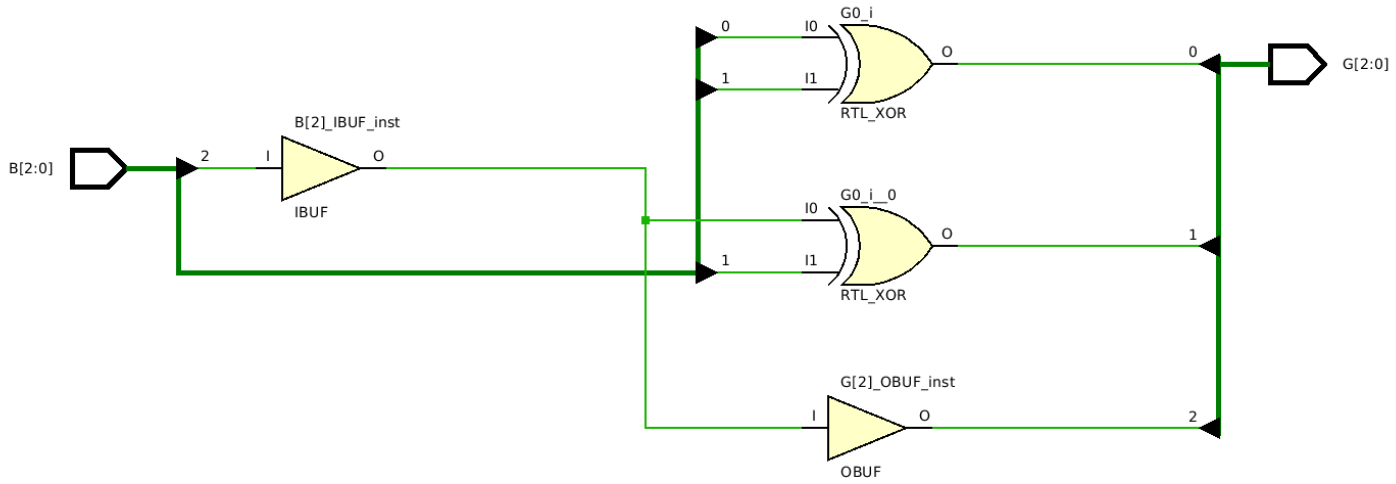
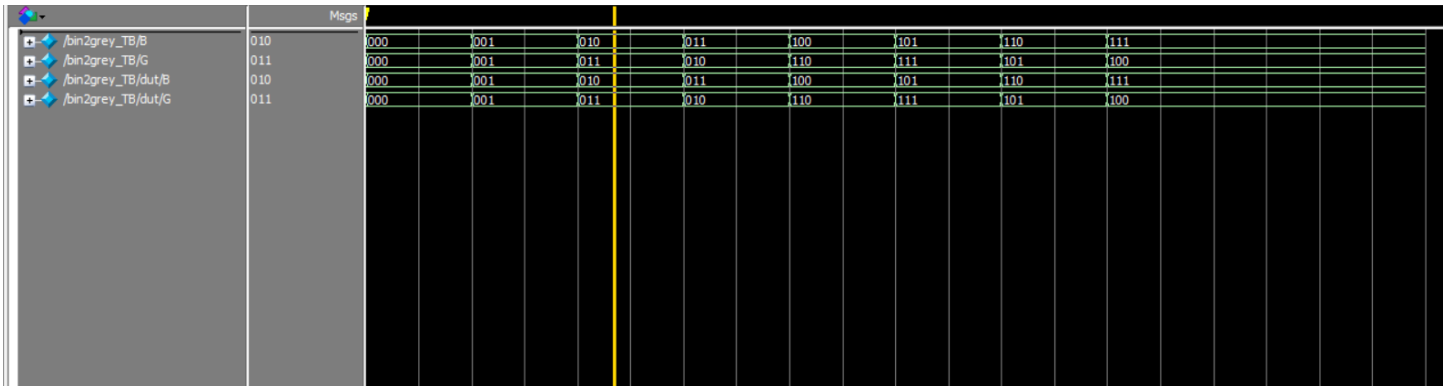
ITI Summer Training

