k.venu gopal reddy

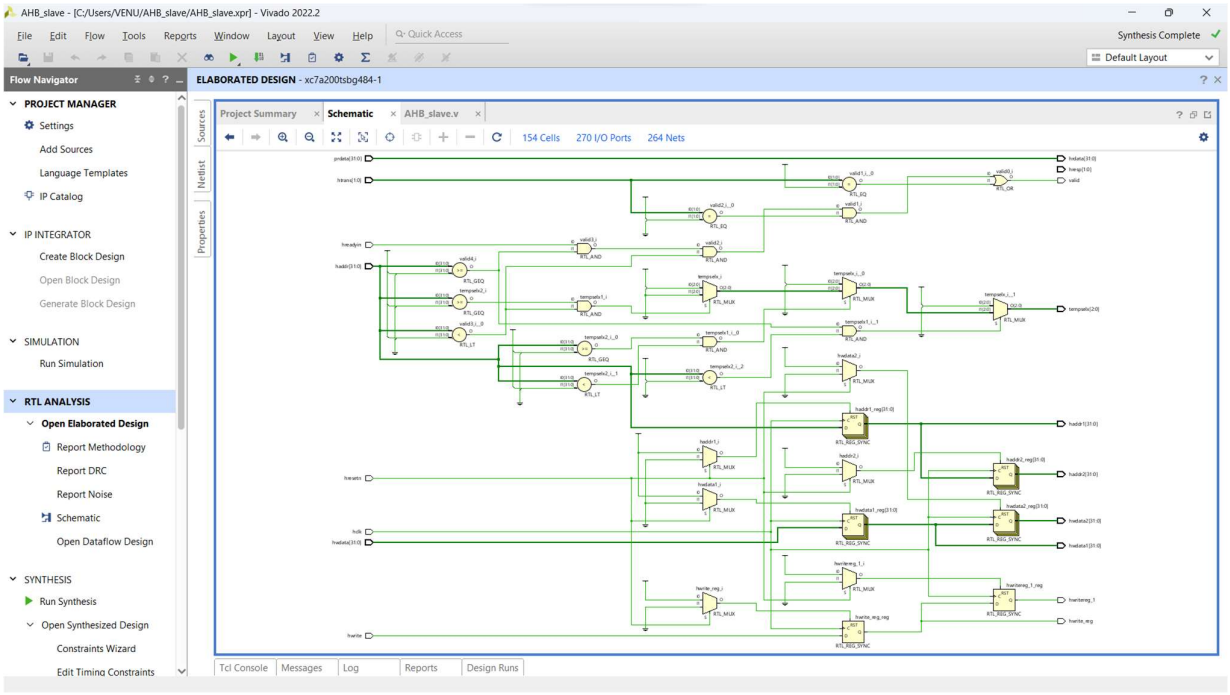
AHB Slave interface:

```
`timescale 1ns / 1ps
module ahb_slave(input hclk,hwrite,hreadyin,hresetn,
input[1:0]htrans,
input[31:0]haddr,hwdata,prdata,
output [31:0]hrdata,
output reg[31:0]haddr1,haddr2,hwdata1,hwdata2,
output reg valid,
output reg hwrite_reg,hwritereg_1,
output reg[2:0]tempselx,
output [1:0]hresp);
//here response signal should be zero bcz we not makin multiple or for ahb interfacing it is ok signal
//implementing pipeline Logic for all AHB intefacing input output ports//address ,data,cntrol signals
always@(posedge hclk)
begin
//active low reset
 if(!hresetn)
  begin
  haddr1<=0;
  haddr2<=0;
  end
else
begin
  haddr1<=haddr;
  haddr2<= haddr1;
  end

 end

always@(posedge hclk)
begin
//active low reset
 if(!hresetn)
  begin
  hwdata1<=0;
  hwdata2<=0;
  end
else
begin
hwdata1<=hwdata;
  hwdata2<=hwdata1;

  end
end
```

```
always@(posedge hclk)
begin
//active low reset
 if(!hresetn)
  begin
   hwrite_reg<=0;
            hwritereg_1<=0;
  end
else
begin
 hwrite_reg<=hwrite;
            hwritereg_1<= hwrite_reg;

  end
end

// implementing valid logic Generation

always@(*)
begin
 valid=1'b0; //initialize
 //htrans for nonseq or seq transaction form type of transfer diagram
 if(hreadyin==1 && haddr>=32'h8000_0000 && haddr<32'h8c00_0000 && htrans==2'b10||htrans==2'b11)
//if hresetn==1 && Hreadyin==1
valid=1;
else
valid=0;
end

//Implementing Tempselectx Logic

always@(*)
begin
//postion of bit tempselx shows which peripheral is connected
tempselx=3'b000;
 if(haddr>=32'h8000_0000&& haddr<32'h8400_0000)
   tempselx=3'b001;
else if(haddr>=32'h8400_0000&& haddr<32'h8800_0000)
   tempselx=3'b010;
else if(haddr>=32'h8800_0000&& haddr<32'h8c00_0000)
   tempselx=3'b100;
end
assign hrdata=prdata;
endmodule
```

