;
   else
       valid=1'b0;
end
// tempsel output generation
always@(*)
begin
  if(slave0_1>=HADDR>=slave0_0)
                        TEMP_SEL=3'b001;
  else if(slave1_1>=HADDR>=slave1_0)
                        TEMP_SEL=3'b010;
  else if(slave2_1>=HADDR>=slave2_0)
              TEMP_SEL=3'b100;
  else
              TEMP_SEL=3'b100;
end
always@(posedge HCLK,negedge HRESETn)
begin
  if(~HRESETn)
              HWRITEreg <= 0;
  else
              HWRITEreg<= HWRITE;
end
endmodule
```

7.4 Verilog code for APB Interface

```
module apbif(
            PENABLEin,              PWRITEin,
            PSELin,             PADDRin,
            PWDATAin,               PENABLE,
            PWRITE,             PSEL,
            PADDR,              PWDATA,
            PRDATA
          );
//input declaration
input PENABLEin,PWRITEin;
input[2:0] PSELin;
input [31:0] PADDRin,PWDATAin;
output  PENABLE,PWRITE;
output [2:0] PSEL;
output [31:0] PADDR,PWDATA;
output reg[31:0] PRDATA;
//port assigment
assign PENABLE=PENABLEin;
assign PWDATA=PWDATAin;
assign PWRITE=PWRITEin;
assign PSEL=PSELin;
assign PADDR=PADDRin;
//prdata generation
always@(*)
begin
if(PWRITEin==0)
   if(PENABLEin)
      PRDATA=$random();
else
   PRDATA=32'bx;
end
endmodule
```

7.5 Testbench of TopModule

```
module toptb();
reg HCLK;
reg HRESETn=0;
wire [31:0]pwdata,prdata,paddr;
wire [2:0] psel;
wire pwrite,penable;
wire [31:0]pwdatao,prdatao,paddro;
wire [2:0] pselo;
wire pwriteo,penableo;
wire [1:0]Hresp;
wire hwrite;

parameter cycle=10;
ahbmaster MASTER(
              .HCLK(HCLK),                .HRESETn(HRESETn),
              .HWRITE(hwrite),              .HREADYin(BRIDGETOP.HREADYin),
              .HADDR(BRIDGETOP.HADDR),              .HWDATA(BRIDGETOP.HWDATA),
              .HRDATA(BRIDGETOP.HRDATA),              .HRESP(Hresp),
              .HREADYout(BRIDGETOP.HREADYout),              .HTRANS(BRIDGETOP.HTRANS)
);
```

```
bridgetop BRIDGETOP(    .HCLK(HCLK),                    .HRESETn(HRESETn),
                .HADDR(MASTER.HADDR),                   .HWDATA(MASTER.HWDATA),
                .HWRITE(MASTER.HWRITE),                 .HTRANS(MASTER.HTRANS),
                .HREADYin(MASTER.HREADYin),                 .HREADYout(MASTER.HREADYout),
                .HRDATA(MASTER.HRDATA),                 .HRESP(Hresp),
                .PENABLE(penable),                  .PWRITE(pwrite),
                .PSEL(psel),                .PWDATA(pwdata),
                .PRDATA(prdatao),                   .PADDR(paddr)
            );

apbif APBINTERFACE(
                .PENABLEin(penable),
                .PWRITEin(pwrite),
                .PSELin(psel),
                .PADDRin(paddr),
                .PWDATAin(pwdata),
                .PENABLE(penableo),
                .PWRITE(pwriteo),
                .PSEL(pselo),
                .PADDR(paddro),
                .PWDATA(pwdatao),
                .PRDATA(prdatao)
          );

//testing
//clock generation
always
begin
   #5;
   HCLK=1'b1;
   #5;
   HCLK=~HCLK;
end

//reset task
task rst();
begin
   HRESETn=1'b0;
   #5;
   HRESETn=1'b1;
end
endtask

initial
begin
   #20;
   rst();
   MASTER.singlewrite();
//MASTER.singleread();
#100 $finish();
end
endmodule
```