LAB 2

Information
Technology
Institute

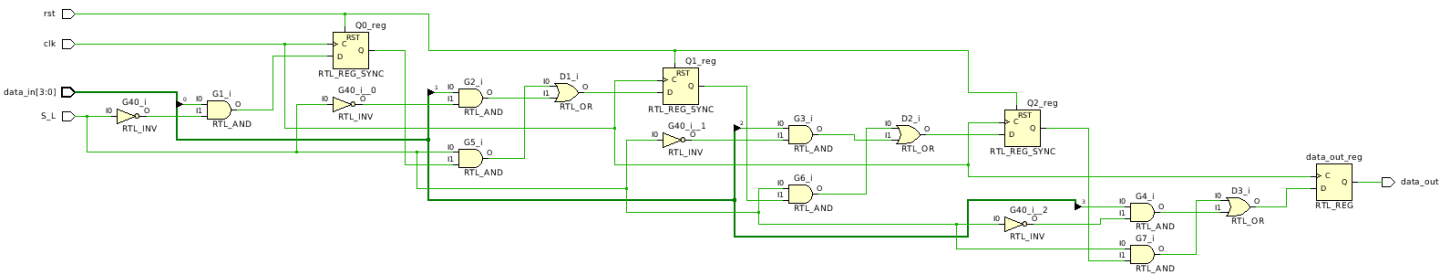
Q1

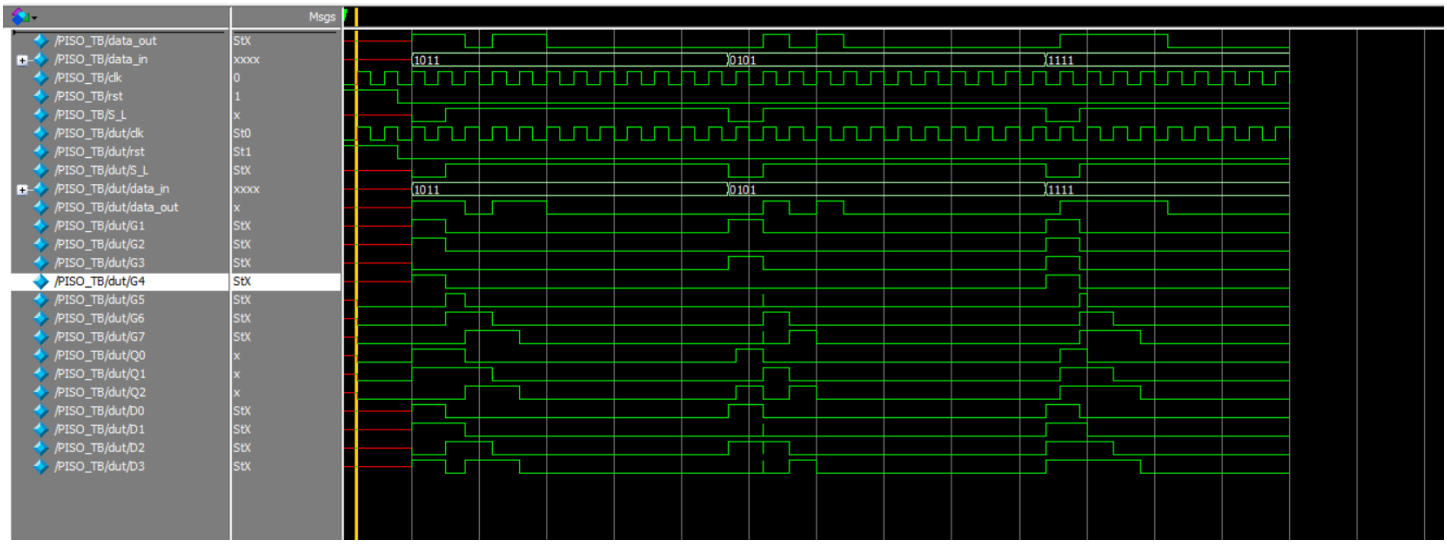

B[2:0]
2
B[2]_IBUF_inst
I
O
IBUF
0
1
I0
I1
G0_i
O
RTL_XOR
G0_i__0
RTL_XOR
G[2]_OBUF_inst
I
O
OBUF
0
1
2
G[2:0]


