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

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

RTL Lab 6,7 Report :

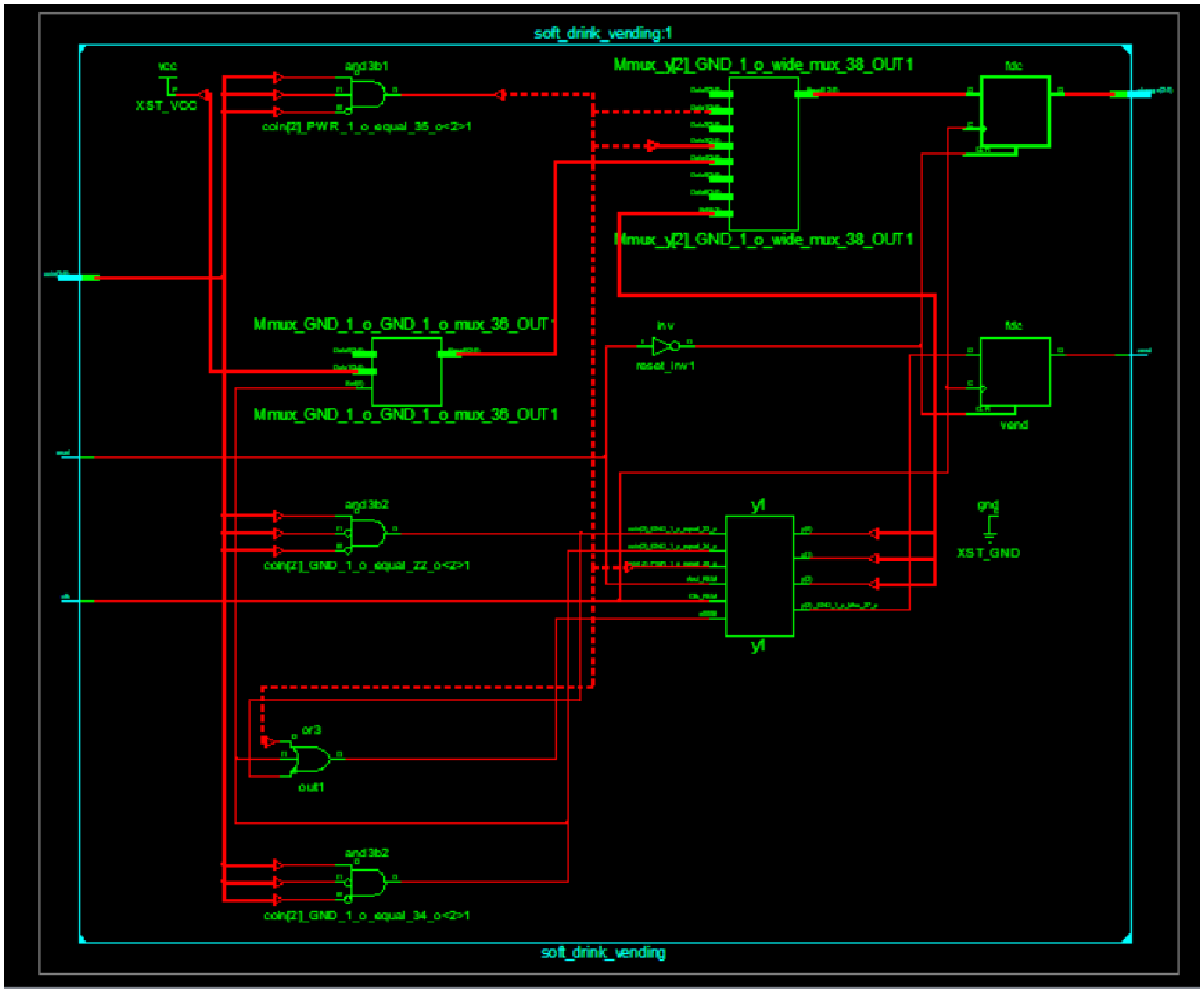
Mis : 122035014
Name : Swapnil Hanmant Phalke

