Maven Silicon

Advanced Design and Verification

Report on

AHB TO APB Bridge Verification

Submitted by

MadhuSudhan.B

BRN-38

INTRODUCTION TO AMBA-2.0 AHB TO APB

Overview of AMBA:-

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

Three buses are defined within the AMBA specification:

- The Advanced High-performance Bus (AHB).
- The Advanced System Bus (ASB.
- The Advanced Peripheral Bus (APB).

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 macrocell functions. AHB is also specified to ensure ease of use in an efficient design flow using synthesis and automated test techniques.

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. APB can be used in conjunction with either version of the system bus.

Advanced System Bus (ASB):-

The AMBA ASB is for high-performance system modules. AMBA ASB is an alternative system bus suitable for use where the high-performance features of AHB are not required. ASB also supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macrocell functions.

This section describes each of the elements in an AMBA system and provides the generic timing parameters that are required to analyze an ASB-based AMBA design.

The following notation is used for the timing parameters:

- Tis - input setup time
- Tih - input hold time

• Tov - output valid time • Toh - output hold time

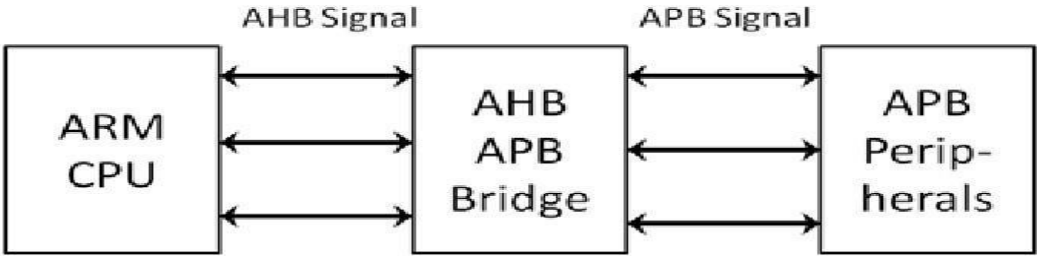


Fig:- 1 Application of AMBA AHB To APB

A-Typical-AMBA-based-microcontroller :-

An AMBA-based microcontroller typically 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 a bridge to the lower bandwidth APB, where most of the peripheral devices in the system are located.

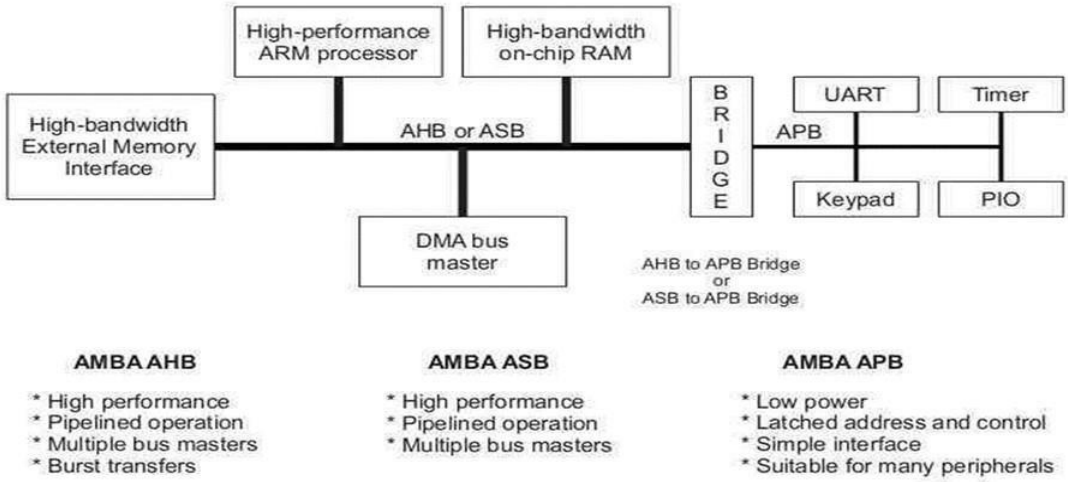


Fig:-2 A-Typical-AMBA-based-microcontroller

Introduction To AMBA AHB:-

AMBA AHB is a bus interface suitable for high-performance synthesizable

designs. It defines the interface between components, such as masters, interconnects, and slaves. AMBA AHB implements the features required for high-performance, high clock frequency systems including:-

• Burst transfers.
• Single clock-edge operation.
• Non-tristate implementation.
• Wide data bus configurations, 64, 128, 256, 512, and 1024 bits.

The most common AHB slaves are internal memory devices, external memory interfaces, and high-bandwidth peripherals. Although low-bandwidth peripherals can be included as AHB slaves, for system performance reasons, they typically reside on the AMBA Advanced Peripheral Bus (APB). Bridging between the higher performance AHB and APB is done using an AHB slave, known as an APB bridge.

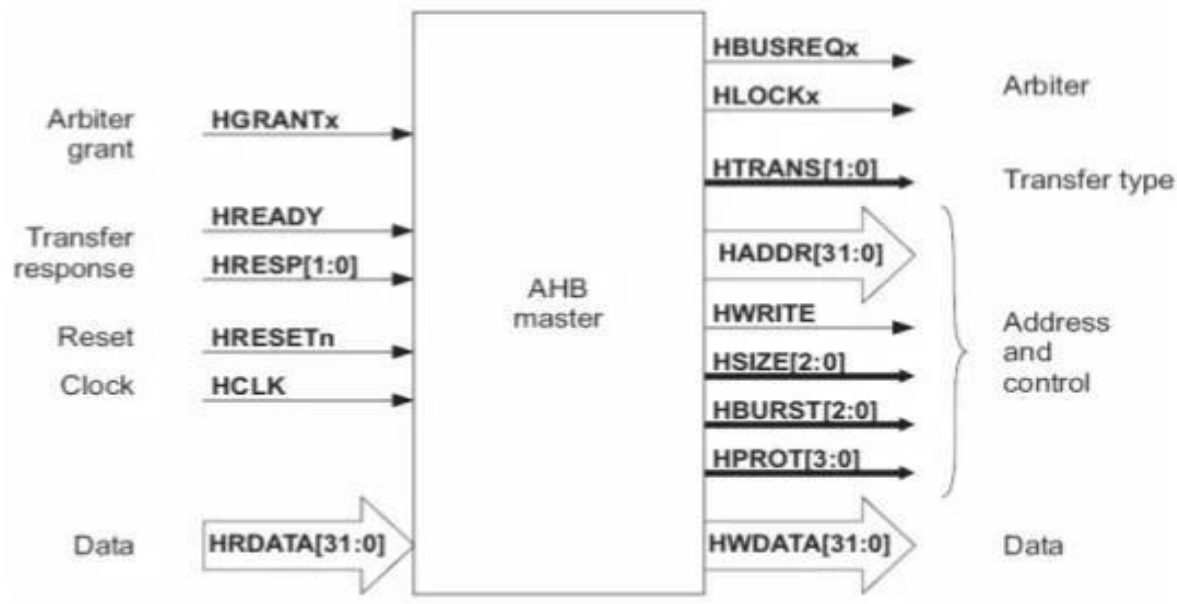


Fig:-3 AHB Master Interface

Bus Interconnection of AMBA AHB To APB:-

The AMBA AHB bus protocol is designed to be used with a central multiplexor interconnection scheme. Using this scheme all bus masters drive out the address and control signals indicating the transfer they wish to perform and the arbiter determines which master has its address and control signals routed to all of the slaves. A central decoder is also required to

control the read data and response signal multiplexor, which selects the appropriate signals from the slave that is involved in the transfer.