Msgs
/bin2grey_TB/B 010
/bin2grey_TB/G 011
/bin2grey_TB/dut/B 010
/bin2grey_TB/dut/G 011
000 001 010 011 100 101 110 111
000 001 011 010 110 111 101 100
000 001 010 011 100 101 110 111
000 001 011 010 110 111 101 100

```
module bin2grey(input[2:0] B,output wire[2:0] G );


assign G[0] = B[0] ^ B[1];
assign G[1] = B[2] ^ B[1];
assign G[2] = B[2];

endmodule




module bin2grey_TB();

reg[2:0] B;
wire[2:0] G;

bin2grey dut(.B(B),.G(G));

initial begin

    B= 3'b000;
    #20
    B= 3'b001;
    #20
    B= 3'b010;
    #20
    B= 3'b011;
    #20
    B= 3'b100;
    #20
    B= 3'b101;
    #20
    B= 3'b110;
    #20
    B= 3'b111;

end


endmodule
```

Q2

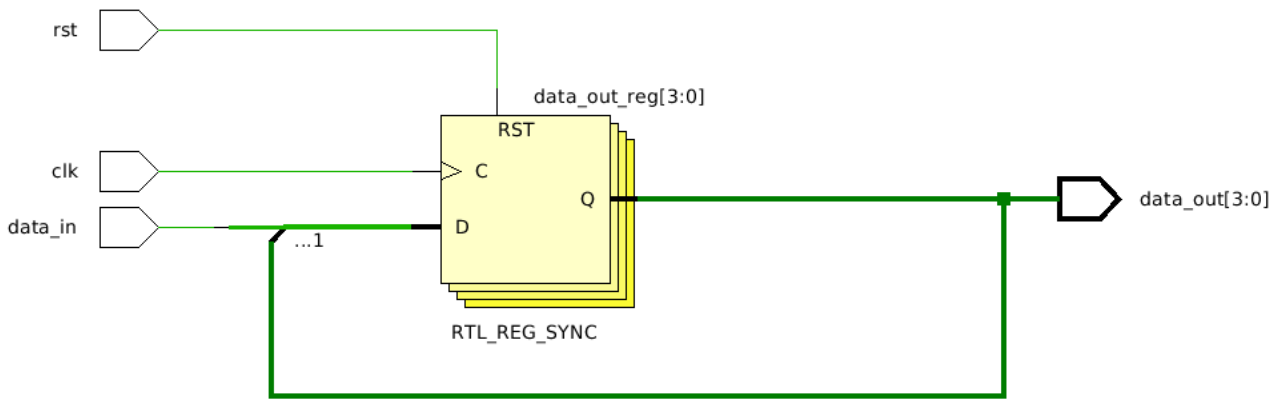

rst
clk
data_in[3:0]
S_L
Q0_reg
RTL_REG_SYNC
G40_i
RTL_INV
G1_i
RTL_AND
G40_i_0
RTL_INV
G2_i
RTL_AND
G5_i
RTL_AND
D1_i
RTL_OR
Q1_reg
RTL_REG_SYNC
G40_i_1
RTL_INV
G3_i
RTL_AND
G6_i
RTL_AND
D2_i
RTL_OR
Q2_reg
RTL_REG_SYNC
G40_i_2
RTL_INV
G4_i
RTL_AND
G7_i
RTL_AND
D3_i
RTL_OR
data_out_reg
RTL_REG
data_out