VENDING MACHINE

```
module vending( clk, reset, coin, vend, change
    );
input clk, reset;
input  wire[2:0] coin;

output reg vend;
output reg [2:0] change;

reg [2:0]y,Y;

parameter idle = 3'b000,
                                five = 3'b001,
                                ten = 3'b010,
                                fifteen = 3'b011,
                                twenty = 3'b100,
                                twentyfive = 3'b101;


parameter nickel=3'b001,
          dime=3'b010,
          nickeldime=3'b011,
          dimedime=3'b100,
          quarter=3'b101;

always @ (posedge clk or negedge reset)
begin
          if (!reset)
          y <= idle;
          else
          y <= Y;
end

always @ (y, coin)
begin
          case (y)

          idle: case (coin)
                    nickel:  Y <= five;
                    dime:    Y <= ten;
                    quarter: Y <= twentyfive;
                    default: Y <= idle;
                    endcase
          five: case (coin)
                    nickel:  Y <= ten;
                    dime:    Y <= twenty;
                    quarter: Y <= twentyfive;
                    default: Y <= five;
                    endcase

          ten: case (coin)
                    nickel:  Y <= fifteen;
                    dime:    Y <= twenty;
                    quarter: Y <= twentyfive;
                    default: Y <= ten;
                    endcase


          fifteen: case (coin)
                    nickel:  Y <= twenty;
                    dime:    Y <= twentyfive;
                    quarter: Y <= twentyfive;
                    default: Y <= fifteen;
                    endcase


          twenty: case (coin)
                    nickel:  Y <= twentyfive;
                    dime:    Y <= twentyfive;
                    quarter: Y <= twentyfive;
                    default: Y <= twenty;
                    endcase

          twentyfive: Y <= idle;

          default: Y <= idle;
          endcase
          end


always @ (posedge clk or negedge reset)
begin
          if (!reset) begin
          vend <= 0;
          change <= 3'b000;
```

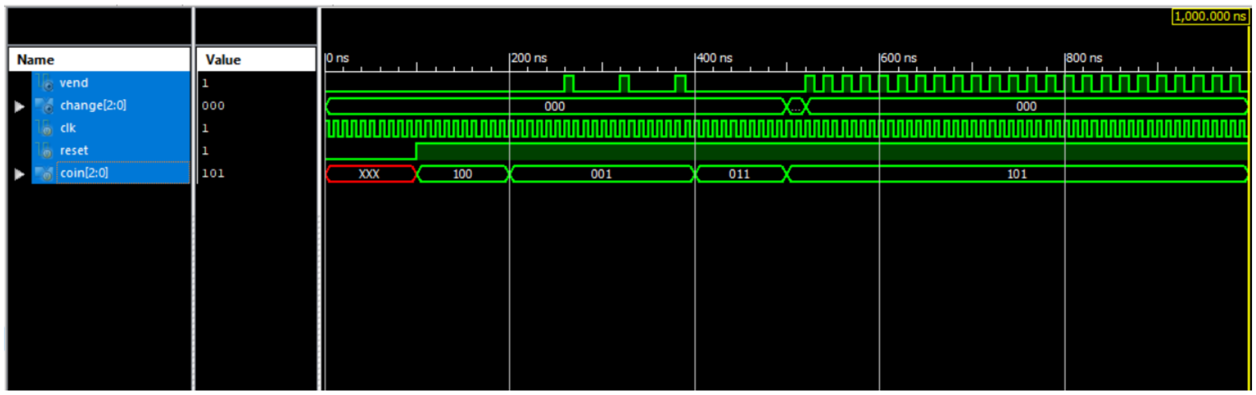
```
end

else case (y)

idle: begin
vend <= 0;
change <= 3'b000;
end

five: begin
vend <= 0;
if (coin == quarter)
change <= nickel;
else
change <= 3'b000;
end

ten: begin
vend <= 0;
if (coin == quarter)
change <= dime;
else
change <= 3'b000;
end

fifteen: begin
vend <= 0;
if (coin == quarter)
change <= nickeldime;
else
change <= 3'b000;
end

twenty: begin
vend <= 0;
if (coin == dime)
change <= nickel;
else if (coin == quarter)
change <= dimedime;
else
change <= 3'b000;
end

twentyfive: begin
```

```
        vend <= 1;
        change <= 3'b000;
        end

        default: begin
        vend <= 0;
        change <= 3'b000;
        end

        endcase
end
endmodule
```


soft_drink_vending:1
vcc
XST_VCC
and3b1
coin[2]_PWR_1_o_equal_35_o<2>1
Mmux_y[2]_GND_1_o_wide_mux_38_OUT1
fdc
Mmux_GND_1_o_GND_1_o_mux_36_OUT
Mmux_GND_1_o_GND_1_o_mux_36_OUT1
inv
reset_inv1
vend
and3b2
coin[2]_GND_1_o_equal_22_o<2>1
yl
gnd
XST_GND
or3
out1
coin[2]_GND_1_o_equal_34_o<2>1
soft_drink_vending

```
module Vending_test;

        // Inputs
```

