College of Engineering, Pune
Dept. of Electronics and Telecommunication Engineering

Course : **RTL Simulation and Synthesis Lab (FY VLSI & ES)**
Year : 2020-21

RTL Lab 9,10 Report :

Mis : 122035014
Name : Swapnil Hanmant Phalke

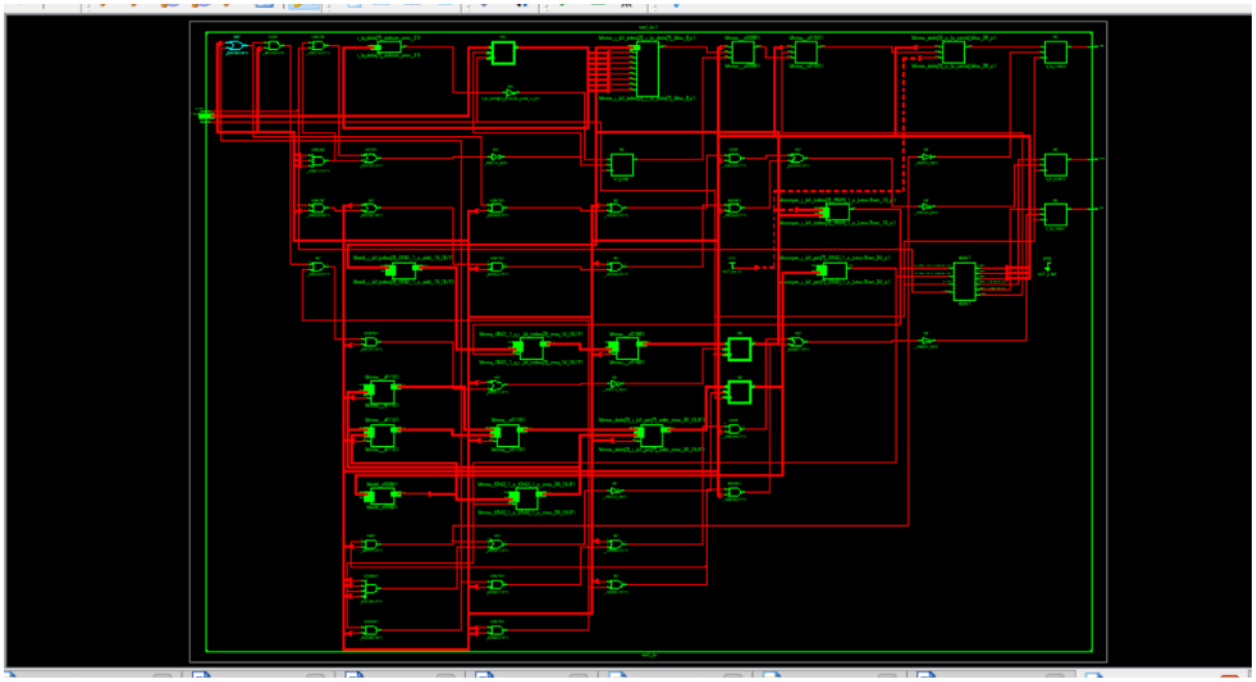
UART_TX

```
module uart_tx(i_clk, i_tx_drive, i_tx_byte, o_tx_active, o_tx_serial, o_tx_done
    );


        input i_clk, i_tx_drive;
  input [7:0]i_tx_byte;
  output o_tx_active, o_tx_done;
  output reg o_tx_serial;


        parameter clk_per_bit = 105;
        parameter data_width = 8;


        parameter s_idle         = 3'b000;
  parameter s_tx_start_bit = 3'b001;
  parameter s_tx_data_bit  = 3'b010;
        parameter s_tx_parity_bit   = 3'b011;
  parameter s_tx_stop_bit  = 3'b100;
  parameter s_cleanup      = 3'b101;


        reg [2:0]state = 0;
  reg [7:0]r_bit_per = 0;
  reg [2:0]r_bit_index = 0;
  reg [7:0]r_tx_data = 0;
  reg r_tx_done = 0;
  reg r_tx_active = 0;

        reg tx_parity;




        always @(posedge i_clk)
   begin

     case (state)
       s_idle :
         begin
```

```
    o_tx_serial <= 1'b1;
    r_tx_done <= 1'b0;
    r_bit_per <= 0;
    r_bit_index <= 0;


    if (i_tx_drive == 1'b1)
      begin
        r_tx_active <= 1'b1;
        r_tx_data <= i_tx_byte;
        state <= s_tx_start_bit;
                                          tx_parity <= ~^r_tx_data;
      end
    else
      state <= s_idle;
  end


              s_tx_start_bit :
  begin
    o_tx_serial <= 1'b0;

    if (r_bit_per < clk_per_bit - 1)
      begin
        r_bit_per <= (r_bit_per + 1) % (2 ** 7);
        state <= s_tx_start_bit;
      end
    else
      begin
        r_bit_per <= 0;
        state <= s_tx_data_bit;
      end
  end



s_tx_data_bit :
  begin
    o_tx_serial <= r_tx_data[r_bit_index];

    if (r_bit_per < clk_per_bit - 1)
      begin
        r_bit_per <= (r_bit_per + 1) % (2 ** 7);
        state <= s_tx_data_bit;
```

```
    end
  else
    begin
      r_bit_per <= 0;


      if (r_bit_index < 7)
        begin
          r_bit_index <= r_bit_index + 1;
          state <= s_tx_data_bit;
        end
      else
        begin
          r_bit_index <= 0;
          state <= s_tx_parity_bit;
        end
    end
end


                    s_tx_parity_bit :