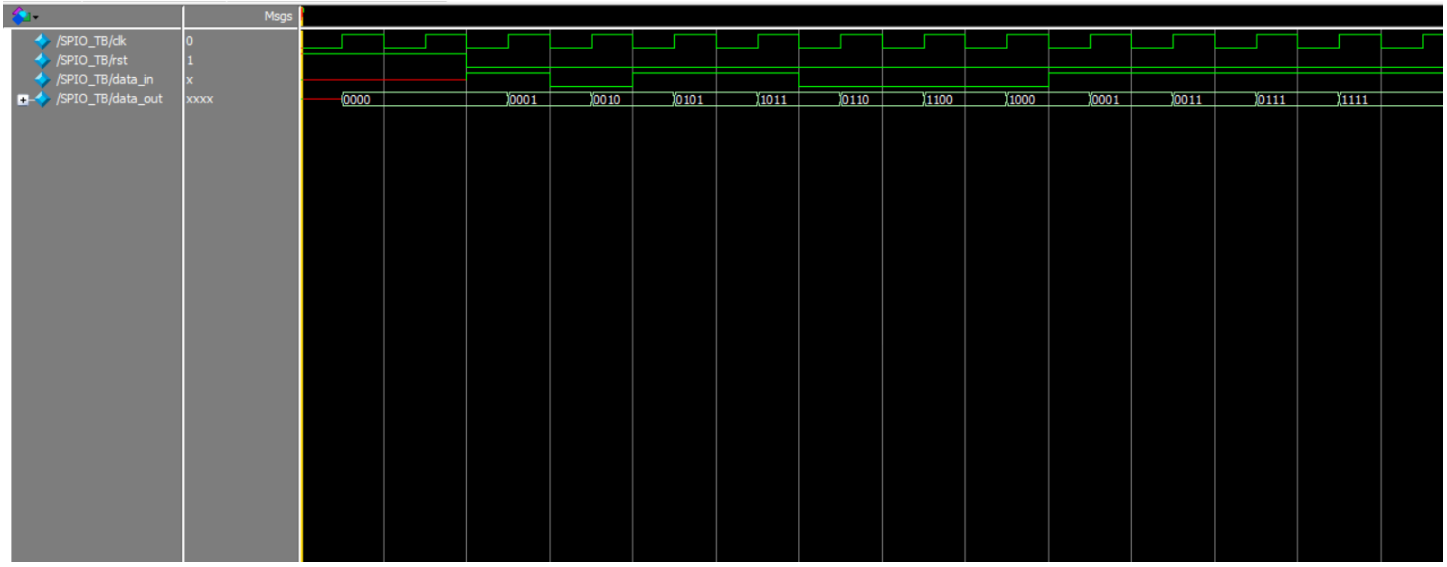

Msgs
/PISO_TB/data_out StX
/PISO_TB/data_in xxxx
/PISO_TB/clk 0
/PISO_TB/rst 1
/PISO_TB/S_L x
/PISO_TB/dut/clk St0
/PISO_TB/dut/rst St1
/PISO_TB/dut/S_L StX
/PISO_TB/dut/data_in xxxx
/PISO_TB/dut/data_out x
/PISO_TB/dut/G1 StX
/PISO_TB/dut/G2 StX
/PISO_TB/dut/G3 StX
/PISO_TB/dut/G4 StX
/PISO_TB/dut/G5 StX
/PISO_TB/dut/G6 StX
/PISO_TB/dut/G7 StX
/PISO_TB/dut/Q0 x
/PISO_TB/dut/Q1 x
/PISO_TB/dut/Q2 x
/PISO_TB/dut/D0 StX
/PISO_TB/dut/D1 StX
/PISO_TB/dut/D2 StX
/PISO_TB/dut/D3 StX
1011
0101
1111
1011
0101
1111

```
module PISO(input clk,rst,S_L,input[3:0] data_in,output reg data_out);

wire G1,G2,G3,G4,G5,G6,G7;
reg Q0,Q1,Q2;
wire D0,D1,D2,D3;


assign G1 = data_in[0] & !S_L;
assign G2 = data_in[1] & !S_L;
assign G3 = data_in[2] & !S_L;
assign G4 = data_in[3] & !S_L;

assign G5 = S_L & Q0;
assign G6 = S_L & Q1;
assign G7 = S_L & Q2;

assign D0 = G1;
assign D1 = G5 | G2;
assign D2 = G6 | G3;
assign D3 = G7 | G4;

always@(posedge clk) begin

 if(rst) begin
   data_out<=1'bx;
     Q0 <= 0;
     Q1 <= 0;
     Q2 <= 0;

 end

 else begin

     Q0 <= D0;
     Q1 <= D1;
     Q2 <= D2;
     data_out <= D3;
 end


end

endmodule




module PISO_TB();

  wire data_out;
  reg[3:0] data_in;
  reg clk,rst,S_L;

  PISO dut(.clk(clk),.rst(rst),.S_L(S_L),.data_in(data_in),.data_out(data_out));


  initial begin
    clk = 0;
   forever begin

    clk = #10 ~clk;

   end

  end


 initial begin

  rst = 1;

  #40

  rst = 0;

  #10
  S_L = 0;
  data_in=4'b1011;
  #25
```

```
    S_L = 1;
    #200



     #10
    S_L = 0;
    data_in=4'b0101;
    #25
    S_L = 1;

   #200

      #10
    S_L = 0;
    data_in=4'b1111;
    #25
    S_L = 1;


   end



  endmodule