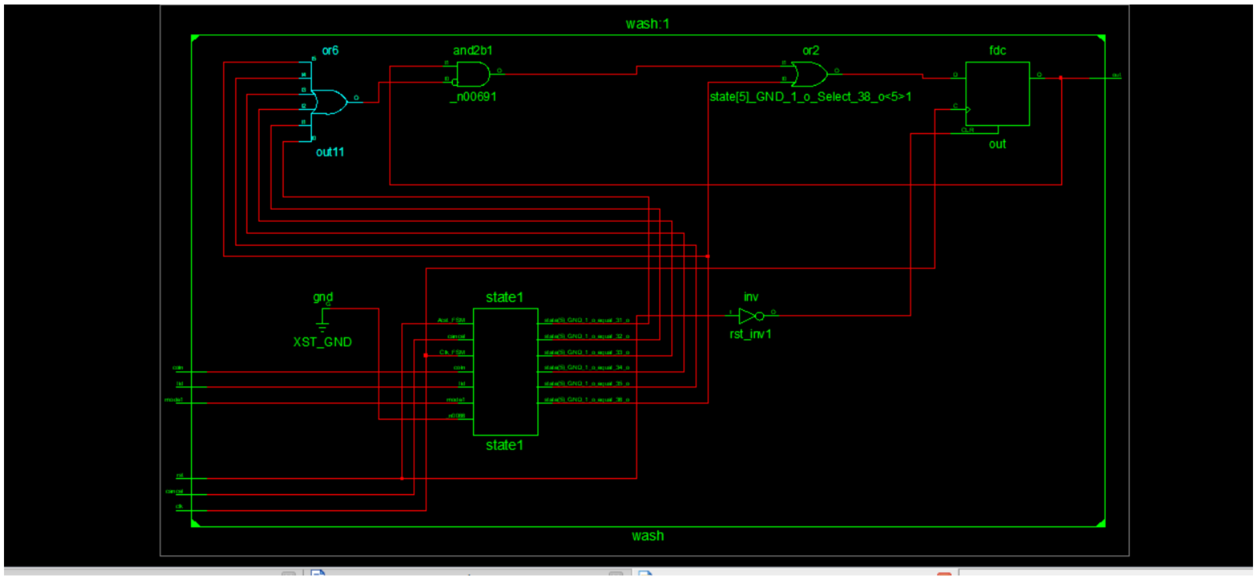
```
	reg clk;
	reg reset;
	reg [2:0] coin;

	// Outputs
	wire vend;
	wire [2:0] change;

	// Instantiate the Unit Under Test (UUT)
	soft_drink_vending uut (
		.clk(clk),
		.reset(reset),
		.coin(coin),
		.vend(vend),
		.change(change)
	);

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

	initial begin
		// Initialize Inputs

		reset = 0;
		#100;
		reset = 1;
		repeat(5)
		begin
		coin = $random;
		#100;
		end

		// Wait 100 ns for global reset to finish
		#100;

		// Add stimulus here

	end

endmodule
```



Washing Machine

```
module wash(out, clk,lid,coin,cancel,mode1, rst);
output out;
input rst;
input clk,lid,coin,cancel,mode1;
reg coin_rtrn,out;
reg[5:0] state;
parameter idle=3'd0, ready=3'd1, soak=3'd2, wash=3'd3,rinse=3'd4,spin=3'd5;
always @(posedge clk or negedge rst)
if(rst==0) begin state=idle; out=0; end
else begin
case (state)
idle: begin out=0; if(cancel==0 && lid==0 && coin==1) state=ready; else state=idle; end
ready: begin out=0; if(cancel==0 && lid==0 && mode1==1) state=soak; else if(cancel==1)
state=idle; else state=ready; end
soak: begin out=0; if(cancel==0 && lid==0 && mode1==1) state=wash; else if(cancel==1)
state=idle; else state=soak; end
wash:  begin out=0; if(cancel==0 && lid==0 && mode1==1) state=rinse; else if(cancel==1)
state=idle; else state=wash; end
rinse:  begin out=0; if(cancel==0 && lid==0 && mode1==1) state=spin; else if(cancel==1)
state=idle; else state=rinse; end
spin: begin out=1; if(cancel==1) state=idle; else state=idle; end
default: state=idle;
endcase
```