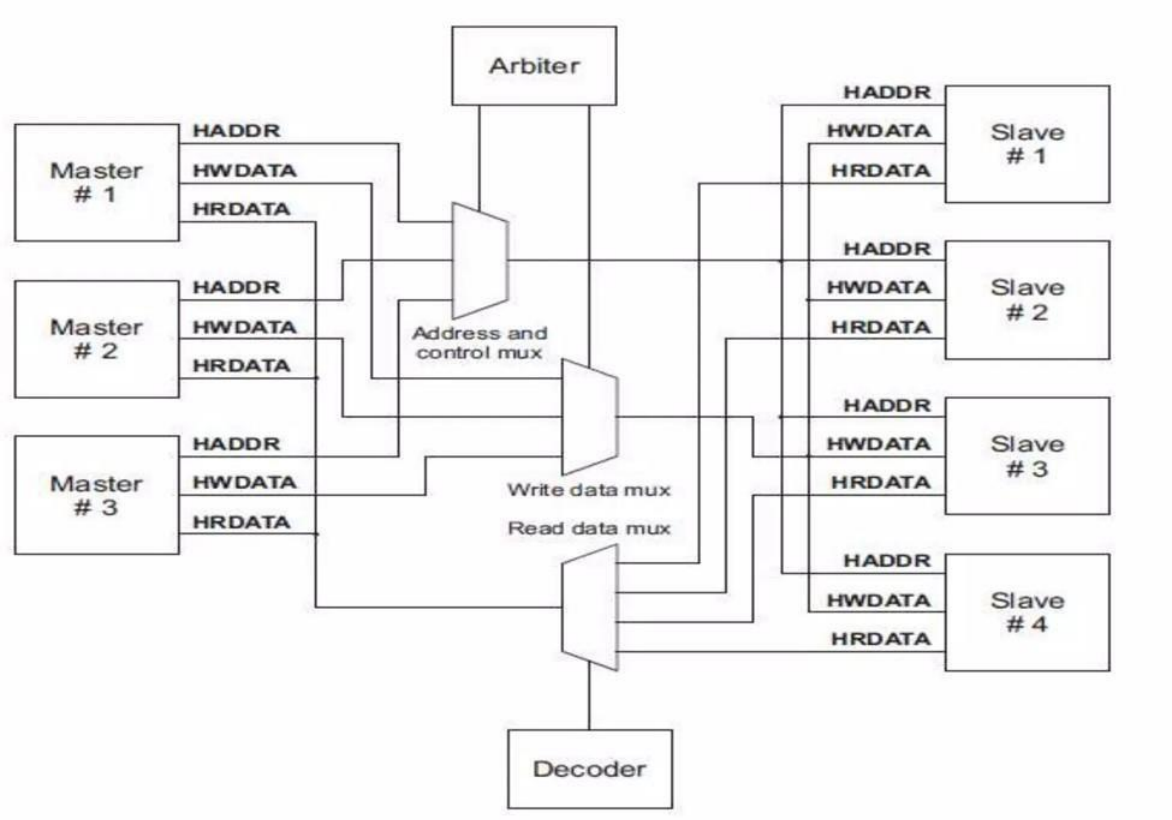


Fig:-4 AMBA AHB To APB Bus Interconnection

AHB Operation:-

During a transfer the slave shows the status using the response signals, HRESP[1:0]:-

OKAY The OKAY response is used to indicate that the transfer is progressing normally and when HREADY goes HIGH this shows the transfer has completed successfully.

ERROR The ERROR response indicates that a transfer error has occurred and the transfer has been unsuccessful.

RETRY and SPLIT Both the RETRY and SPLIT transfer responses indicate that the transfer cannot complete immediately, but the bus master should continue to attempt the transfer. In normal operation a master is allowed to complete all the transfers in a particular burst

Basic transfer:-

An AHB transfer consists of two distinct sections:
• The address phase, which lasts only a single cycle.
• The data phase, which may require several cycles.
This is achieved using the HREADY signal

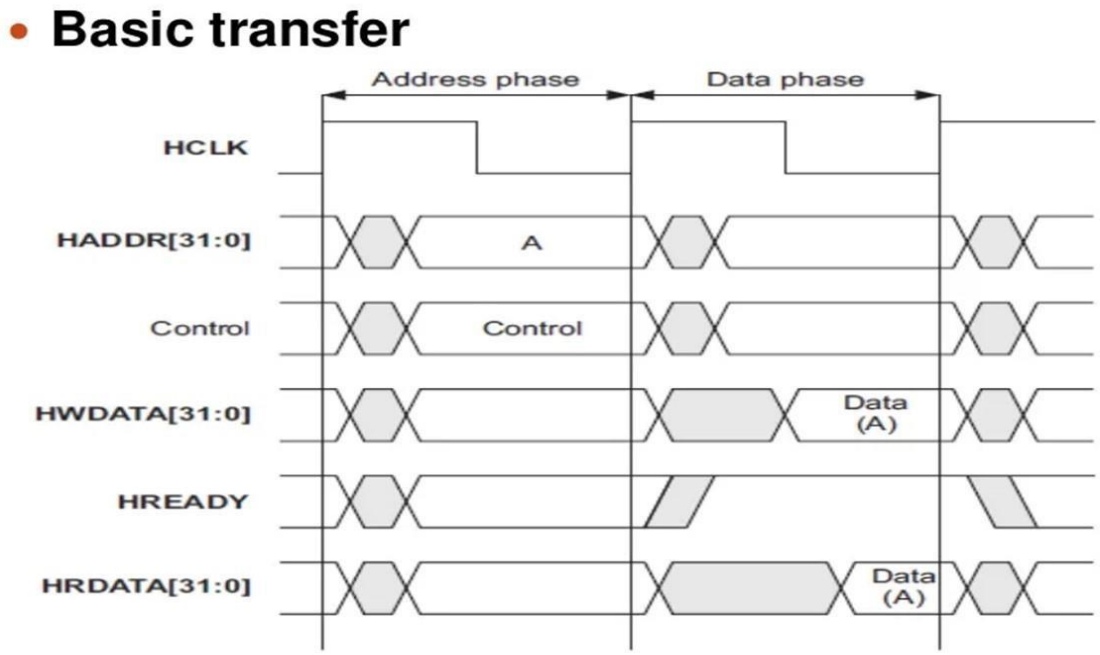


Fig::-5 Basic Transfer

Burst Operation:-

HBURST[2:0]	Type	Description
000	SINGLE	Single transfer
001	INCR	Incrementing burst of unspecified length
010	WRAP4	4-beat wrapping burst

011	INCR4	4-beat incrementing burst
100	WRAP8	8-beat wrapping burst
101	INCR8	8-beat incrementing burst
110	WRAP16	6-beat wrapping burst
111	INCR16	16-beat incrementing burst