                     begin
                       o_tx_serial <= tx_parity;

                if (r_bit_per < clk_per_bit - 1)
    begin
      r_bit_per <= (r_bit_per + 1) % (2 ** 7);
      state <= s_tx_parity_bit;
    end
  else
                         state <= s_tx_stop_bit;
              end


                    s_tx_stop_bit :
begin
  o_tx_serial <= 1'b1;


  if (r_bit_per < clk_per_bit-1)
    begin
     r_bit_per <= (r_bit_per + 1) % (2 ** 7);
      state <= s_tx_stop_bit;
    end
```

```
          else
            begin
              r_tx_done <= 1'b1;
              r_bit_per <= 0;
              state <= s_cleanup;
              r_tx_active <= 1'b0;
            end
        end



      s_cleanup :
        begin
          r_tx_done <= 1'b1;
          state <= s_idle;
        end


      default :
        state <= s_idle;

    endcase
  end

  assign o_tx_active = r_tx_active;
  assign o_tx_done = r_tx_done;

endmodule
```



```
module uart_tx_tb;

	// Inputs
	reg i_clk;
	reg i_tx_drive;
	reg [7:0] i_tx_byte;

	// Outputs
	wire o_tx_active;
	wire o_tx_serial;
	wire o_tx_done;

	parameter i_clk_per = 87;
	parameter bit_period = 8600;
	reg r_bit_clk;
	reg r_exp_parity;
	//reg tx_parity;


	// Instantiate the Unit Under Test (UUT)
	uart_tx uut (
		.i_clk(i_clk),
		.i_tx_drive(i_tx_drive),
		.i_tx_byte(i_tx_byte),
```

```
        .o_tx_active(o_tx_active),
        .o_tx_serial(o_tx_serial),
        .o_tx_done(o_tx_done)
);
initial begin
i_clk <= 0;
forever #(i_clk_per/2) i_clk <= ~(i_clk);
end

initial
begin
        repeat(3) @(posedge i_clk);
        r_bit_clk <=1;
        forever #(bit_period/2) r_bit_clk <= ~(r_bit_clk);
end

initial begin
        // Initialize Inputs
        @(posedge i_clk);

        i_tx_drive <= 0;
        i_tx_byte <= 0;


        @(posedge i_clk);

        i_tx_drive <= 1;
        i_tx_byte <= 8'h8A;

        @(posedge i_clk);
        r_exp_parity <= ^(i_tx_byte);


        @(posedge i_clk);
        i_tx_drive <= 0;
        @(o_tx_done);
        //if (r_exp_parity == uart_tx.tx_parity)
                //$display("Parity check TEST passed");
        //else
// $display("Parity check TEST passed");
        //Wait 100 ns for global reset to finish


        // Add stimulus here
```