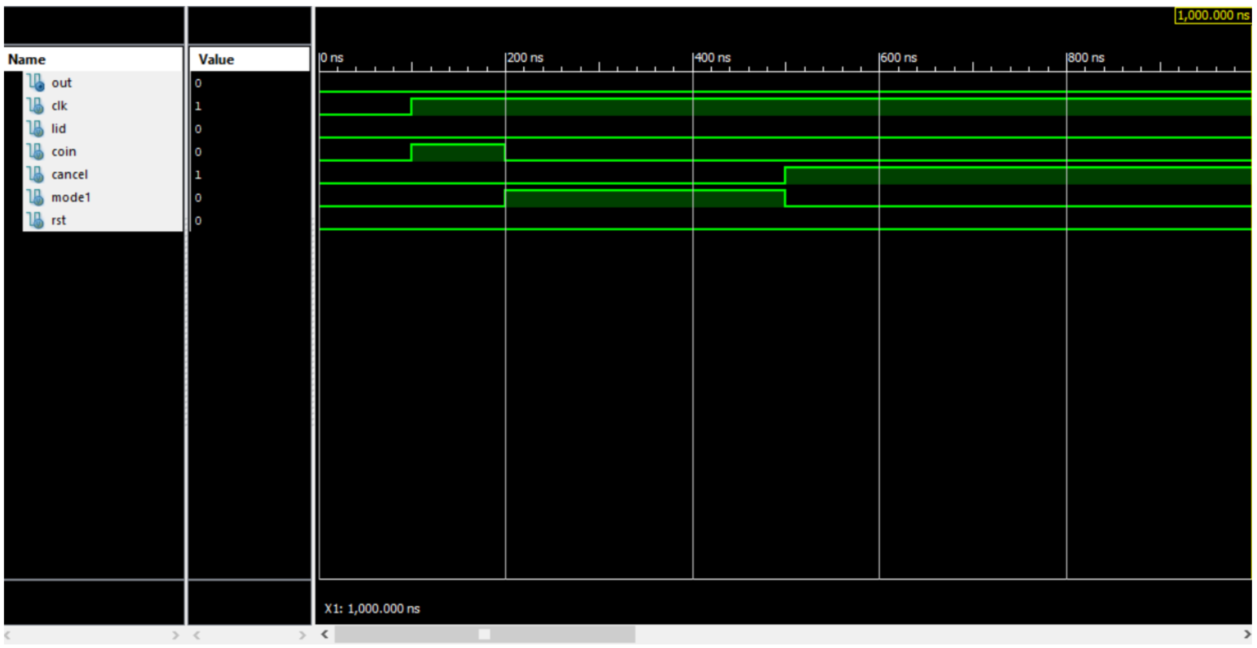
```
end
endmodule
```



```
module washtest;

	// Inputs
	reg clk;
	reg lid;
	reg coin;
	reg cancel;
	reg mode1;
	reg rst;

	// Outputs
	wire out;
```

```
// Instantiate the Unit Under Test (UUT)
wash uut (
	.out(out),
	.clk(clk),
	.lid(lid),
	.coin(coin),
	.cancel(cancel),
	.mode1(mode1),
	.rst(rst)
);

initial begin
	// Initialize Inputs
	clk = 0;
	lid = 0;
	coin = 0;
	cancel = 0;
	mode1 = 0;
	rst = 0;

	// Wait 100 ns for global reset to finish
	#100;



clk = 1;
	lid = 0;
	coin = 1;
	cancel = 0;
	mode1 = 0;
	rst = 0;


	#100;



clk = 1;
	lid = 0;
	coin = 0;
	cancel = 0;
	mode1 = 1;
	rst = 0;
```