k.venu gopal reddy

RTL Schematic :



APB controller :

```
`timescale 1ns / 1ps
module apb_controller_fsm(input hclk,hresetn,hwritereg,hwrite,valid,
input[31:0]haddr,hwdata,hwdata1,hwdata2,haddr1,haddr2,prdata,
input[2:0]tempselx,
output reg penable,pwrite,
output reg hreadyout,
output reg[2:0]psel,
output reg[31:0]paddr,pwdata);
reg penable_temp,pwrite_temp,hr_readyout_temp;
reg[2:0]psel_temp;
reg[31:0]paddr_temp,pwdata_temp;

 parameter ST_IDLE=3'b000,
        ST_READ=3'b001,
        ST_RENABLE=3'b010,
        ST_WWAIT=3'b011,
        ST_WRITE=3'b100,
        ST_WENABLE=3'b101,
        ST_WRITEP=3'b110,
        ST_WENABLEP=3'b111;
  reg[2:0]present,next;
```

```
            //present state logic

            always@(posedge hclk)//present state
            begin
if(!hresetn)
            present<=ST_IDLE;
else
  present<=next;
          end

//next state
always@(*)
begin
next=ST_IDLE;
case(present)
     ST_IDLE:begin
                    if(valid==1 && hwrite==1)
                    next=ST_WWAIT;
                    else if(valid==1 && hwrite==0)
                    next=ST_READ;
                    else next=ST_IDLE;
                    end

ST_WWAIT:begin
                    if(valid==1)
                    next= ST_WRITEP;
           else
                    next= ST_WRITE;
          end

ST_WRITE:
          begin
                    if(valid==1)
                    next= ST_WENABLEP;
           else
                    next= ST_WENABLE;
          end

ST_WRITEP:next= ST_WENABLEP;

 ST_WENABLE:begin
                    if(valid==1 && hwrite==1)
                    next=ST_WWAIT;
                    else if(valid==1 && hwrite==0)
                    next=ST_READ;
                    else if(valid==0)
                    next=ST_IDLE;
                    end
```