end

endmodule

UART_RX

module uart_rx

(i_clk, i_Rx_Serial, o_Rx_done, o_Rx_DV, o_Rx_Byte
);

parameter CLK_PER_BIT = 105;
parameter DATA_WIDTH = 8;

localparam s_IDLE = 3'b000;
localparam s_RX_START_BIT = 3'b001;
localparam s_RX_DATA_BITS = 3'b010;
localparam s_RX_parity = 3'b011;
localparam s_RX_STOP_BIT = 3'b100;
localparam s_CLEANUP = 3'b101;

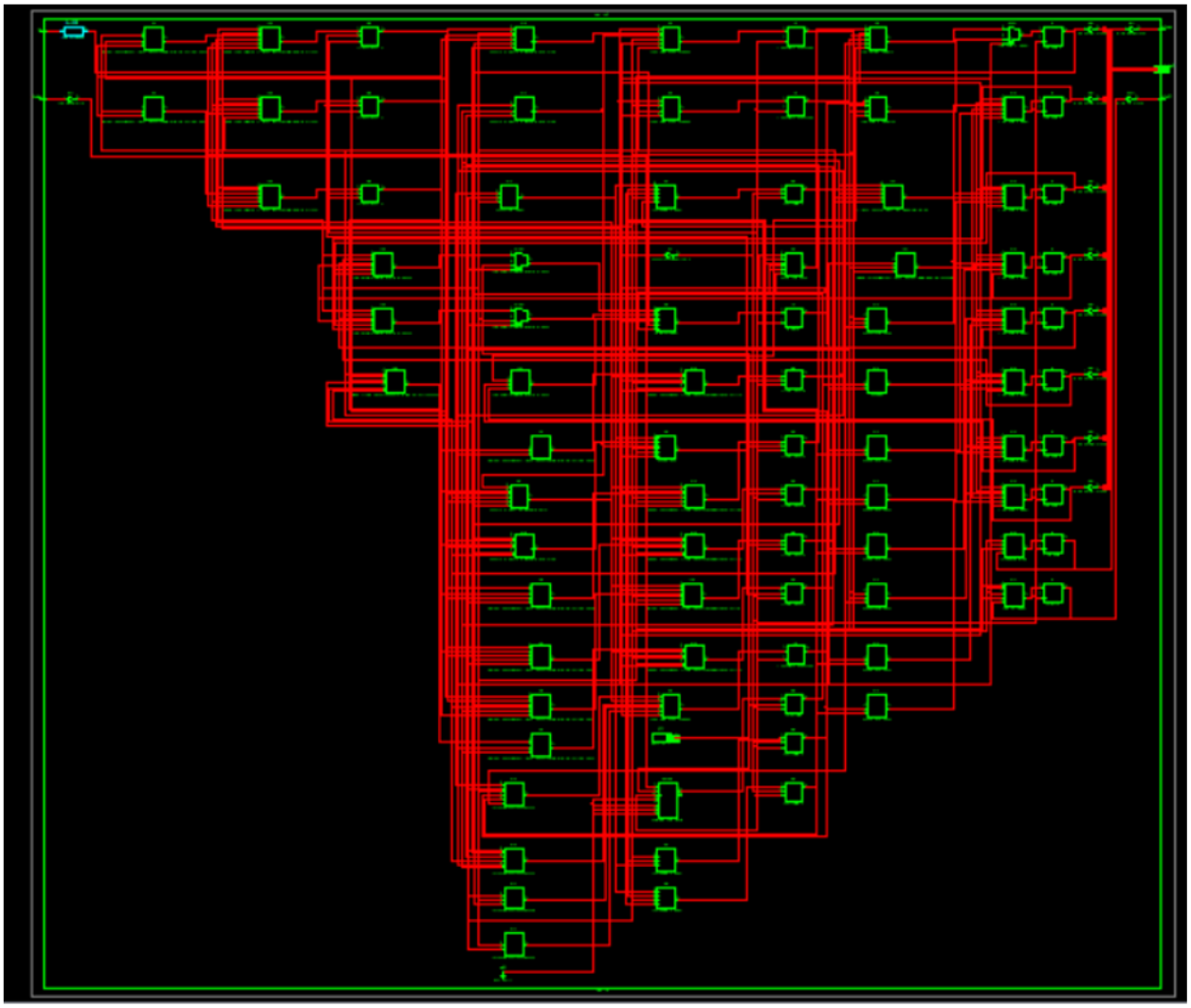
input i_clk;

```
 input        i_Rx_Serial;
         output       o_Rx_done;
 output       o_Rx_DV;
 output [DATA_WIDTH-1:0] o_Rx_Byte;

reg           r_Rx_Data_R = 1'b1;
reg           r_Rx_Data   = 1'b1;

reg [7:0]     r_Clock_Count = 0;
reg [2:0]     r_Bit_Index   = 0;
reg [7:0]     r_Rx_Byte     = 0;
reg           r_Rx_DV       = 0;
reg [2:0]     r_SM_Main     = 0;
reg           rx_parity;
reg           r_Rx_done;


always @(posedge i_clk)
  begin
    r_Rx_Data_R <= i_Rx_Serial;
    r_Rx_Data   <= r_Rx_Data_R;
  end


always @(posedge i_clk)
  begin

    case (r_SM_Main)
      s_IDLE :
        begin
          r_Rx_DV       <= 1'b0;
          r_Clock_Count <= 0;
          r_Bit_Index   <= 0;
                                    r_Rx_done     <= 0;

          if (r_Rx_Data == 1'b0)
            begin
                                      r_SM_Main <= s_RX_START_BIT;
                                      rx_parity <=~^ r_Rx_Byte;
                                      end
          else
            r_SM_Main <= s_IDLE;
        end
```

```
s_RX_START_BIT :
  begin
    if (r_Clock_Count == (CLK_PER_BIT-1)/2)
      begin
        if (r_Rx_Data == 1'b0)
          begin