```

Q3

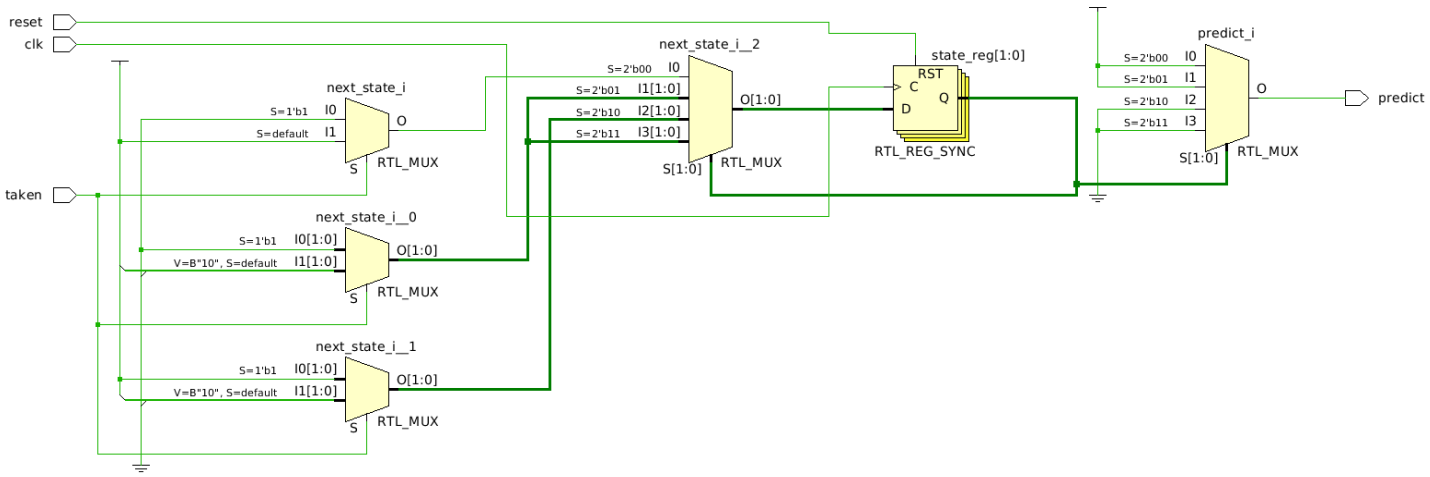


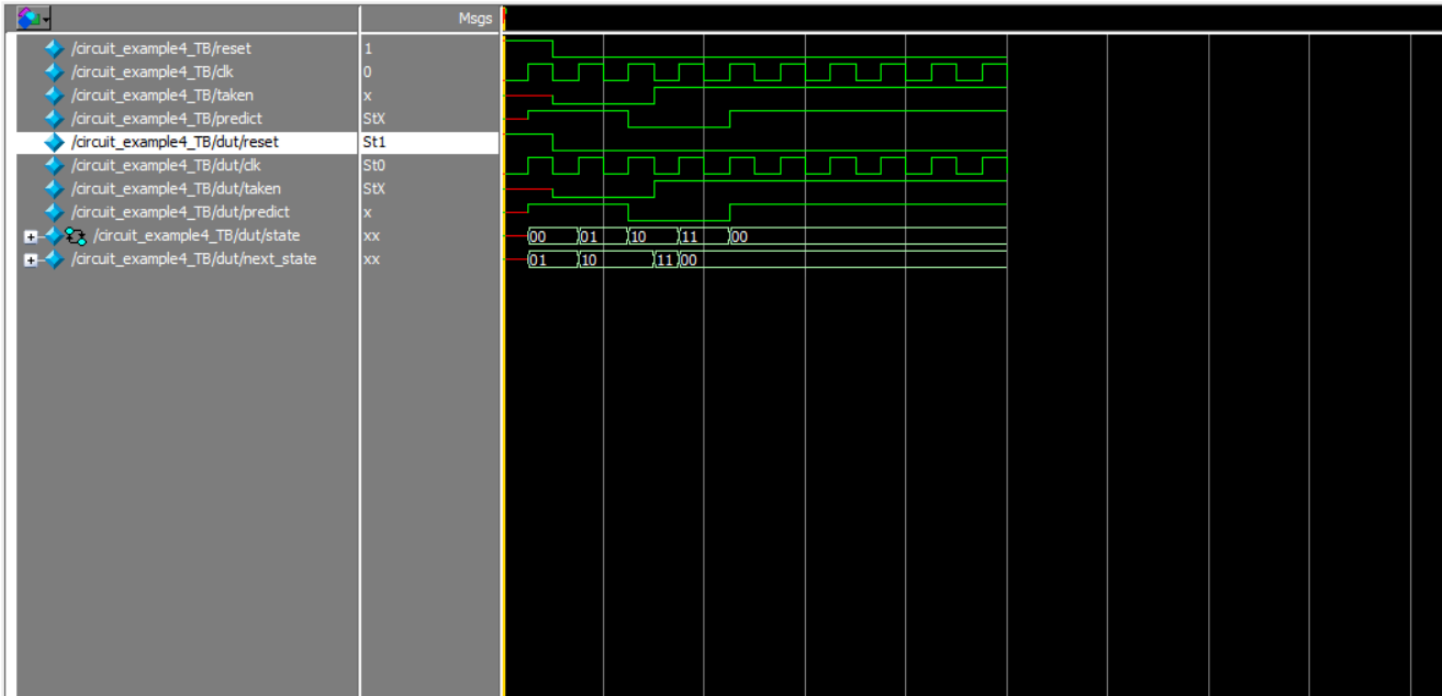



```
module SPIO(input clk,rst,input data_in,output reg [3:0]data_out);



always@(posedge clk) begin

 if(rst) begin
   data_out<=0;
 end

 else begin

     data_out[0] <= data_in;
     data_out[1] <= data_out[0];
     data_out[2] <= data_out[1];
     data_out[3] <= data_out[2];
 end



end

endmodule



module SPIO_TB();


reg clk,rst,data_in;

wire[3:0] data_out;


SPIO dut(.clk(clk),.rst(rst),.data_in(data_in),.data_out(data_out));

initial begin
  clk =0;
  forever begin
```