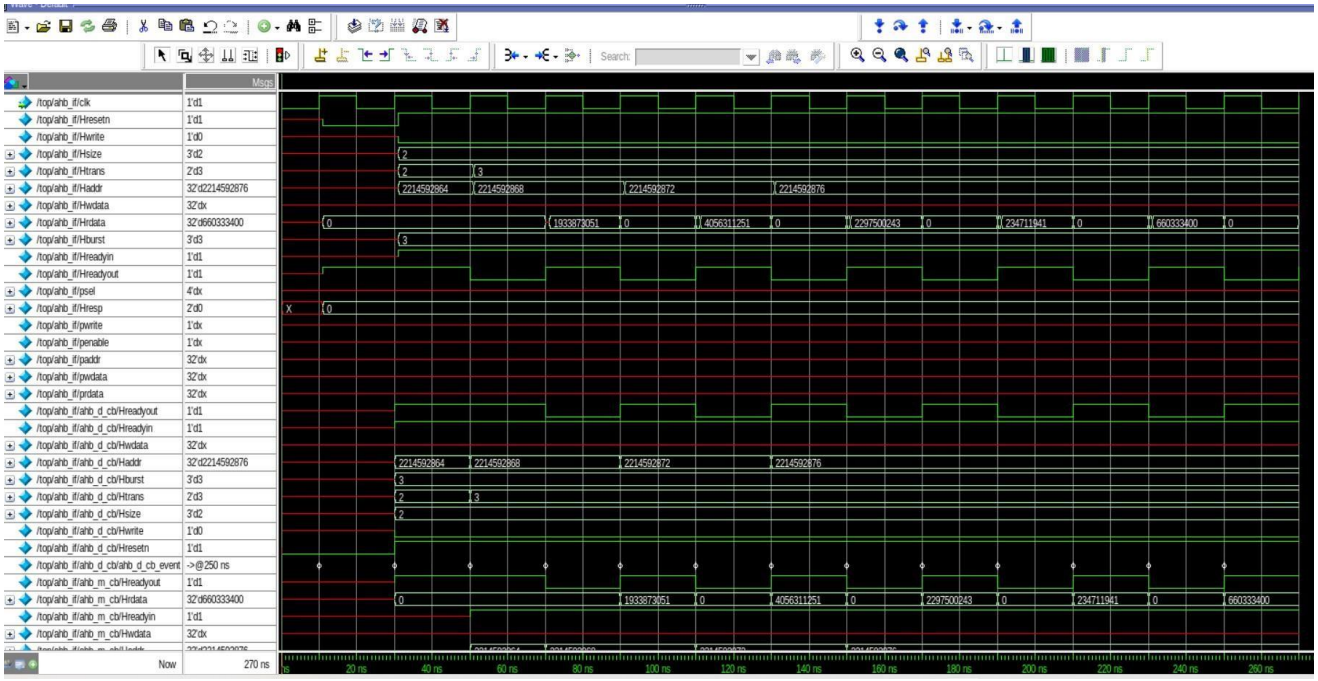


Fig:- 8 AHB Write Transfer

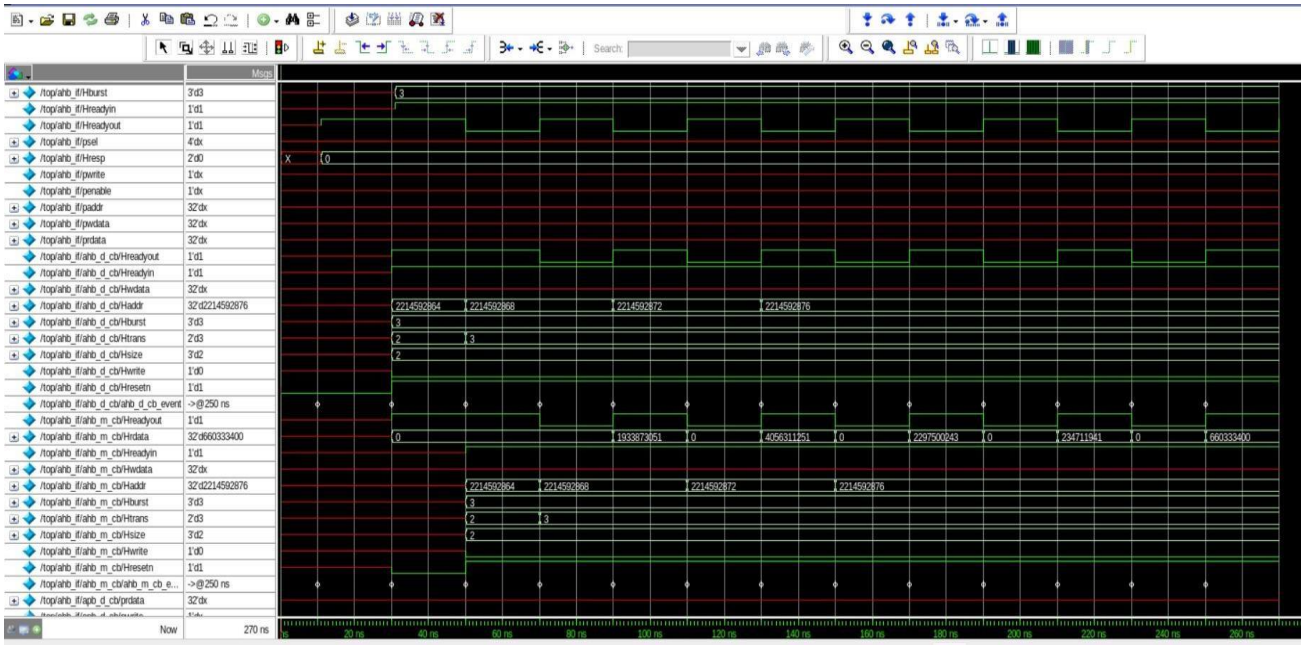


Fig:- 9 AHB Read Transfer

Introduction To AMBA APB:-

The Advanced Peripheral Bus (APB) is part of the Advanced Microcontroller Bus Architecture (AMBA) protocol family. It defines a low-cost interface that is optimized for minimal power consumption and reduced interface complexity. The APB protocol is not pipelined, use it to connect to low-bandwidth peripherals that do not require the high performance of the AXI protocol. The APB protocol relates a signal transition to the rising edge of the clock, to simplify the integration of APB peripherals into any design flow. Every transfer takes at least two cycles.

The APB can interface with:

- AMBA Advanced High-performance Bus (AHB)
- AMBA Advanced High-performance Bus Lite (AHB-Lite)
- AMBA Advanced Extensible Interface (AXI)
- AMBA Advanced Extensible Interface Lite (AXI4-Lite)

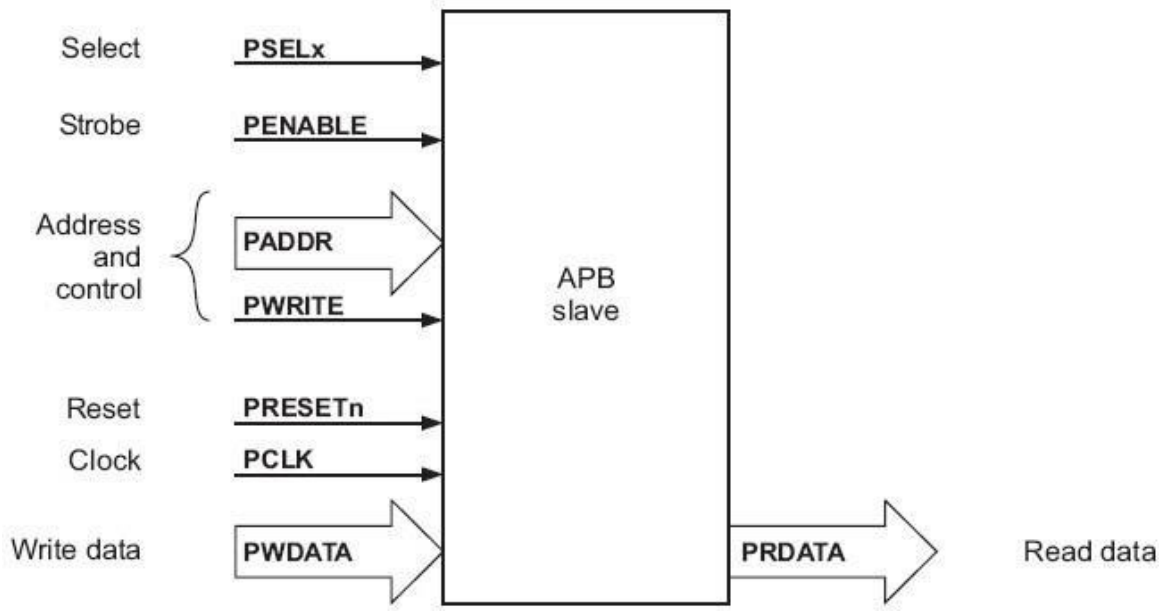


Fig:- 10 APB Slave Interface

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 ensures that all signal transitions are only related to the rising edge of the clock. This improvement means the APB peripherals can be integrated easily into any design .