            r_Clock_Count <= 0;
            r_SM_Main     <= s_RX_DATA_BITS;
          end
        else
          r_SM_Main <= s_IDLE;
      end
    else
      begin
        r_Clock_Count <= r_Clock_Count + 1;
        r_SM_Main     <= s_RX_START_BIT;
      end
  end



s_RX_DATA_BITS :
  begin
    if (r_Clock_Count < CLK_PER_BIT-1)
      begin
        r_Clock_Count <= r_Clock_Count + 1;
        r_SM_Main     <= s_RX_DATA_BITS;
      end
    else
      begin
        r_Clock_Count          <= 0;
        r_Rx_Byte[r_Bit_Index] <= r_Rx_Data;


        if (r_Bit_Index < 7)
          begin
            r_Bit_Index <= r_Bit_Index + 1;
            r_SM_Main   <= s_RX_DATA_BITS;
          end
        else
          begin
            r_Bit_Index <= 0;
            r_SM_Main   <= s_RX_parity;
```

```
        end
      end
  end


            s_RX_parity:
            begin
              r_Rx_Byte <= rx_parity;
             if (r_Clock_Count < CLK_PER_BIT-1)
  begin
    r_Clock_Count <= r_Clock_Count + 1;
    r_SM_Main     <= s_RX_parity;
  end
                        else
                        r_SM_Main       <= s_RX_STOP_BIT;
            end



s_RX_STOP_BIT :
  begin

    if (r_Clock_Count < CLK_PER_BIT-1)
      begin
        r_Clock_Count <= r_Clock_Count + 1;
        r_SM_Main     <= s_RX_STOP_BIT;
      end
    else
      begin
        r_Rx_DV       <= 1'b1;
        r_Clock_Count <= 0;
        r_SM_Main     <= s_CLEANUP;
      end
  end



s_CLEANUP :
  begin
    r_SM_Main <= s_IDLE;
    r_Rx_DV   <= 1'b0;
                                r_Rx_done     <= 1;
  end
```