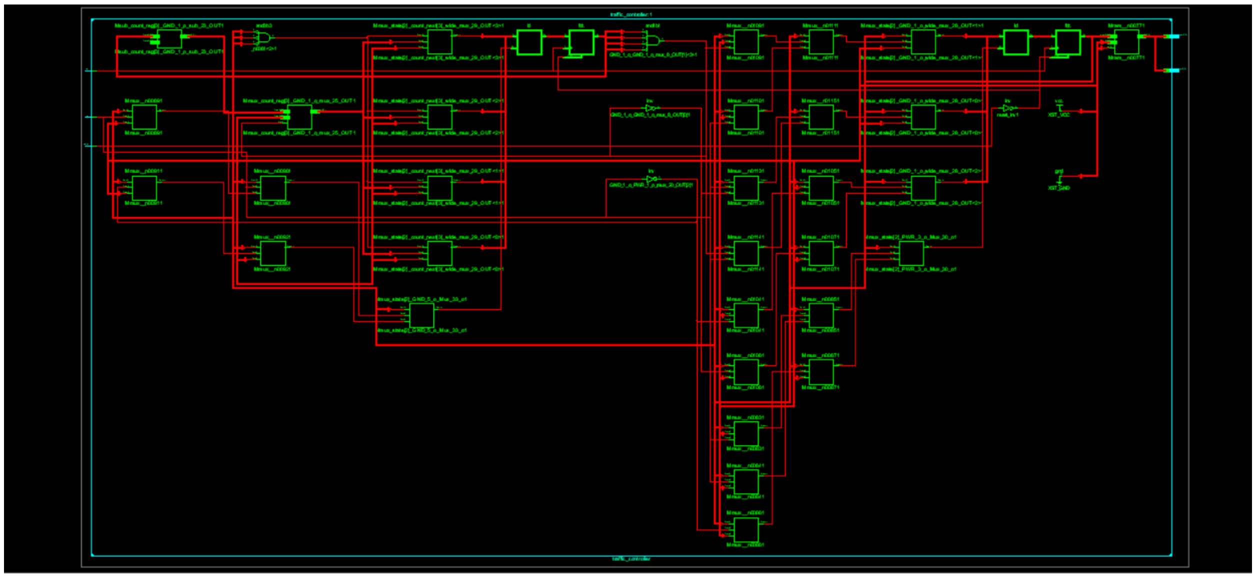
```
                #100;



        clk = 1;
                lid = 0;
                coin = 0;
                cancel = 0;
                mode1 = 1;
                rst = 0;
                // Add stimulus here
                        #100;
                clk = 1;
                lid = 0;
                coin = 0;
                cancel = 0;
                mode1 = 1;
                rst = 0;
                        #100;

                clk = 1;
                lid = 0;
                coin = 0;
                cancel = 1;
                mode1 = 0;
                rst = 0;


        end

endmodule
```