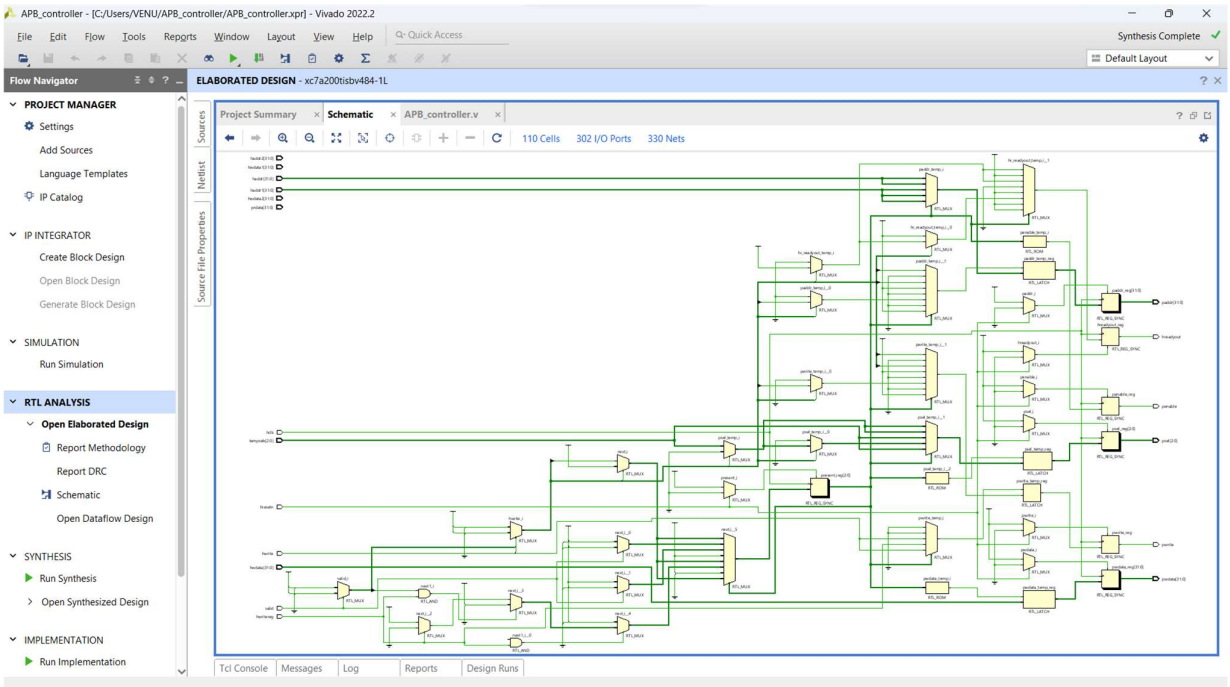
```
 ST_WENABLEP:
                    begin
                    if(valid==1 && hwritereg==1)
                    next=ST_WRITEP;
                    else if(valid==0 && hwritereg==1)
                    next=ST_WRITE;
                    else if(hwritereg==0)
                    next=ST_READ;
                    end

          ST_READ:next=ST_RENABLE;

ST_RENABLE:begin
if(valid==1 && hwrite==1)
next=ST_WWAIT;
else if(valid==1 && hwrite==0)
next=ST_READ;
else if(valid==0)
next=ST_IDLE;
    end
 default:next=ST_IDLE;
                    endcase
end

//temporary output logic
always@(*)
begin
case(present)

ST_IDLE: if(valid==1 && hwrite==0)
        begin
                              paddr_temp=haddr;
                              pwrite_temp=hwrite;
                              psel_temp=tempselx;
                              penable_temp=0;
                              hr_readyout_temp=0;
                              end
                    else if(valid==1 && hwrite==1)
                    begin

                    psel_temp=0;
                    penable_temp=0;
                    hr_readyout_temp=1;
                    end
                    else
begin
```

```
                psel_temp=0;
                penable_temp=0;
                hr_readyout_temp=1;
                end

ST_READ: begin
  penable_temp=1;
        hr_readyout_temp=1;
        end


ST_RENABLE: if(valid==1 && hwrite==0)
begin
 paddr_temp=haddr;
 pwrite_temp=hwrite;
  psel_temp=tempselx;
  penable_temp=0;

                hr_readyout_temp=0;
                end

                else if(valid==1 && hwrite==1)
                begin
                psel_temp=0;
                penable_temp=0;
                hr_readyout_temp=1;
        end

        else
        begin
         psel_temp=0;
         penable_temp=0;
         hr_readyout_temp=1;

         end

ST_WWAIT:begin
 paddr_temp=haddr1;
 pwdata_temp=hwdata;
 pwrite_temp=hwrite;

  psel_temp=tempselx;
  penable_temp=0;
   hr_readyout_temp=0;

        end
ST_WRITE:begin
```

```
 penable_temp=1;
  hr_readyout_temp=1;
          end
ST_WENABLE:
          if(valid==1 && hwrite==0)
        begin

                        psel_temp=0;
                        penable_temp=0;
                        hr_readyout_temp=1;
                        end
                else if(valid==1 && hwrite==0)
                 begin
                        paddr_temp=haddr1;
                        pwrite_temp=hwritereg;
                        psel_temp=tempselx;
                        penable_temp=0;
                         hr_readyout_temp=0;
                        end

                        else
                        begin
                        hr_readyout_temp=1;
                         psel_temp=0;
                                penable_temp=0;
                        end



ST_WRITEP: begin
                                penable_temp=1;
                                hr_readyout_temp=1;
                                end

          ST_WENABLEP: begin
                        paddr_temp=haddr1;
                        pwdata_temp=hwdata;
                        pwrite_temp=hwrite;
                        psel_temp=tempselx;
                        penable_temp=0;
                        hr_readyout_temp=1;

                        end

          endcase
          end

          //actual output logic after registering
```

```
always@(posedge hclk)
begin
if(!hresetn)
begin

 paddr<=0;
 pwdata<=0;
 pwrite<=0;
 psel<=0;
 penable<=0;
 hreadyout<=1;

 end

 else
  begin
  paddr<=paddr_temp;
  pwdata<=pwdata_temp;
  pwrite<=pwrite_temp;
  psel<=psel_temp;
  penable<=penable_temp;
  hreadyout<=hr_readyout_temp;
  end
  end
  endmodule
```