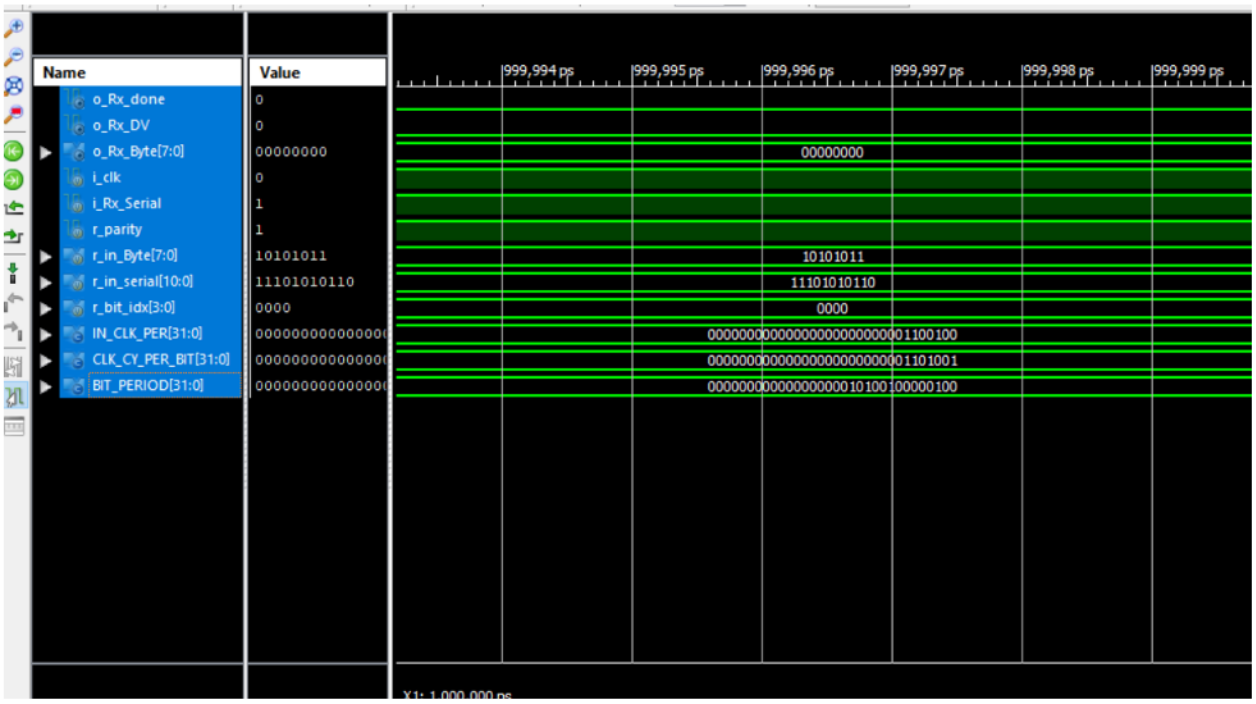
```
        default :
          r_SM_Main <= s_IDLE;

      endcase
    end

  assign o_Rx_DV   = r_Rx_DV;
  assign o_Rx_Byte = r_Rx_Byte;
  assign o_Rx_done = r_Rx_done;

endmodule
```



```
module uart_rx_test;

            // Inputs
```

```
	reg i_clk;
	reg i_Rx_Serial;

	// Outputs
	wire o_Rx_done;
	wire o_Rx_DV;
	wire [7:0] o_Rx_Byte;

	parameter IN_CLK_PER = 100;
	parameter CLK_CY_PER_BIT = 105;
	parameter BIT_PERIOD = 10500;

	reg r_parity;
	reg [7:0] r_in_Byte;
	reg [10:0] r_in_serial;
	reg [3:0] r_bit_idx;


	// Instantiate the Unit Under Test (UUT)
	uart_rx uut (
		.i_clk(i_clk),
		.i_Rx_Serial(i_Rx_Serial),
		.o_Rx_done(o_Rx_done),
		.o_Rx_DV(o_Rx_DV),
		.o_Rx_Byte(o_Rx_Byte)
	);

	initial begin
		// Initialize Inputs
		r_in_Byte = 8'hAB;
		r_bit_idx = 0;
		i_clk = 0;
		forever #(IN_CLK_PER/2) i_clk <= ~(i_clk);
		end


	initial begin
@ (posedge i_clk);
r_parity = ^(r_in_Byte);
@ (negedge i_clk);
r_in_serial = {1'b1, r_parity, r_in_Byte, 1'b0};
repeat (2)
begin
			i_Rx_Serial = 1;
```

```
                @ (negedge i_clk);
        end
    end

    always @ (posedge i_clk)
    begin
        if (o_Rx_done == 1'b1)
        begin
        $display("%t Done",$time);
        end
    end

endmodule
```