Trafficlightcontroller

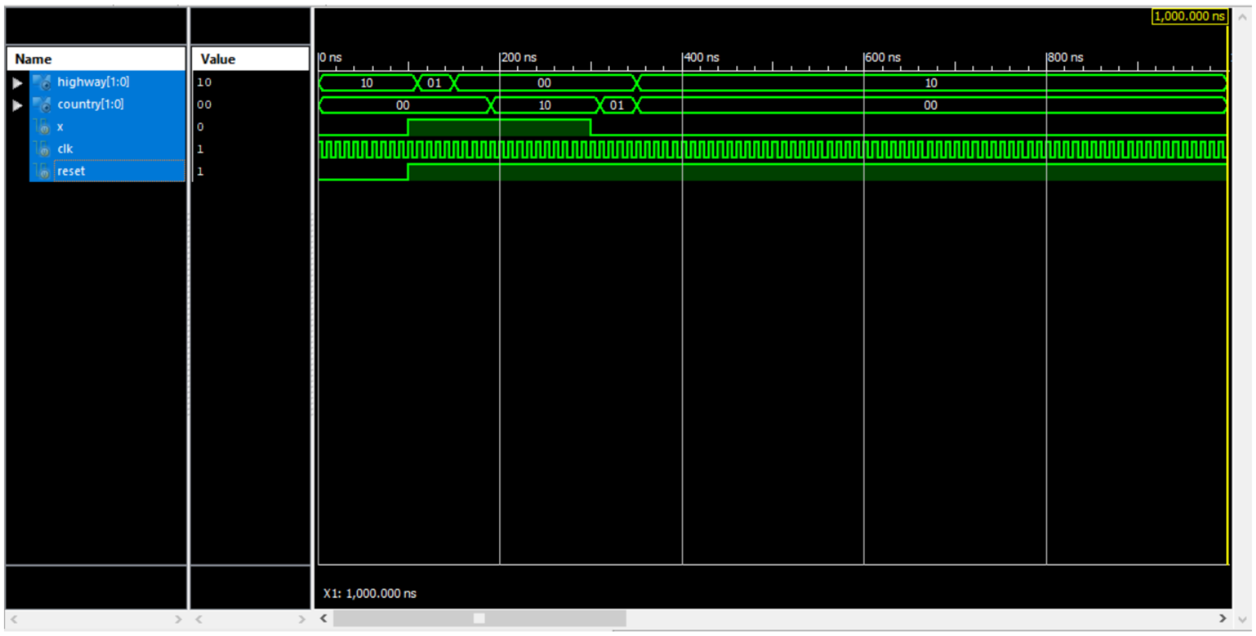
```
module trafficlightcontroller(highway, country, x, clk, reset
    );

input x, clk, reset;
output reg [1:0]highway, country;
reg [2:0] state, next_state;
reg [3:0] count_reg, count_next;
parameter s0 = 3'b000, s1 = 3'b001, s2 = 3'b010, s3 = 3'b011, s4 = 3'b100;
parameter red = 2'b00, yellow = 2'b01, green = 2'b10;

always @ (posedge clk or negedge reset)
begin
 if (!reset)
 begin
 state <= s0;
 count_reg <= 0;
 end
 else
 begin
 state <= next_state;
 count_reg <= count_next;
 end
end

always @ (state, x, count_reg)
begin
 case(state)
 s0 : if (x == 0)
                    next_state <= s0;
      else
                    begin
                    count_next <= 4'b0011;
                    next_state <= s1;
                    end
 s1 : if(x == 1)
                    begin
                    count_next <= count_reg - 1'b1;
                    if (count_reg == 4'b0000)
                    begin
                    next_state <= s2;
                    count_next <= 4'b0011;
                    end
                    else next_state <= s1;
                    end

 s2 :  if (x == 1)
                    begin
                    count_next <= count_reg - 1'b1;
                    if (count_reg == 4'b0000)
                    begin
                    next_state <= s3;
                    count_next <= 4'b0011;
                    end
                    else next_state <= s2;
                    end


 s3 : if (x == 0)
      next_state <= s4;

                    else next_state <= s3;
 s4 :
    if (x == 0)
      begin
      count_next <= count_reg - 1'b1;
                    if (count_reg == 4'b0000)
                    begin
                    next_state <= s0;
                    count_next <= 4'b0011;
                    end
                    else next_state <= s4;
                    end
 default : next_state <= s0;
 endcase
end

always @ (state)
begin
case (state)
s0 : begin highway <= 2'b10;
      country <= 2'b00;
                    end
                    s1 : begin highway <= 2'b01;
      country <= 2'b00;
                    end
                    s2 : begin highway <= 2'b00;
      country <= 2'b00;
                    end
                    s3 : begin highway <= 2'b00;
      country <= 2'b10;
                    end
                    s4 : begin highway <= 2'b00;
      country <= 2'b01;
                    end
                    default : begin highway <= 2'b10;
                    country <= 2'b00;
                    end

endcase
end
```

```
endmodule
```



```
module traffictest;

	// Inputs
	reg x;
	reg clk;
	reg reset;

	// Outputs
	wire [1:0] highway;
	wire [1:0] country;

	// Instantiate the Unit Under Test (UUT)
	traffic_controller uut (
		.highway(highway),
		.country(country),
		.x(x),
		.clk(clk),
		.reset(reset)
	);
	initial begin
	clk = 1;
```