RTL schematic:



Bridge Top :

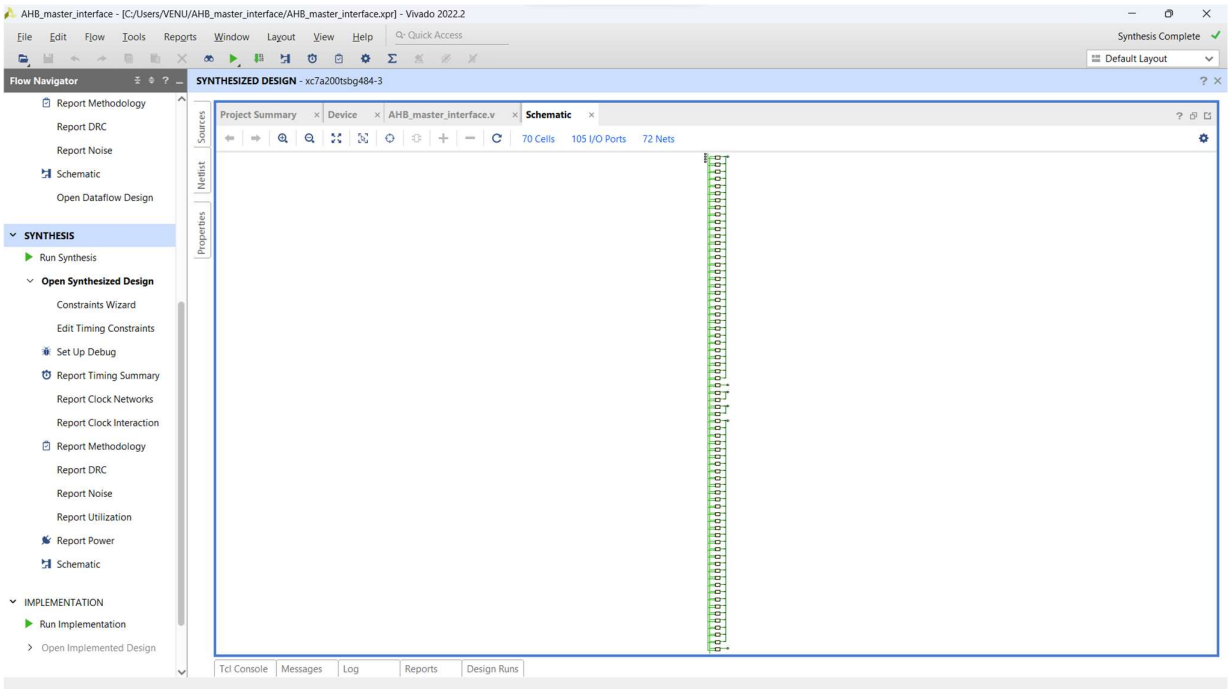
```
module bridge_top(input hclk,hresetn,hwrite,hreadyin,
input[1:0]htrans,
input[31:0]haddr,hwdata,prdata,
output pwrite,penable,hreadyout,
output[2:0]pselx,
output [1:0]hresp,
output[31:0]pwdata,paddr,hrdata);

//intermediate ports

wire [31:0]haddr1,haddr2,hwdata1,hwdata2;
wire [2:0]tempselx;
wire valid,hwritereg;

 //module instantiation
 ahb_slave ahb_s(hclk,hwrite,hreadyin,hresetn,htrans,haddr,hwdata,prdata,
hrdata,haddr1,haddr2,hwdata1,hwdata2,valid,hwritereg,tempselx,hresp);
 apb_controller_fsm
apb_c(hclk,hresetn,hwritereg,hwrite,valid,haddr,hwdata,hwdata1,hwdata2,haddr1,haddr2,prdata,temps
elx,
 penable,pwrite,hreadyout,pselx,paddr,pwdata);
endmodule
```