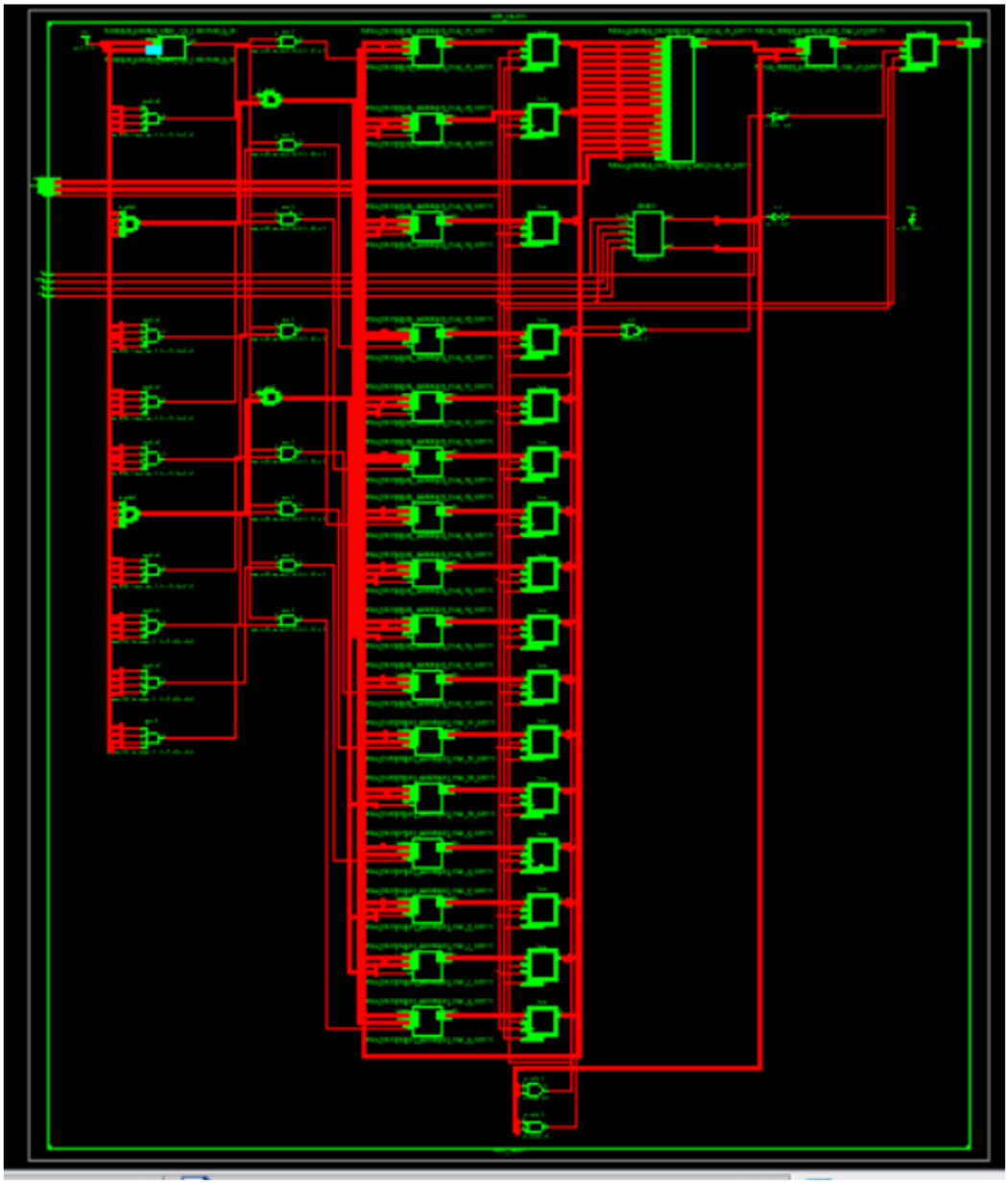


APB

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    18:10:52 05/02/2021
// Design Name:
// Module Name:    apb
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module APB_slave(clk,rst_n,paddr,pwrite,psel,penable,pwdata,prdata);
  parameter  addrWidth = 8;
  parameter  dataWidth = 32;
  input                     clk;
  input                     rst_n;
  input       [addrWidth-1:0] paddr;
  input                     pwrite;
  input                     psel;
  input                     penable;
  input       [dataWidth-1:0] pwdata;
  output reg [dataWidth-1:0] prdata;

reg[dataWidth-1:0] mem [15 : 0];
reg [1:0] state;
parameter IDLE = 2'b00;
parameter SETUP = 2'b01;
parameter W_ENABLE =2'b10;
parameter R_ENABLE = 2'b11;
integer i=0;

always @(negedge rst_n or posedge clk) begin
  if (rst_n == 0) begin
    prdata <= 0;
```

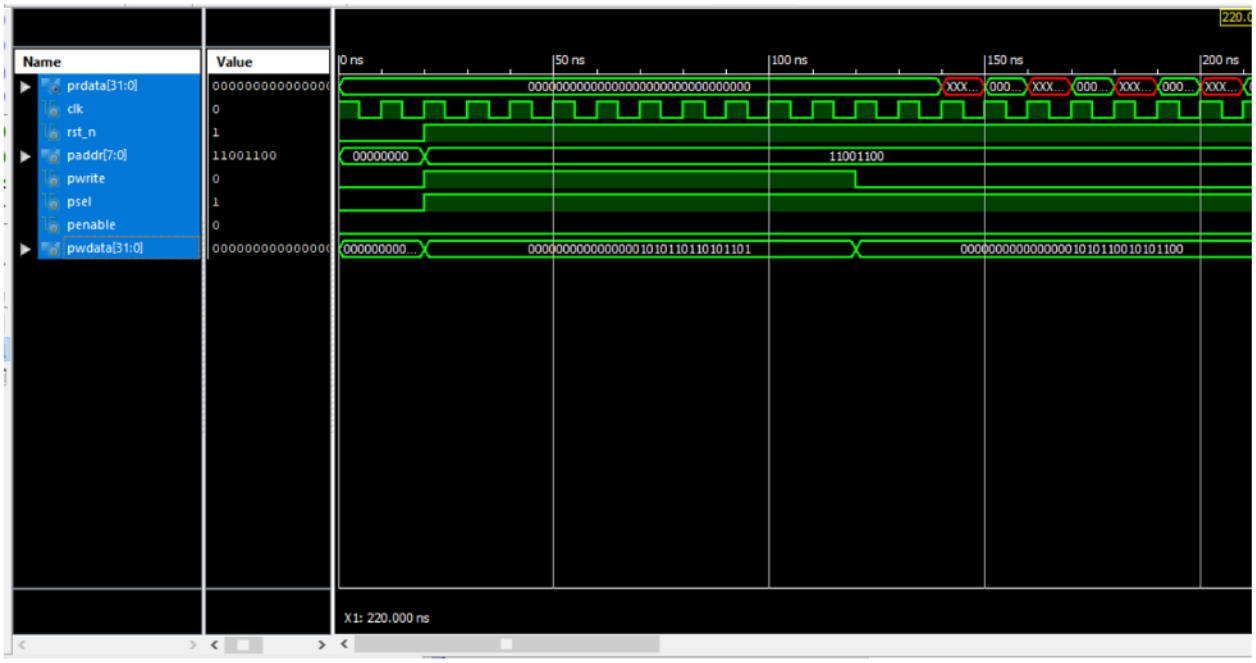
```
        state<= IDLE;
        for (i=0;i<16;i=i+1)
            mem[i]<=0;
end