The following advantages of APB:-

- performance is improved at high-frequency operation.
- performance is independent of the mark-space ratio of the clock .
- static timing analysis is simplified by the use of a single clock edge.
- no special considerations are required for automatic test insertion.
- manyApplication-Specific Integrated Circuit (ASIC) libraries have a better selection of rising edge registers.
- easy integration with cycle based simulator.

AHB To APB Block Diagram:-

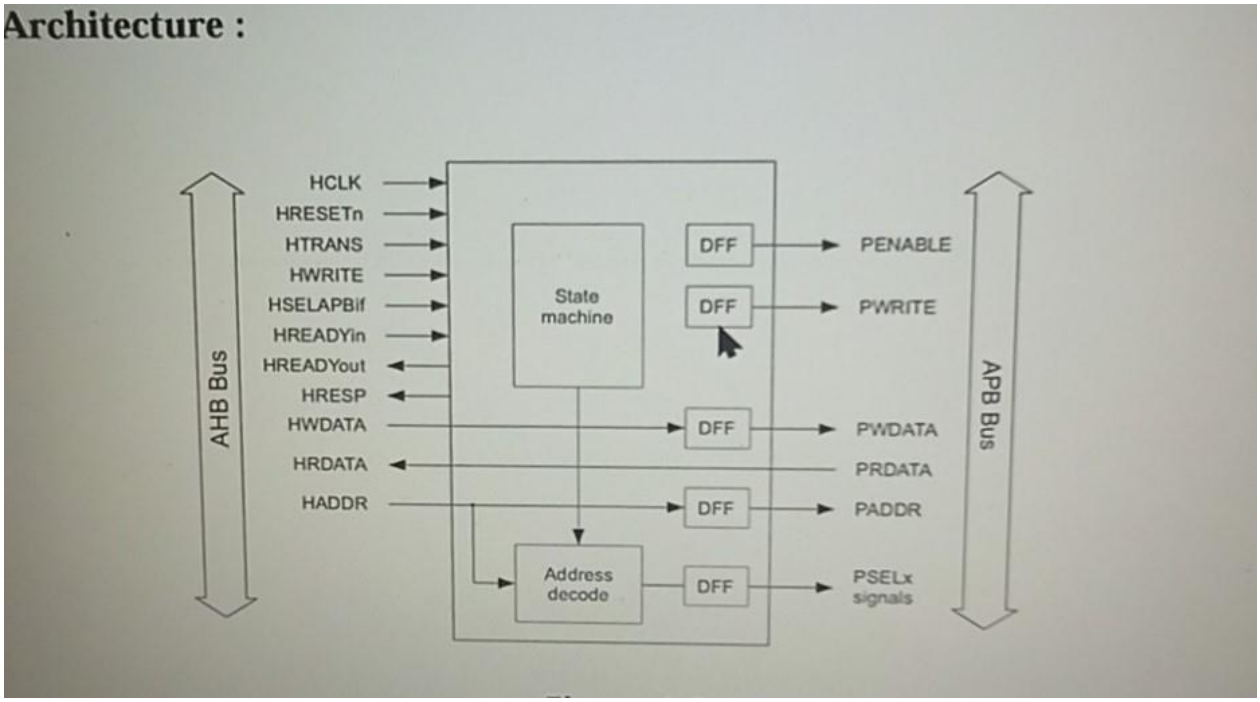


Fig:-11 AHB To APB Block Diagram

The main sections of this module are:-

• AHB slave bus interface
• APB transfer state machine, which is independent of the device memory map
• APB output signal generation.

To add new APB peripherals, or alter the system memory map, only the address decode sections need to be modified. The base addresses of each of the peripherals (timer, interrupt controller, and remap and pause controller) are defined in the AHB to APB bridge interface, which selects the peripheral according to its base address. The whole APB address range is also defined in the bridge. These base addresses can be implementation-specific. The peripherals standard specifies only the register offsets (from an unspecified base address), register bit meaning, and minimum supported function.

The APB data bus is split into two separate directions: -

• read (PRDATA), where data travels from the peripherals to the bridge.
• write (PWDATA), where data travels from the bridge to the peripherals.

This simplifies driving the buses because turnaround time between the peripherals and bridge is avoided. In the default system, because the bridge is the only master on the bus, PWDATA is driven continuously. PRDATA is a multiplexed connection of all peripheral PRDATA outputs on the bus, and is only driven when the slaves are selected by the bridge during APB read transfers. It is possible to combine these two buses into a single bidirectional bus, but precautions must be taken to ensure that there is no bus clash between the bridge and the peripherals.

The APB bridge responds to transaction requests from the currently granted AHB master.

The AHB transactions are then converted into APB transactions.

- the AHB transactions with the HREADYout signal
- the generation of all APB output signals.

The individual PSELx signals are decoded from HADDR, using the state machine to enable the outputs while the APB transaction is being performed. If an undefined location is accessed, operation of the system continues as normal, but no peripherals are selected.

Signals Description :-

Signals	Type	Direction	Description
HCLK	Bus clock	Input	This clock times all bus transfers.
HRESETn	Reset	Input	The bus reset signal is active LOW, and is used to reset the system and the bus.

Signals	Type	Direction	Description
HCLK	Bus clock	Input	This clock times all bus transfers.
HADDR[31:0]	Address	Input	The 32-bit system address bus.
HTRANS[1:0]	Transfer type	Input	This indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.

HWRITE	Transfer direction	Input	When HIGH this signal indicates a write transfer, and when LOW, a read transfer.
HWDATA[31:0]	Write data bus	Input	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HSELAPB if	Slave select	Input	Each APB slave has its own slave select signal, and this signal indicates that the current transfer is intended for the selected slave. This signal is a combinatorial decode of the address bus.
HRDATA[31:0]	Read data bus	Output	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HREADYin HREADYout	Transfer done	Input/Output	When HIGH the HREADY signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer.
HRESP[1:0]	Transfer response	Output	The transfer response provides additional information on the status of a transfer. This module will always generate the OKAY response.
PRDATA[31:0]	Peripheral read data bus	Input	The peripheral read data bus is driven by the selected peripheral bus slave during read cycle.
PWDATA[31:0]	Peripheral write data bus	Output	The peripheral write data bus is continuously driven by this module, changing during write cycles (when PWRITE is HIGH).
PENABLE	Peripheral enable	Output	This enable signal is used to time all accesses on the peripheral bus. PENABLE goes HIGH on the second clock rising edge of the transfer, and LOW

Signals	Type	Direction	Description
HCLK	Bus clock	Input	This clock times all bus transfers.
			on the third (last) rising clock edge of the transfer.
PSELx	Peripheral slave select	Output	There is one of these signals for each APB peripheral present in the system. The signal indicates that the slave device is selected, and that a data transfer is required. It has the same timing as the peripheral address bus. It becomes HIGH at the same time as PADDR, but will be set LOW at the end of the transfer.
PADDR[31:0]	Peripheral address bus	Output	This is the APB address bus, which may be up to 32 bits wide and is used by individual peripherals for decoding register accesses to that peripheral. The address becomes valid after the first rising edge of the clock at the start of the transfer. If there is a following APB transfer, then the address will change to the new value, otherwise it will hold its current value until the start of the next APB transfer.
PWRITE	Peripheral transfer direction	Output	This signal indicates a write to a peripheral when HIGH, and a read from a peripheral when LOW. It has the same timing as the peripheral address