AHB master Interface block :

```
module ahb_master_interface(input hclk,hresetn,hreadyout,
input[31:0]hrdata,output reg[31:0]haddr,hwdata,
output reg hwrite,hreadyin,
output reg [1:0]htrans,
output [1:0]hresp);
reg[2:0]hburst;//single 4,16,...
reg[2:0]hsize;//size 8,16bit,...

task single_write();
begin
@(posedge hclk);
#1;
begin
hwrite=1;  //for read transaction hwrite =0;
htrans=2'd2;
hsize=0;
hburst=0;
hreadyin=1;
haddr=32'h8000_0001;
end
@(posedge hclk);
```

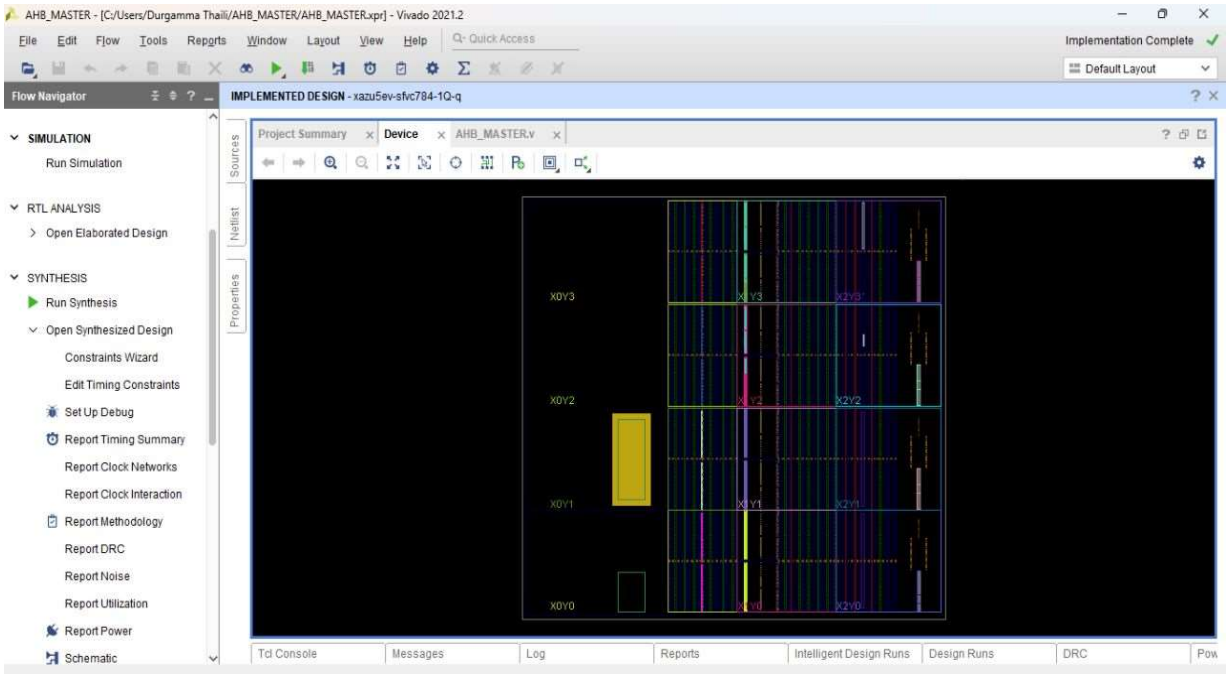
```
#1;
begin
   htrans=2'd0;  //end of single transfer
            hwdata=8'h80; // not required for read transaction
            end
            end
            endtask
task single_read();
begin
@(posedge hclk);
#1;
begin
hwrite=0;  //for read transaction hwrite =0;
htrans=2'd2;
hsize=0;
hburst=0;
hreadyin=1;
haddr=32'h8000_0001;
end
@(posedge hclk);
#1;
begin
   htrans=2'd0;//end of single transfer
            end
            end
            endtask
endmodule
```

RTL schematic:

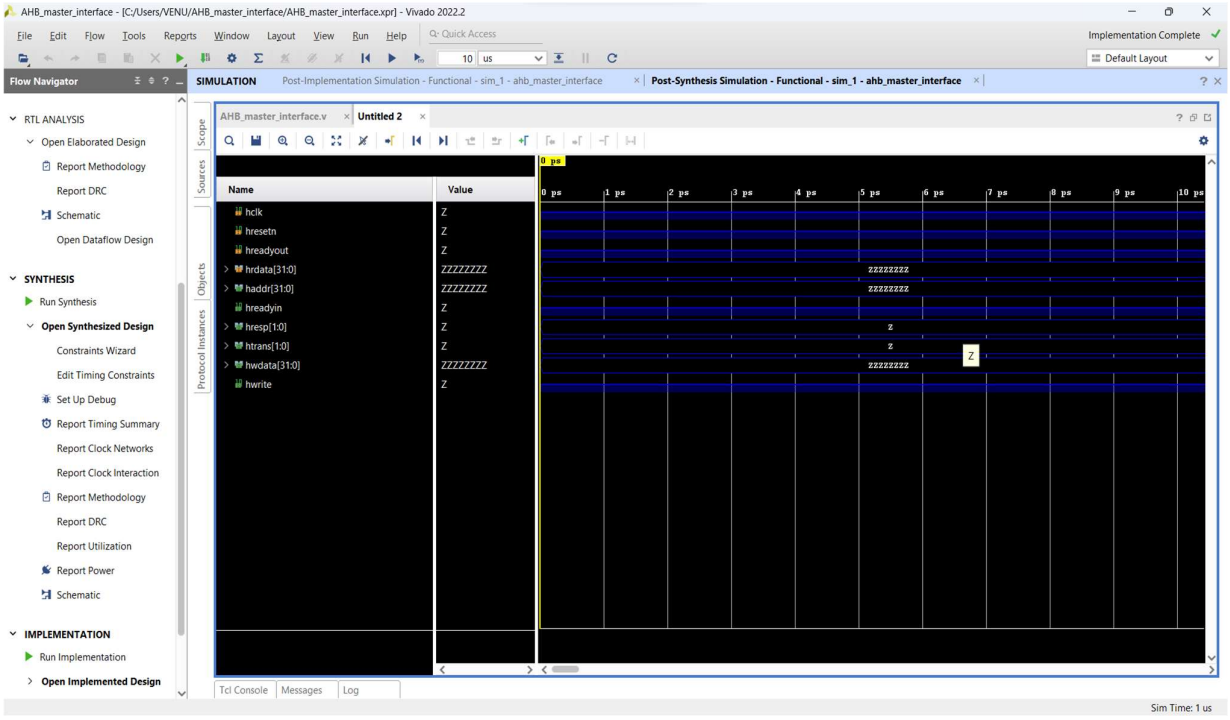


k.venu gopal reddy

IMPLEMENTED DESIGN :



RTL SIMULATION :