else begin
  case (state)

          IDLE : begin
                    if(psel)
                                    state<=SETUP;
                                   else
                                    state<=IDLE;
          end

     SETUP : begin

     prdata <= 0;

     if (psel && !penable) begin
       if (pwrite) begin
         state <= W_ENABLE;
       end
       else begin
         state <= R_ENABLE;
       end
     end
   end

   W_ENABLE : begin

       mem[paddr] <= pwdata;

     if(psel && !penable) begin
     state <= SETUP;
     end
      else if(!psel && !penable) begin
                          state<= IDLE;
                          end
                end
   R_ENABLE : begin
              prdata <= mem[paddr];

      if(psel==1 && penable==0) begin
```

```
        state <= SETUP;
        end
         else if(psel==0 && penable==0) begin
                                state<= IDLE;
                                end
                        end
     default : state<= IDLE;
    endcase
  end
end
endmodule
```



```
module test;

	// Inputs
	reg clk;
	reg rst_n;
	reg [7:0] paddr;
	reg pwrite;
	reg psel;
	reg penable;
	reg [31:0] pwdata;

	// Outputs
	wire [31:0] prdata;

	// Instantiate the Unit Under Test (UUT)
	APB_slave uut (
		.clk(clk),
		.rst_n(rst_n),
		.paddr(paddr),
		.pwrite(pwrite),
		.psel(psel),
		.penable(penable),
		.pwdata(pwdata),
		.prdata(prdata)
	);

	initial begin
	clk=1;
	forever clk=#5 ~clk;
	end

	initial begin
		// Initialize Inputs
		rst_n = 0;
		paddr = 0;
		pwrite = 0;
		psel = 0;
		penable = 0;
		pwdata = 0;
		#20;
```

```
                rst_n = 1'b1;
                paddr = 8'hCC;
                pwrite = 1'b1;
                psel = 1'b1;
                penable = 1'b0;
                pwdata = 32'hADAD;
                #100;

                paddr = 8'hCC;
                pwrite = 1'b0;
                psel = 1'b1;
                penable = 1'b0;
                pwdata = 32'hACAC;
                #100;