APB Output waveform:-

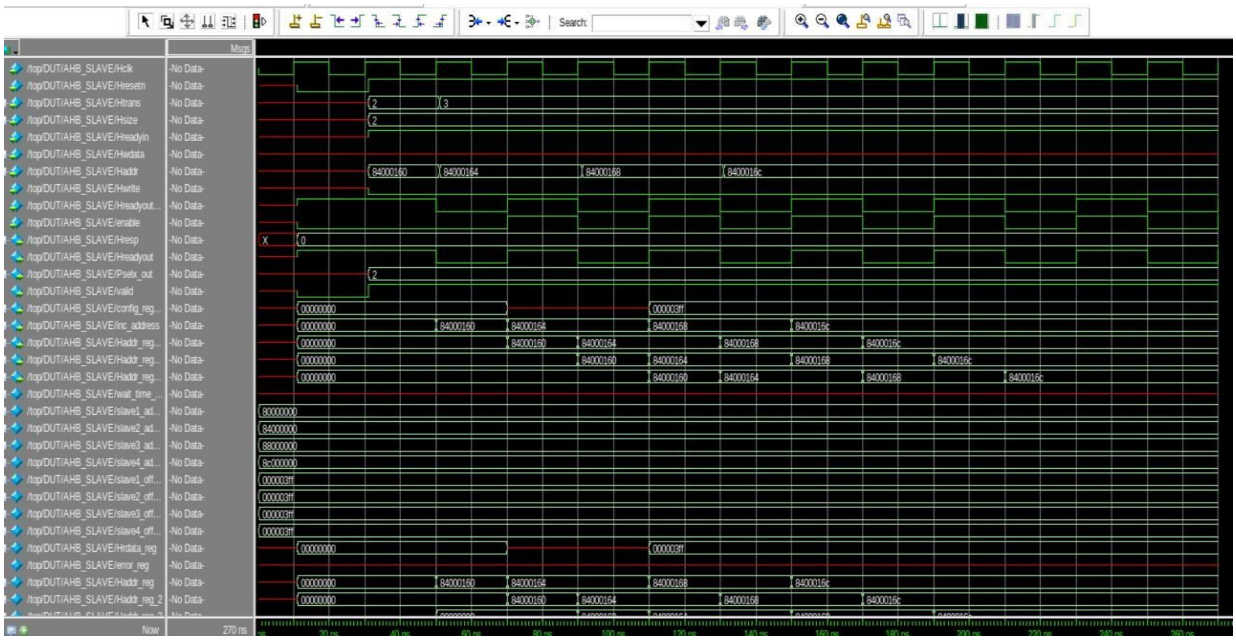


Fig:-12 APB Write Transfer

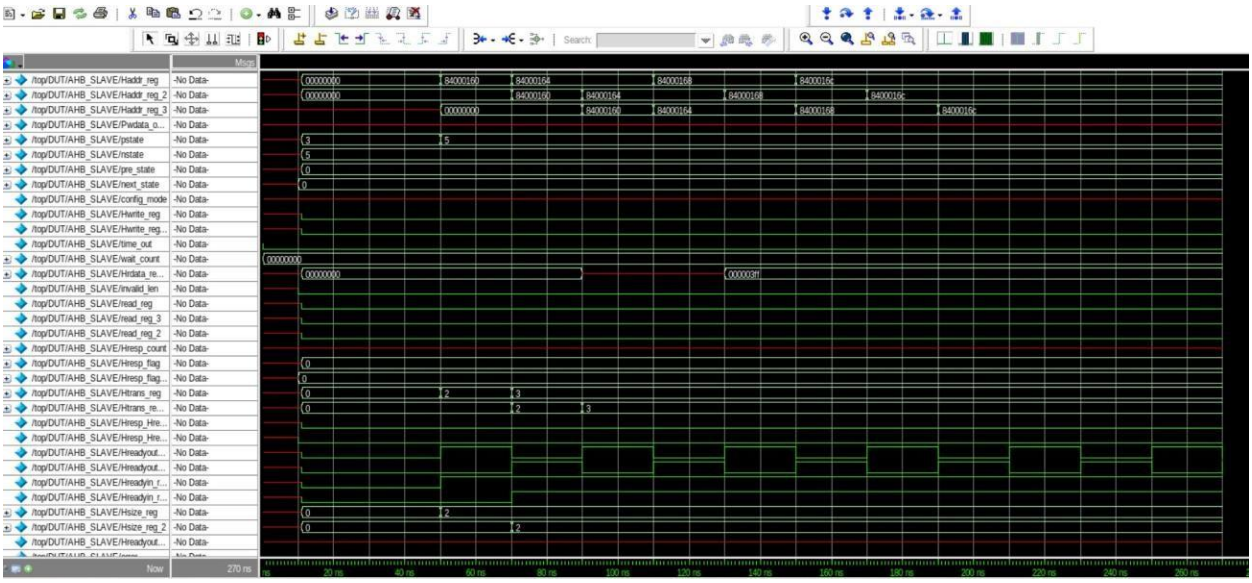


Fig:-13 APB Read signal

Scoreboard Description:-

1. We need to write test cases to cover all scenarios by writing test case and constraint so that we can improve percentages.

2. For AHB , APB signal We can write covergroup , coverpoints ,bins to improve the percentage

3. AHB Coverage Model :-

We can write coverpoint and bins for AHB signals HADDR,HSIZE ,HTRANS HWRITE HWDATA , HRDATA to improve the percentage .

Cross bins for AHB Signal:-

AHB_W_CROSS : cross HADDR,HSIZE,HTRANS,HWRITE,HWDATA.

AHB_R_CROSS : cross HADDR,HSIZE, HTRANS, HWRITE, HRDATA.

4. APB Coverage Model :-

We can write coverpoint and bins for ahb signals PADDR , PWRITE ,PSEL PWDATA ,PRDATA to improve the percentage .

Cross bins for APB Signals:-

APB_W_CROSS : cross PADDR,PWRITE,PSEL,PWDATA.

APB_R_CROSS : cross PADDR,PWRITE,PSEL,PRDATA.

Assertions:-

We need to write the assertions check and whether our design is behaving as per our specifications or not and also it gives the true ness of our design.

AHB Covergroup :-

```
covergroup ahb_cg;

        option.per_instance=1;

        HADDR : coverpoint ahb.Haddr{bins A0={[32'h8000_0000:32'h8000_03ff]};
```

```
                                        bins B0={[32'h8400_0000:32'h8400_03ff]};
                                    bins C0={[32'h8800_03ff:32'h8800_03ff]}; bins
                                    D0={[32'h8c00_0000:32'h8c00_03ff]};}

        HSIZE : coverpoint ahb.Hsize {bins ZERO_S={2'b00};
                                    bins ONE_S={2'b01};
                                    bins TWO={2'b10};}
        HTRANS : coverpoint ahb.Htrans {bins NONSEQ={2'b10};
                                    bins SEQ={2'b11};}
        HWRITE : coverpoint ahb.Hwrite {bins ZERO_HW={1'b0};
                                    bins ONE_HW={1'b1};}
        HWDATA : coverpoint ahb.Hwdata {option.auto_bin_max=4;}
        HRDATA : coverpoint ahb.Hrdata {option.auto_bin_max=4;}
    AHB_W_CROSS : cross HADDR,HSIZE,HTRANS,HWRITE,HWDATA;
    AHB_R_CROSS : cross HADDR,HSIZE, HTRANS, HWRITE, HRDATA;
endgroup
```

APB Covergroup :-

```
covergroup apb_cg;
        option.per_instance=1;
        PADDR: coverpoint apb.paddr{bins A1={[32'h8000_0000:32'h8000_03ff]};
                                        bins B1={[32'h8400_0000:32'h8400_03ff]};
                                    bins C1={[32'h8800_03ff:32'h8800_03ff]}; bins
                                    D1={[32'h8c00_0000:32'h8c00_03ff]};}
```

```
PWRITE : coverpoint apb.pwrite {bins ZERO_PW={1'b0};
                                bins ONE_PW={1'b1};}
PSEL : coverpoint apb.psel {bins ONE_SEL={4'b0001};
                            bins TWO_SEL={4'b0010};
                            bins FOUR_SEL={4'b0100};
                            bins EIGHT_SEL={4'b1000};}

PWDATA : coverpoint apb.pwdata {option.auto_bin_max=4;}
PRDATA : coverpoint apb.prdata {option.auto_bin_max=4;}
APB_W_CROSS : cross PADDR,PWRITE,PSEL,PWDATA;
APB_R_CROSS : cross PADDR,PWRITE,PSEL,PRDATA;
endgroup
```