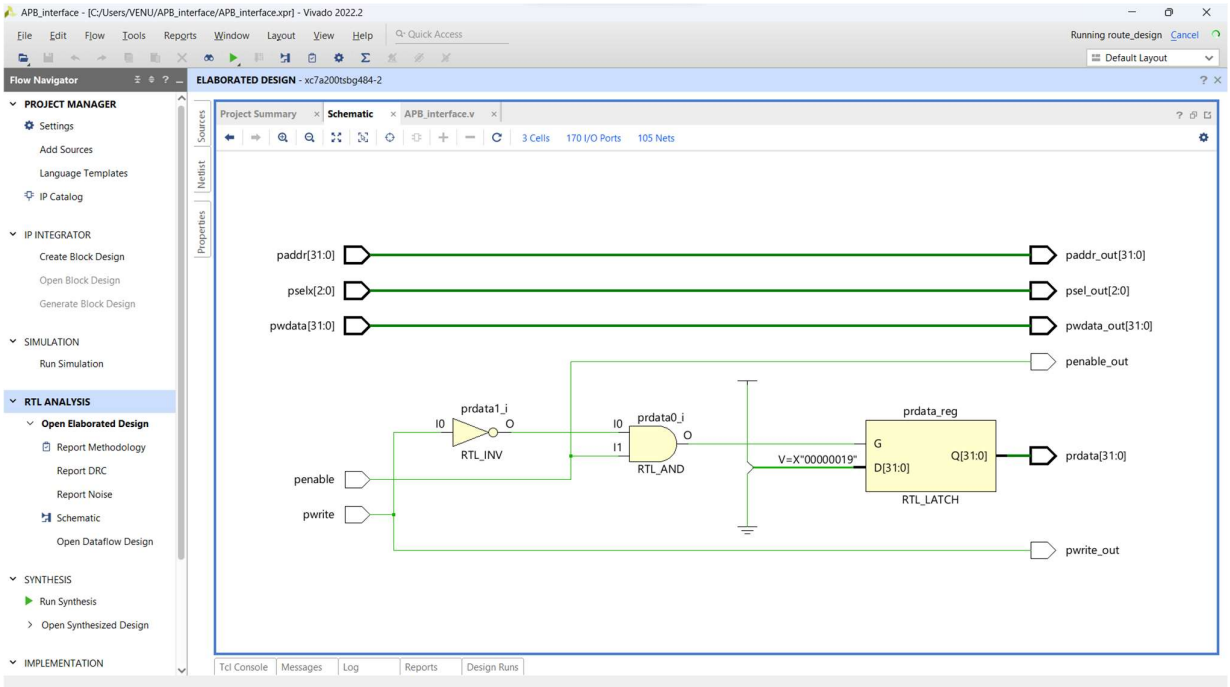


k.venu gopal reddy

APB Interface Block :

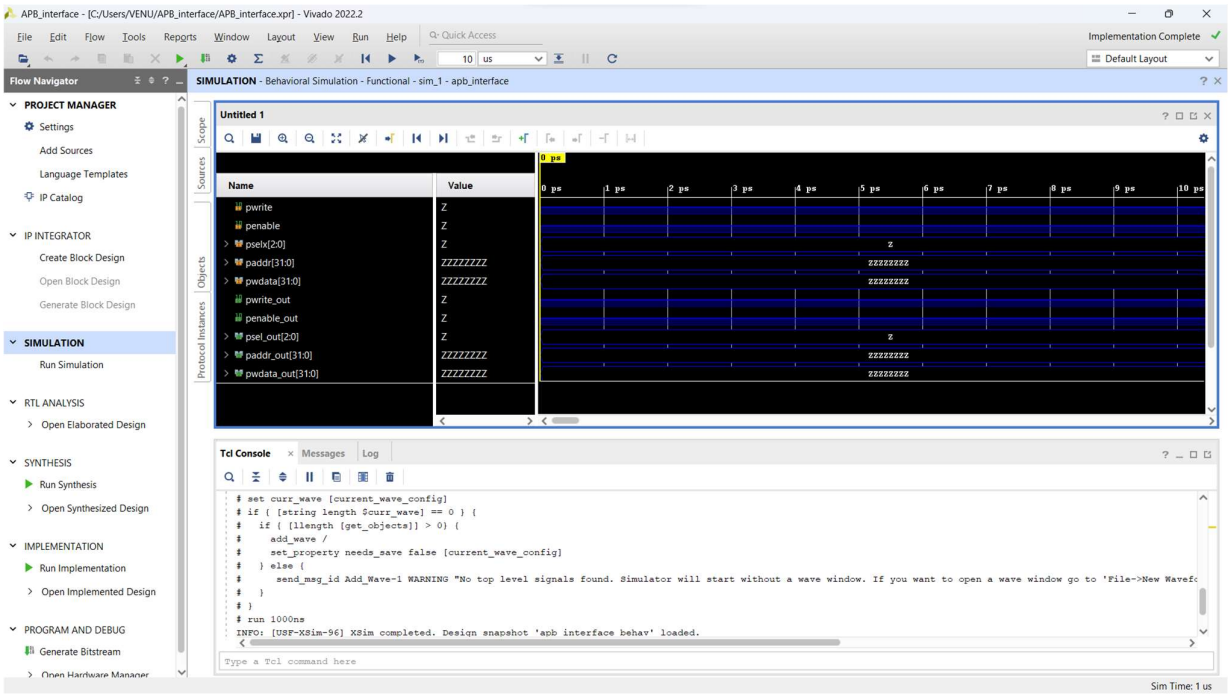
```
module apb_interface(input pwrite,penable,input[2:0]pselx,input[31:0]paddr,pwdata,
output pwrite_out,penable_out,output[2:0]psel_out,
output[31:0]paddr_out,pwdata_out,
output reg[31:0]prdata);

  assign pwrite_out=pwrite;
        assign paddr_out=paddr;
        assign psel_out=pselx;
        assign pwdata_out=pwdata;
        assign penable_out=penable;

always@(*)
begin
if(!pwrite && penable)//pwrite=0 for read transaction
begin
prdata=8'd25;
end
end
endmodule
```

RTL SCHEMATIC DESIGN :



k.venu gopal reddy

RTL SIMULATION :