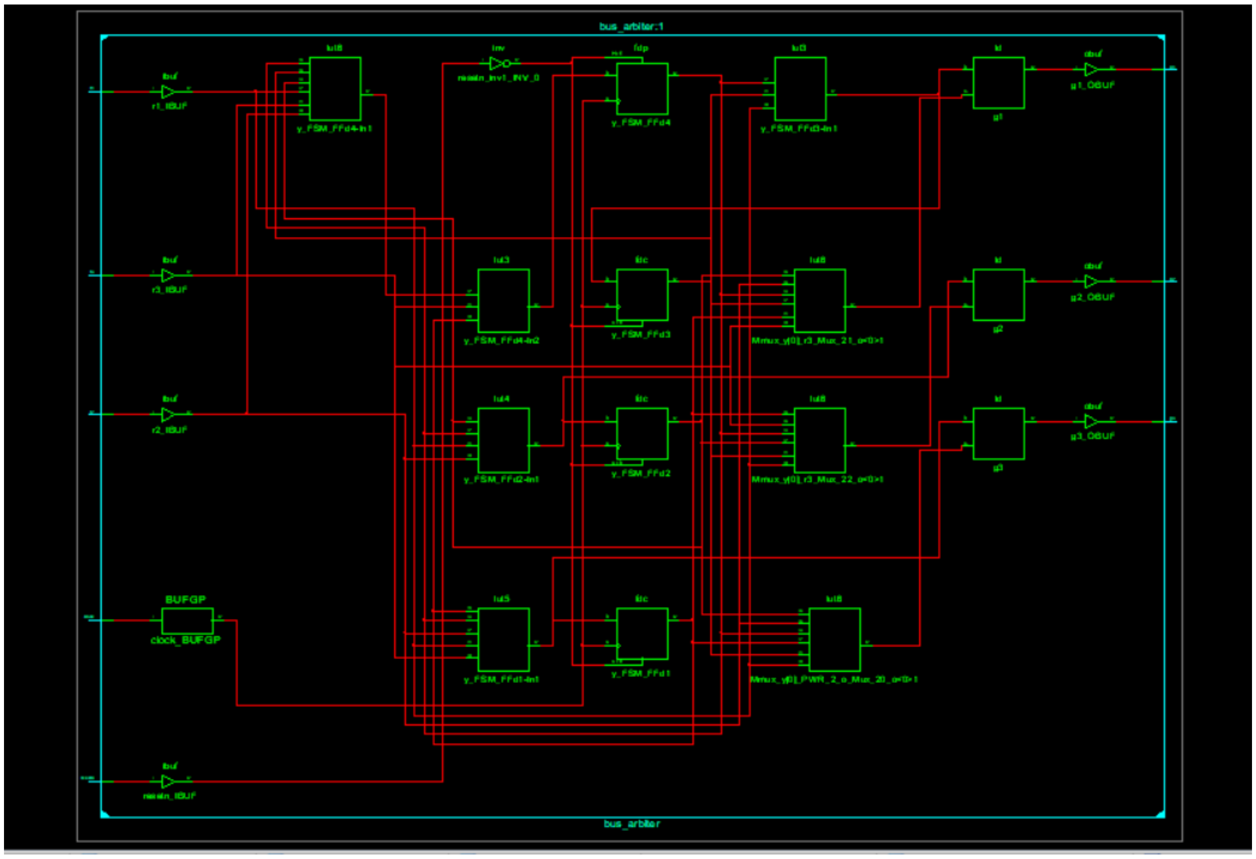
```
        forever #5 clk = ~clk;
        end

        initial begin
                // Initialize Inputs
                x = 0;

                reset = 0;

                // Wait 100 ns for global reset to finish
                #100;

                reset = 1;
                x =1;
                #200;
                x = 0;

                // Add stimulus here

        end

endmodule
```