Software Tools Used:-

1. GVIM (Text Editor)
2. Model sim (Wave form analysis)
3. VCS,Questa (Simulation Editor Tools)
4. Language (UVM)

Single Transfer Code:

```
class single
    extends ahb_b_seq;
     `uvm_object_utils(single) function new(string
    name="single"); super.new(name); endfunction task
    body(); req=ahb_xtn::type_id::create("req"); begin
```

```
        start_item(req); assert(req.randomize() with
        {Htrans==2'b10;
                                                Hburst==3'b001;
                                                Haddr%1024;});
`uvm_info("single",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)
                                $display(req.lent);
                finish_item(req);
                hwrite=req.Hwrite;
                haddr=req.Haddr; hsize=req.Hsize;
                hburst=req.Hburst;
        /*      start_item(req); assert(req.randomize() with
                {Htrans==2'b00;
                                                Hburst==3'b000;
                                                Hsize==hsize;
                                        Haddr%1024;});
        `uvm_info("single",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)
                finish_item(req);*/
        //      start_item(req);
                for(int i=0;i<5;i++) begin start_item(req);
                assert (req.randomize() with
                {Htrans==2'b11;
                                                Hburst==3'b001;
                                                Hsize==hsize;
```

```
Haddr==((haddr%1024+(req.lent+(3'b001<<Hsize)))<1024);});

`uvm_info("single",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)


                finish_item(req);

                haddr=req.Haddr;

                end

        /*      start_item(req);
                assert(req.randomize() with {Htrans==2'b00;

                                                Hburst==3'b000;

                                                Hsize==hsize;

                                                Haddr%1024;});

`uvm_info("single",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)

                finish_item(req);*/

end endtask endclass
```

Calculation of Boundary Value for INC-4:-

```
//inc4 class inc4seq extends ahb_b_seq;

`uvm_object_utils(inc4seq) function new(string

name="inc4seq"); super.new(name); endfunction task

body(); req=ahb_xtn::type_id::create("req");

start_item(req); assert(req.randomize() with

{Htrans==2'b10;

                                                Hburst==3'b011;});
```

```
//`uvm_info("m_seq_inc4",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)
finish_item(req); hwrite=req.Hwrite; haddr=req.Haddr; hsize=req.Hsize; hburst=req.Hburst;
for(int i=0;i<3;i++) begin start_item(req); assert(req.randomize() with {Htrans==2'b11;
                                                    Hburst==hburst;
                                                    Hwrite==hwrite;
                                                    Hsize==hsize;
                                                    Haddr==haddr+(3'b001<<hsize);});
//`uvm_info("m_seq_inc4",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)
                        finish_item(req);
                        haddr=req.Haddr;
                        end
        endtask
endclass
```