                $finish;
        end

endmodule
```



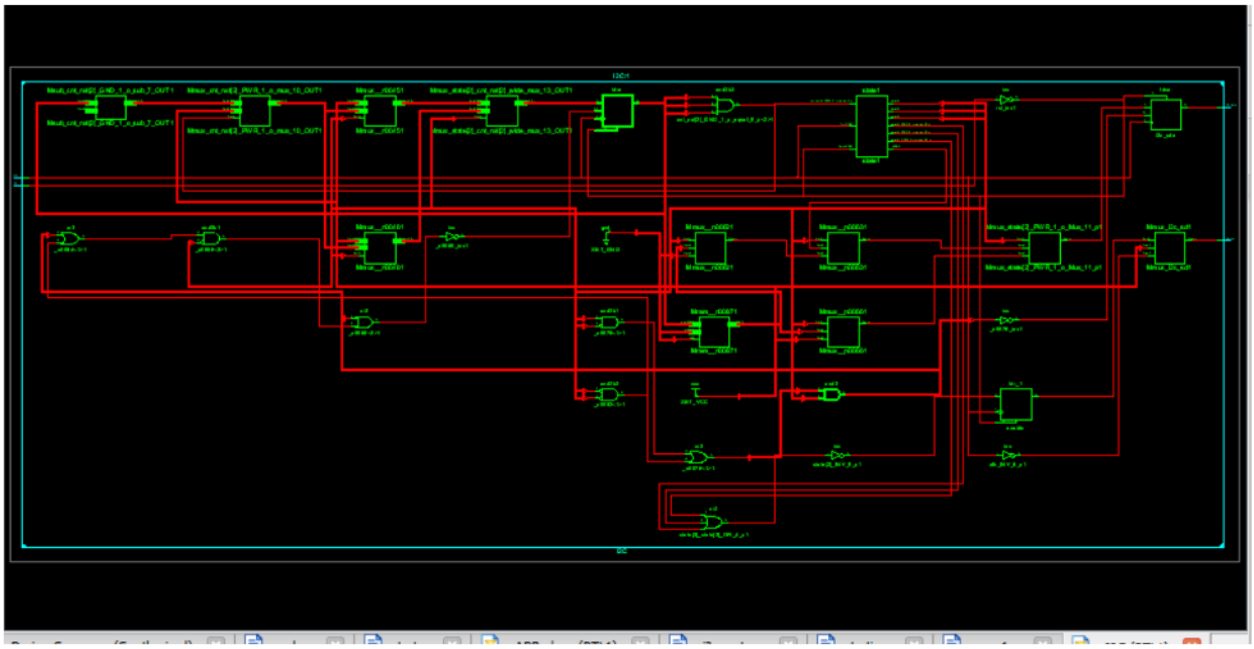
I2C_TX

```
module I2C(clk, rst, i2c_sda, i2c_scl);

input clk,rst;
output reg i2c_sda;
output i2c_scl;
parameter idle=3'b000, start=3'b001, addr=3'b010, rw=3'b011, wack1=3'b100, data=3'b101,
wack2=3'b110, stop=3'b111;
reg [7:0]address;
reg [7:0]data_in;
reg [2:0]cnt_nxt;
reg [2:0]state;
reg enable;
always @(posedge clk)
begin
if (~rst)
begin

i2c_sda<=1'b1;
data_in<=8'b10101010;
state<=idle;
cnt_nxt<=3'b000;
else
 begin
```

```
address<=8'b01010000;
data_in<=8'b10101010;
 case(state)
idle:
 begin
i2c_sda<=1'b1;
 state<=start;
end
start:
 begin
 i2c_sda<=1'b0;
 cnt_nxt<=3'b111;
state<=addr;
end
  addr:
begin

cnt_nxt<=cnt_nxt-3'b001;

i2c_sda<=address[cnt_nxt];
                                                                    if(cnt_nxt==3'b000)
                                                                          begin


state<=rw;

cnt_nxt<=3'b111;
                                                                          end
                                                                    else
                                                                      begin

state<=addr;

                                                                          end
                                                                 end


                                          rw:
                                                                begin
                                                                 i2c_sda<=1'b0;
                                                                 state<=wack1;
                                                                end
```

```
                                wack1:
                                        begin
                                        state<=data;
                                        cnt_nxt<=3'b111;
                                        end

                                data:
                                        begin

                                                cnt_nxt<=cnt_nxt-3'b001;

i2c_sda<=data_in[cnt_nxt];
                                                        if(cnt_nxt==3'b000)
                                                                begin

state<=wack2;

cnt_nxt<=3'b111;
                                                                        end
                                                        else

state<=data;

                                        end

                                wack2:

                                        begin
                                        state<=stop;
                                        end

                                stop:
                                        begin
                                        i2c_sda<=1'b1;
                                        state<=idle;
                                        end
                                default:
                                        begin
                                                state<=idle;
                                                i2c_sda<=1'b1;
end
 endcase
  end
```

```
end

assign i2c_scl=enable?~clk:1'b1;
always@(negedge clk)
begin
if (~rst)
enable<=1'b0;
else if(state==idle || state == start || state == stop)
   enable<=1'b0;
 else
   enable<=1'b1;
end

endmodule
```



```
module new1;

        // Inputs
        reg clk;
        reg rst;

        // Outputs
        wire i2c_sda;
        wire i2c_scl;

        // Instantiate the Unit Under Test (UUT)
```

```
        I2C uut (
                .clk(clk),
                .rst(rst),
                .i2c_sda(i2c_sda),
                .i2c_scl(i2c_scl)
        );

        initial begin
                // Initialize Inputs
                clk = 0;
                rst = 0;

                // Wait 100 ns for global reset to finish
                #100;
    rst=1;
                // Add stimulus here

        end
    always clk=#5~clk;

endmodule
```