BUS

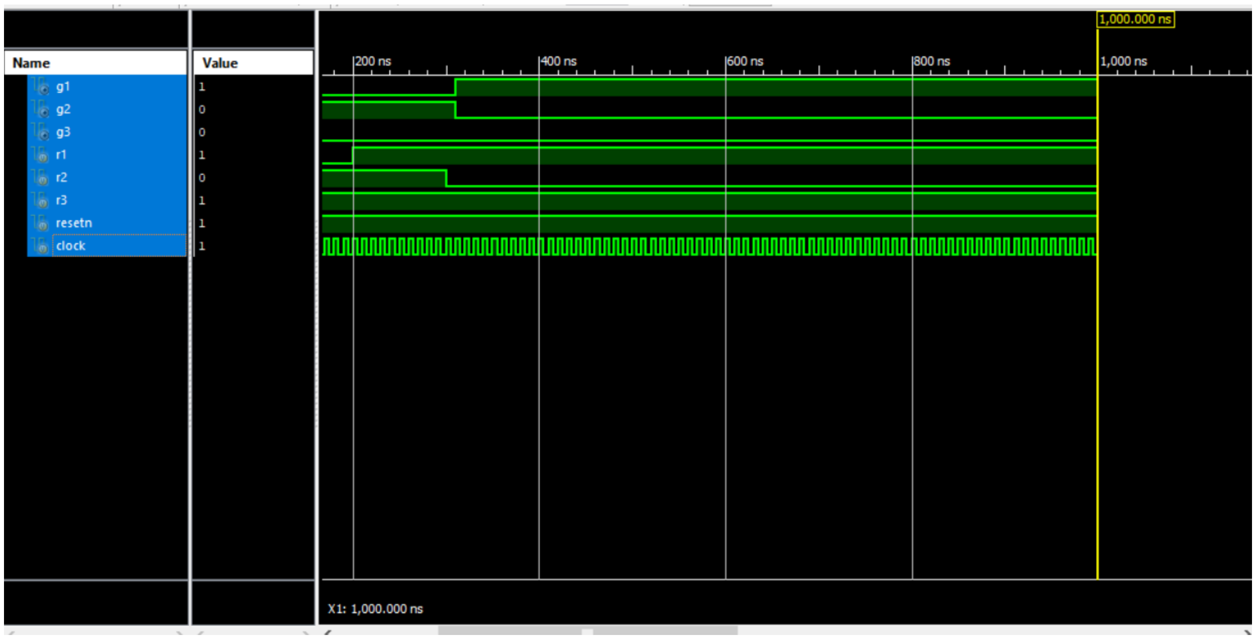
```
module bus(r1,r2,r3,resetn,clock,g1,g2,g3
   );

        input r1,r2,r3;
        input resetn,clock;
        output reg g1,g2,g3;

        reg[0:1]y,Y;
        parameter Idle=2'b00,gnt1=2'b01,gnt2=2'b10,gnt3=2'b11;

always@(posedge clock or negedge  resetn)
begin
if(!resetn)
y<=Idle;
else
y<=Y;
end


always@(y,r1,r2,r3)
begin
case(y)
Idle :casex({r1,r2,r3})
                3'b000: Y <= Idle;
                3'b1xx: Y <= gnt1;
                3'b01x: Y <= gnt2;
                3'b001: Y <= gnt3;
                default: Y <= Idle;
                endcase

gnt1:if(r1)Y <= gnt1;
                else    Y <= Idle;

gnt2:if(r2)Y <= gnt2;
                else    Y <= Idle;

gnt3:if(r3)Y <= gnt3;
                else    Y <= Idle;
default : Y <= Idle;
endcase
end
always@(y)
```

```
begin case(y)
Idle:casex({r1,r2,r3})
                    3'b000: begin
                    g1<=0;
                    g2<=0;
                    g3<=0;
                    end
                    3'b1xx: begin g1 <= 1; g2 <= 0; g3 <= 0; end
                    3'b01x: begin g1 <= 0; g2 <= 1; g3 <= 0; end
                    3'b001: begin g1 <= 0; g2 <= 0; g3 <= 1; end
                    default:begin
                    g1<=0;
                    g2<=0;
                    g3<=0;
                    end
                    endcase
gnt1: if(r1)g1 <= 1;
                    else     begin
                    g1<=0;
                    g2<=0;
                    g3<=0;
                    end
gnt2: if(r2)g2 <= 1;
                    else     begin
                    g1<=0;
                    g2<=0;
                    g3<=0;
                    end
gnt3: if(r3)g3 <= 1;
                    else      begin
                    g1<=0;
                    g2<=0;
                    g3<=0;
                    end
default: begin
     g1<=0;
                    g2<=0;
                    g3<=0;
                    end
endcase
end

endmodule
```



```
// Inputs
reg r1;
reg r2;
reg r3;
reg resetn;
reg clock;

// Outputs
wire g1;
wire g2;
wire g3;

// Instantiate the Unit Under Test (UUT)
bus_arbiter uut (
        .r1(r1),
        .r2(r2),
        .r3(r3),
        .resetn(resetn),
        .clock(clock),
        .g1(g1),
        .g2(g2),
        .g3(g3)
```

```
    );
    initial begin
    clock = 1;
    forever #5 clock = ~clock;
    end

    initial begin
        // Initialize Inputs

        resetn = 0;
        #100;
        resetn = 1;
        repeat(5)
        begin
         r1 = $random;
        r2 = $random;
        r3 = $random;
        #100;
        end


        // Wait 100 ns for global reset to finish


        // Add stimulus here

    end

endmodule
```



Name	Value
g1	1
g2	0
g3	0
r1	1
r2	0
r3	1
resetn	1
clock	1