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

end



initial begin

  rst = 1;
  #20
  rst = 0;


  data_in = 1;
  #10
  data_in = 0;
  #10
  data_in = 1;
  #10
  data_in = 1;
  #10




  data_in = 0;
  #10
  data_in = 0;
  #10
  data_in = 0;
  #10
  data_in = 1;
```

Q4



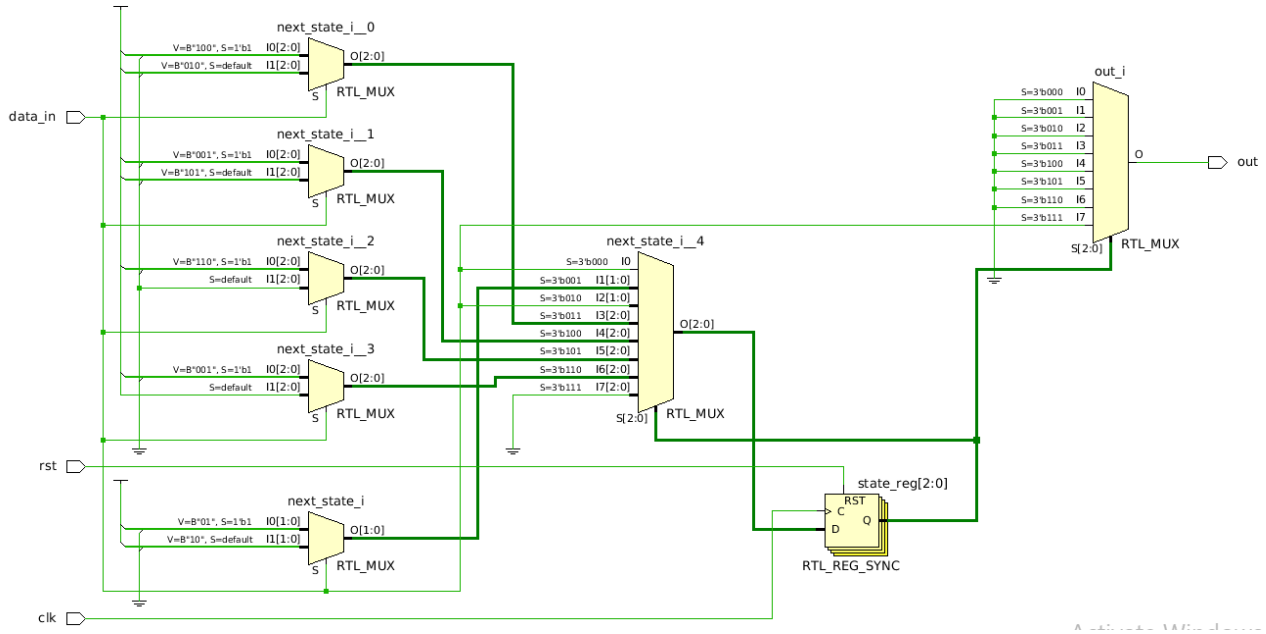


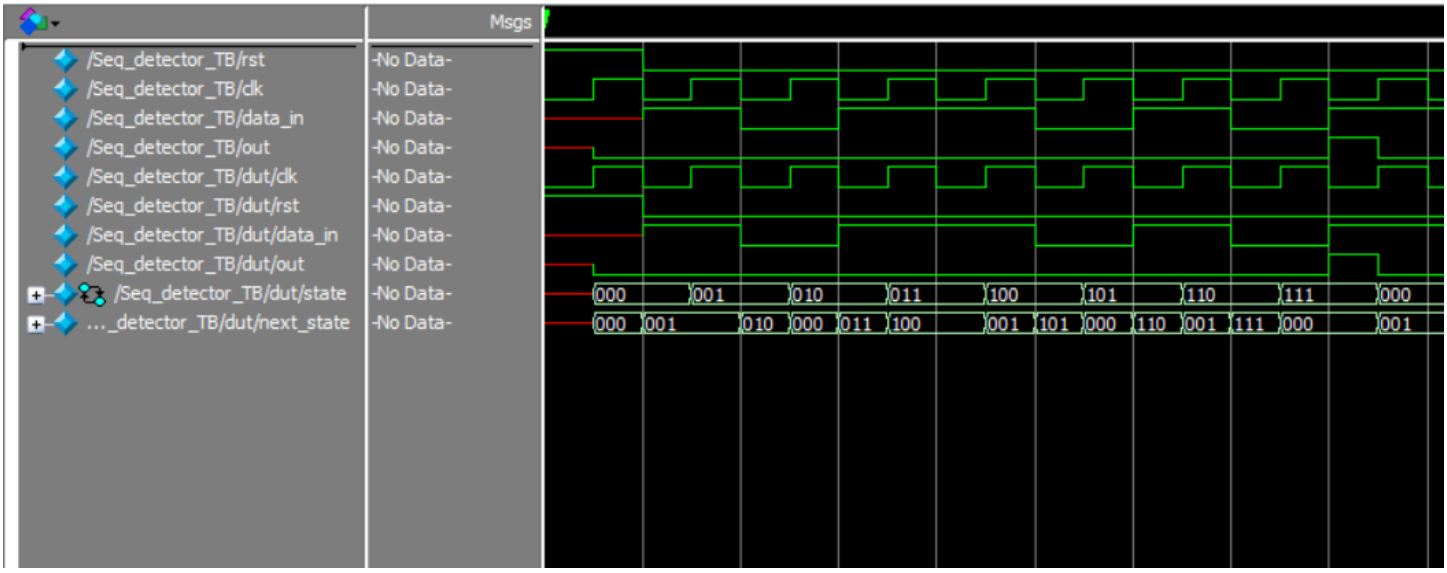

```
module circuit_example4(input reset ,clk,taken,output reg predict);

  reg[1:0] state,next_state;

  always@(posedge clk)begin

    if(reset) begin

      state <= 2'b00;

    end

    else
      state <= next_state;

  end



  always@(taken or state) begin

    case(state)

      2'b00: begin

            if(taken)
                 next_state = 2'b00;

            else
                 next_state = 2'b01;

             end



      2'b01: begin

            if(taken)
                 next_state = 2'b00;

```

```
            else
                next_state = 2'b10;

             end



        2'b10: begin

            if(taken)
                next_state = 2'b11;

            else
                next_state = 2'b10;

             end


        2'b11: begin

            if(taken)
                next_state = 2'b00;

            else
                next_state = 2'b10;

             end


      endcase


    end
```

```
  always@(state) begin

    case(state)

      2'b00: predict = 1 ;

      2'b01: predict = 1 ;

      2'b10: predict = 0 ;

      2'b11: predict = 0 ;


    endcase


  end


endmodule

```

Q5






```
module Seq_detector (input clk,rst,data_in,output reg out);


  parameter state_0 = 3'b000;
  parameter state_1 = 3'b001;
  parameter state_2 = 3'b010;
  parameter state_3 = 3'b011;
  parameter state_4 = 3'b100;
  parameter state_5 = 3'b101;
  parameter state_6 = 3'b110;
  parameter state_7 = 3'b111;


  reg[2:0] state,next_state;




  always@(posedge clk) begin

    if(rst)
            state <= state_0;

    else
            state <= next_state;

  end



 always@(state or data_in) begin


   case(state)

   state_0: begin

             if(data_in) begin
                         next_state <= state_1;
                         out <= 0;
                        end
```