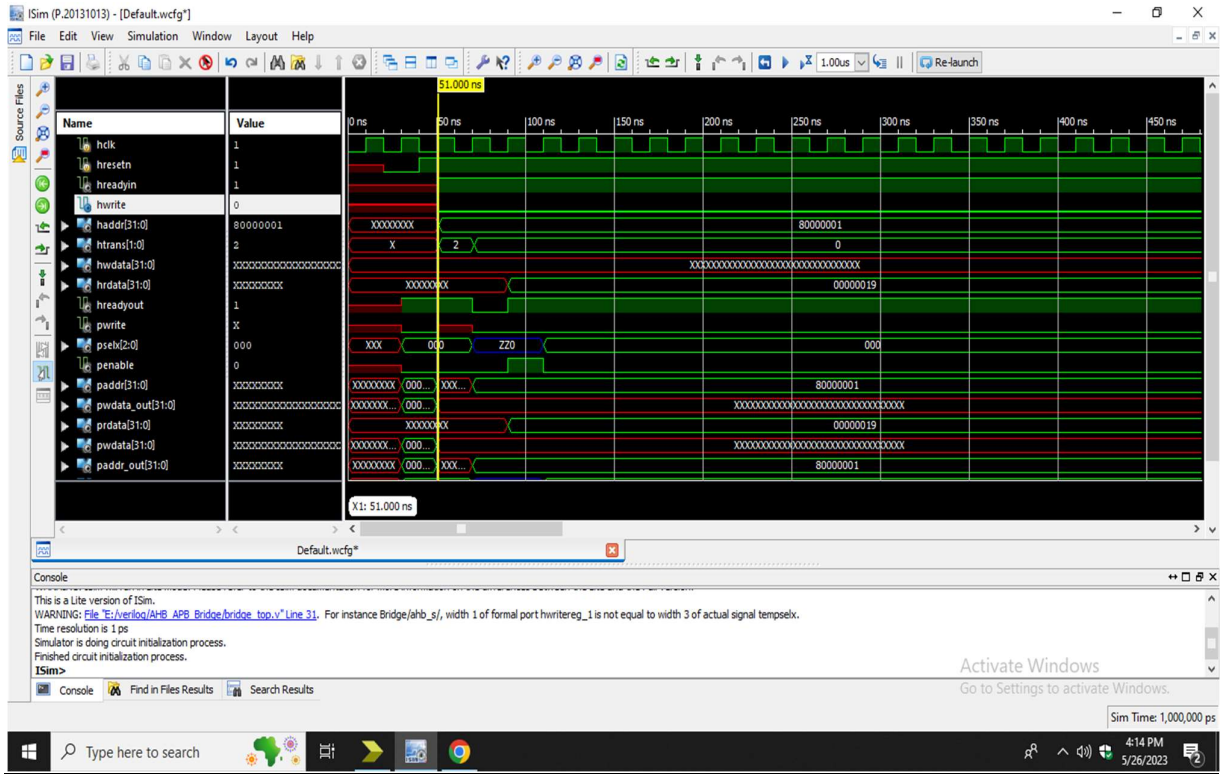


Top Module Verification:

```
module Top();
reg hclk,hresetn;
wire pwrite,penable;
wire[31:0]haddr;
wire hreadyout;
wire[31:0]hrdata;
wire[31:0]hwdata,prdata;
wire hwrite,hreadyin;
wire [1:0]htrans;
wire [1:0]hresp;
wire [2:0]pselx;
wire [31:0]pwdata,paddr;
wire [31:0]paddr_out,pwdata_out;
wire [2:0]psel_out;
wire pwrite_out,penable_out;

ahb_master_interface AHB_MASTER(hclk,hresetn,hreadyout,hrdata,haddr,hwdata,
hwrite,hreadyin,htrans,hresp);
bridge_top Bridge(hclk,hresetn,hwrite,hreadyin,htrans,haddr,hwdata,prdata,
pwrite,penable,hreadyout,pselx,hresp,pwdata,paddr,hrdata);
apb_interface APB_INTERFACE(pwrite,penable,pselx,paddr,pwdata,
pwrite_out,penable_out,psel_out,paddr_out,pwdata_out,prdata);
```

```
initial
begin
hclk=1'b0;
forever#10 hclk=~hclk;
end
        //assigning one by one
task reset;
begin
@(negedge hclk);
hresetn=1'b0;
@(negedge hclk);
hresetn=1'b1;
end
endtask

initial
begin
reset;
AHB_MASTER.single_write();
AHB_MASTER.single_read();
end
endmodule
```

Single Read Transaction Waveform



k.venu gopal reddy

Single Write Transaction Waveform