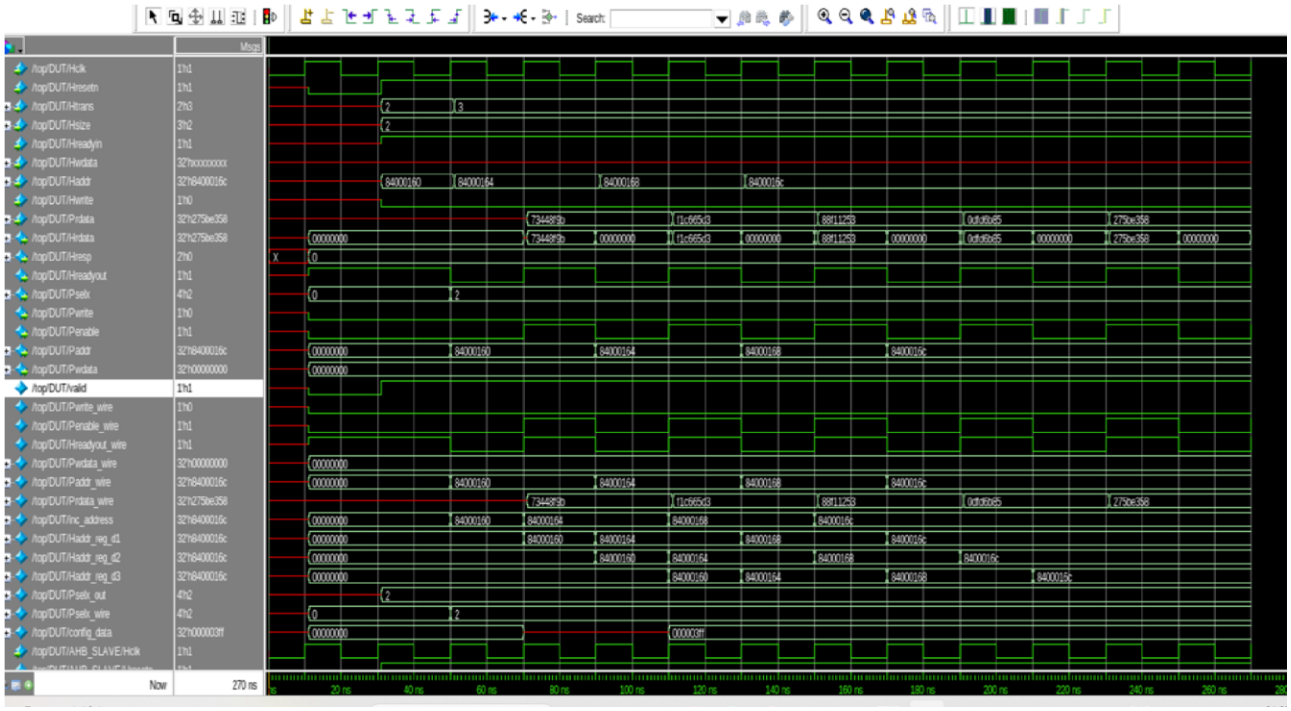
Calculation of WRAP Value for WRAP-4:-

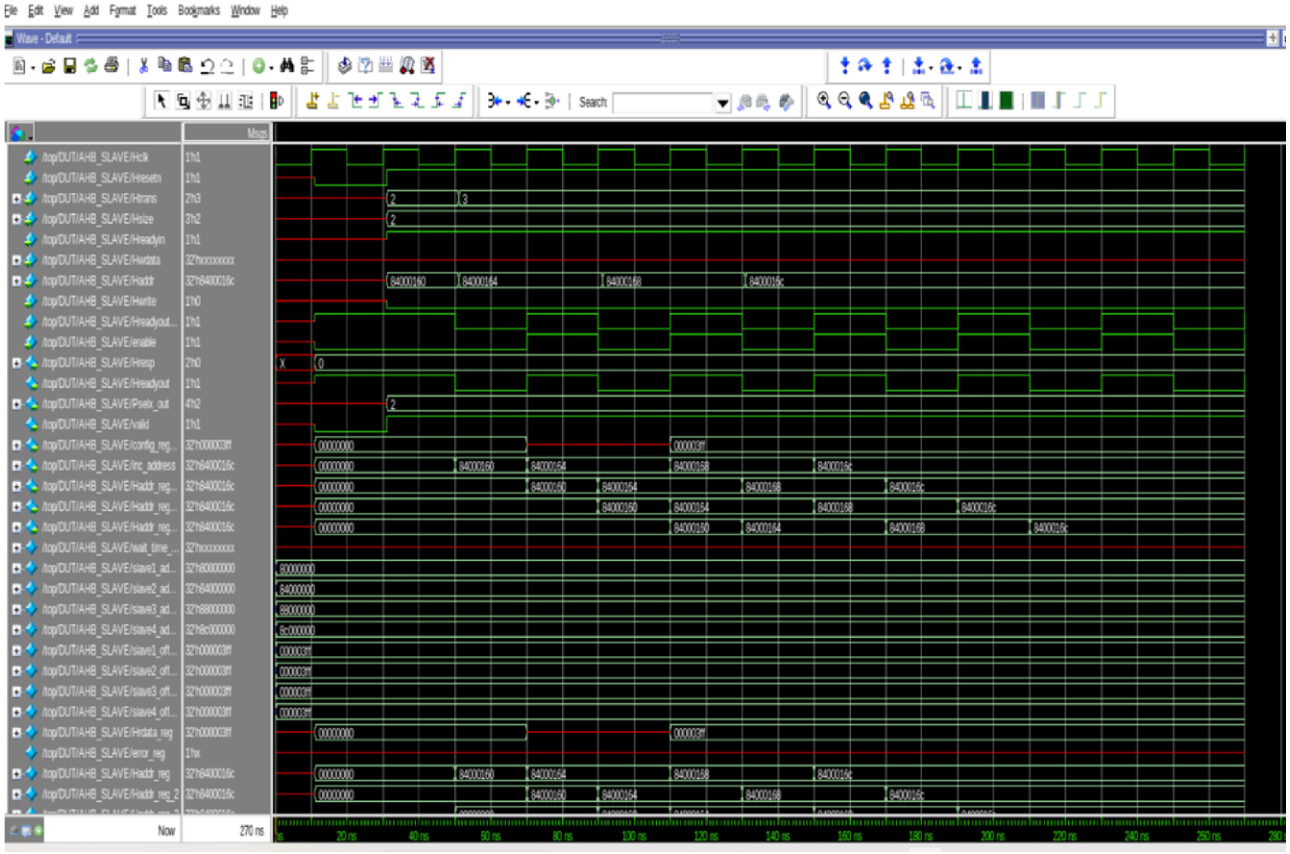
```
//wrap4 class wrp4 extends
ahb_b_seq;
        `uvm_object_utils(wrp4)
        function new(string name="wrp4");
        super.new(name); endfunction task body();
        req=ahb_xtn::type_id::create("req");
        start_item(req); assert(req.randomize() with
        {Htrans==2'b10;
                                        Hburst==3'b010;});
```

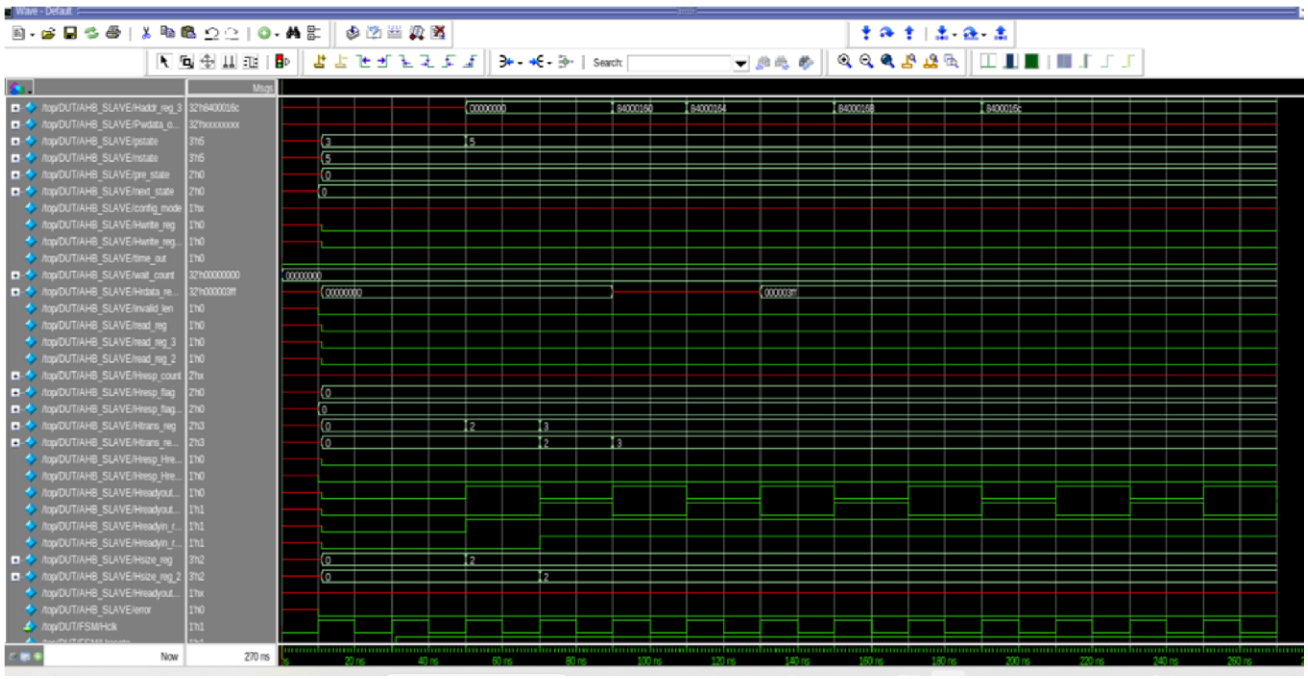
```
//`uvm_info("m_seq_wrp4",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)
                finish_item(req);
                hwrite=req.Hwrite;
                haddr=req.Haddr;
                hsize=req.Hsize;
                hburst=req.Hburst;

if(hsize==0)
                begin for(int i=0;i<3;i++) begin start_item(req);
                assert(req.randomize() with {Htrans==2'b11;
                                                Hburst==hburst;
                                                Hwrite==hwrite;
                                                Hsize==hsize;

Haddr=={haddr[31:2],(haddr[1:0]+1'b1)};});
//`uvm_info("m_seq_wrp4",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)
                        finish_item(req);
                haddr=req.Haddr; end end
if(hsize==1)
                begin for(int i=0;i<3;i++) begin start_item(req);
                assert(req.randomize() with {Htrans==2'b11;
                                                Hburst==hburst;
                                                Hwrite==hwrite;
                                                Hsize==hsize;
```

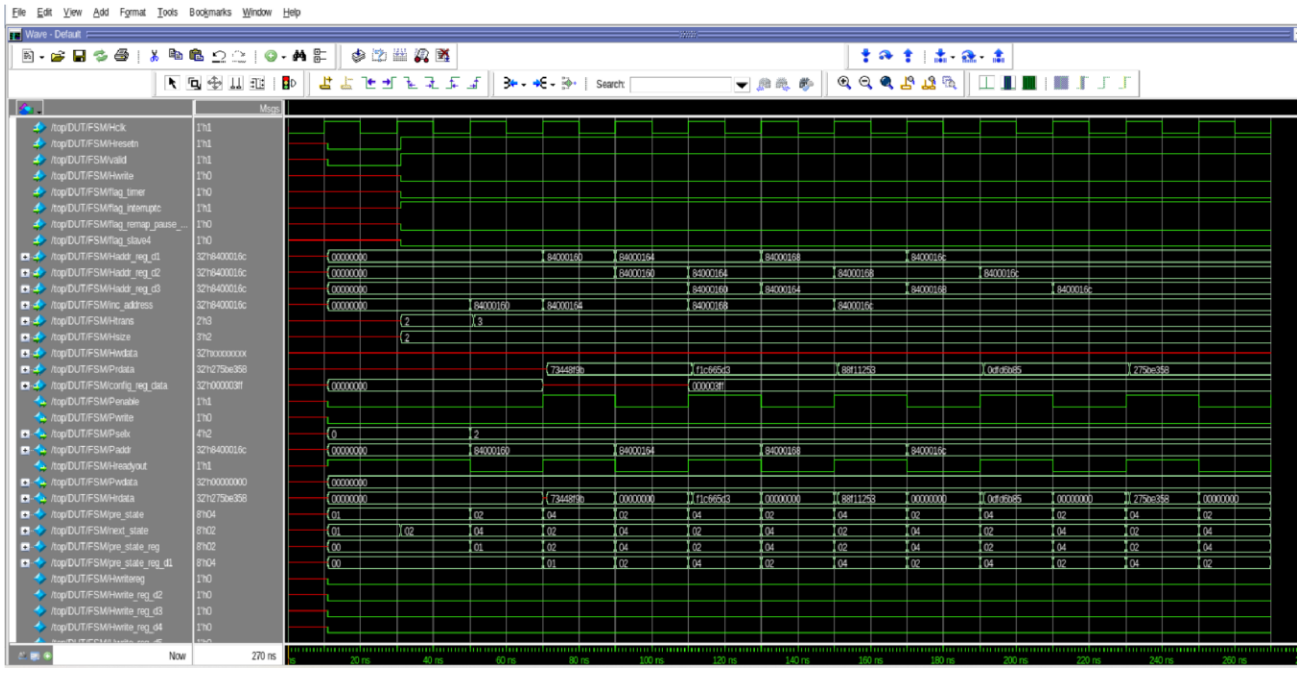
```
Haddr=={haddr[31:3],(haddr[2:1]+1'b1),haddr[0]};});
//`uvm_info("m_seq_wrp4",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)
                                finish_item(req);
                            haddr=req.Haddr; end
                    end
if(hsize==2)
                    begin for(int i=0;i<3;i++) begin start_item(req);
                    assert(req.randomize() with {Htrans==2'b11;
                                                                Hburst==hburst;
                                                                Hwrite==hwrite;
                                                                Hsize==hsize;

Haddr=={haddr[31:4],(haddr[3:2]+1'b1),haddr[1:0]};});
//`uvm_info("m_seq_wrp4",$sformatf("printing from seqc \n %s", req.sprint()),UVM_LOW)
                                finish_item(req);
haddr=req.Haddr; end end endtask endclass
```

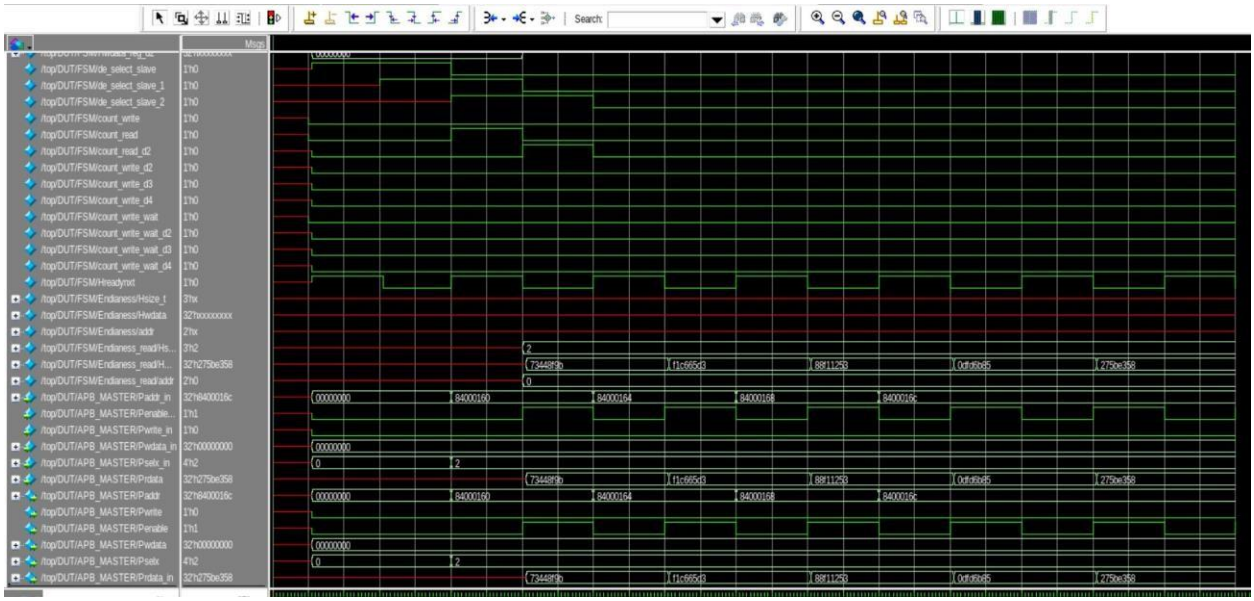
Simulation Final Result:-






Wave - Default
Search:
Msgs
/top/DUT/AHB_SLAVE/Haddr_reg_3
/top/DUT/AHB_SLAVE/Pwdata_o...
/top/DUT/AHB_SLAVE/pstate
/top/DUT/AHB_SLAVE/nstate
/top/DUT/AHB_SLAVE/pre_state
/top/DUT/AHB_SLAVE/next_state
/top/DUT/AHB_SLAVE/config_mode
/top/DUT/AHB_SLAVE/Hwrite_reg
/top/DUT/AHB_SLAVE/Hwrite_reg...
/top/DUT/AHB_SLAVE/time_out
/top/DUT/AHB_SLAVE/wait_count
/top/DUT/AHB_SLAVE/Hrdata_re...
/top/DUT/AHB_SLAVE/invalid_len
/top/DUT/AHB_SLAVE/read_reg
/top/DUT/AHB_SLAVE/read_reg_3
/top/DUT/AHB_SLAVE/read_reg_2
/top/DUT/AHB_SLAVE/Hresp_count
/top/DUT/AHB_SLAVE/Hresp_flag
/top/DUT/AHB_SLAVE/Hresp_flag...
/top/DUT/AHB_SLAVE/Htrans_reg
/top/DUT/AHB_SLAVE/Htrans_re...
/top/DUT/AHB_SLAVE/Hresp_Hre...
/top/DUT/AHB_SLAVE/Hresp_Hre...
/top/DUT/AHB_SLAVE/Hreadyout...
/top/DUT/AHB_SLAVE/Hreadyout...
/top/DUT/AHB_SLAVE/Hreadyin_r...
/top/DUT/AHB_SLAVE/Hreadyin_r...
/top/DUT/AHB_SLAVE/Hsize_reg
/top/DUT/AHB_SLAVE/Hsize_reg_2
/top/DUT/AHB_SLAVE/Hreadyout...
/top/DUT/AHB_SLAVE/error
/top/DUT/FSM/Hclk
Now
270 ns


File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
Search:
Msgs
/top/DUT/FSM/Hclk
/top/DUT/FSM/Hresetn
/top/DUT/FSM/valid
/top/DUT/FSM/Hwrite
/top/DUT/FSM/flag_timer
/top/DUT/FSM/flag_interruptc
/top/DUT/FSM/flag_remap_pause_...
/top/DUT/FSM/flag_slave4
/top/DUT/FSM/Haddr_reg_d1
/top/DUT/FSM/Haddr_reg_d2
/top/DUT/FSM/Haddr_reg_d3
/top/DUT/FSM/inc_address
/top/DUT/FSM/Htrans
/top/DUT/FSM/Hsize
/top/DUT/FSM/Hwdata
/top/DUT/FSM/Prdata
/top/DUT/FSM/config_reg_data
/top/DUT/FSM/Penable
/top/DUT/FSM/Pwrite
/top/DUT/FSM/Pselx
/top/DUT/FSM/Paddr
/top/DUT/FSM/Hreadyout
/top/DUT/FSM/Pwdata
/top/DUT/FSM/Hrdata
/top/DUT/FSM/pre_state
/top/DUT/FSM/next_state
/top/DUT/FSM/pre_state_reg
/top/DUT/FSM/pre_state_reg_d1
/top/DUT/FSM/Hwritereg
/top/DUT/FSM/Hwrite_reg_d2
/top/DUT/FSM/Hwrite_reg_d3
/top/DUT/FSM/Hwrite_reg_d4
Now
270 ns



Difficulties Faced During Project Time:-

The following are the difficulties faced during project time:-

- Debugging.
- Some Times Server will be slow (server will not be support).
- Analyzing the waveforms.
- Some times data mismatch during run time.