```
                        end

            else        begin
                         next_state <= state_0;
                         out <= 0;
                        end
          end


    state_1: begin

            if(data_in) begin
                        next_state <= state_1;
                        out <= 0;
                        end

            else        begin
                        next_state <= state_2;
                        out <= 0;
                        end

          end


    state_2: begin

            if(data_in) begin
                        next_state <= state_3;
                        out <= 0;
                        end

            else        begin
                        next_state <= state_0;
                        out <= 0;
                        end

          end



    state_3: begin
```

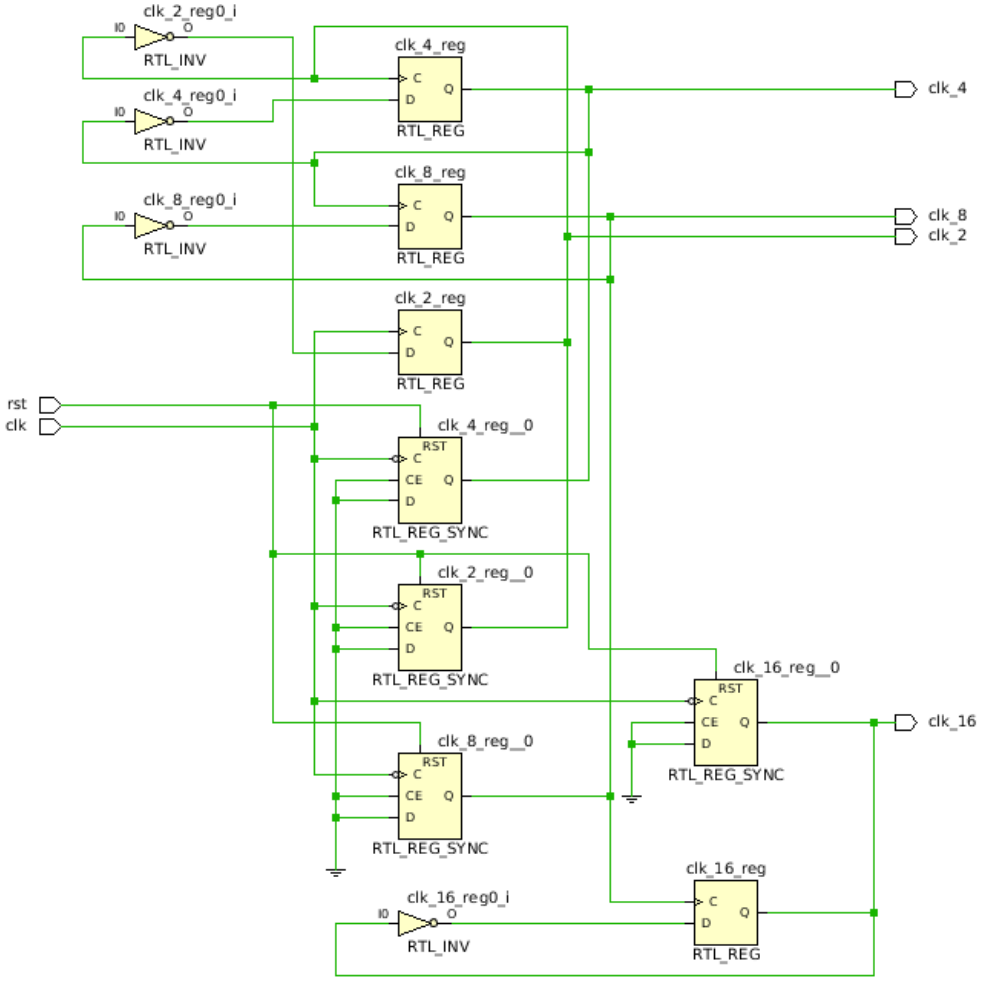
```
                if(data_in) begin
                            next_state <= state_4;
                            out <= 0;
                            end

                else        begin
                            next_state <= state_2;
                            out <= 0;
                            end

            end


    state_4: begin

                if(data_in) begin
                            next_state <= state_1;
                            out <= 0;
                            end

                else        begin
                            next_state <= state_5;
                            out <= 0;
                            end

            end



    state_5: begin

                if(data_in) begin
                            next_state <= state_6;
                            out <= 0;
                            end

                else        begin
                            next_state <= state_0;
                            out <= 0;
```

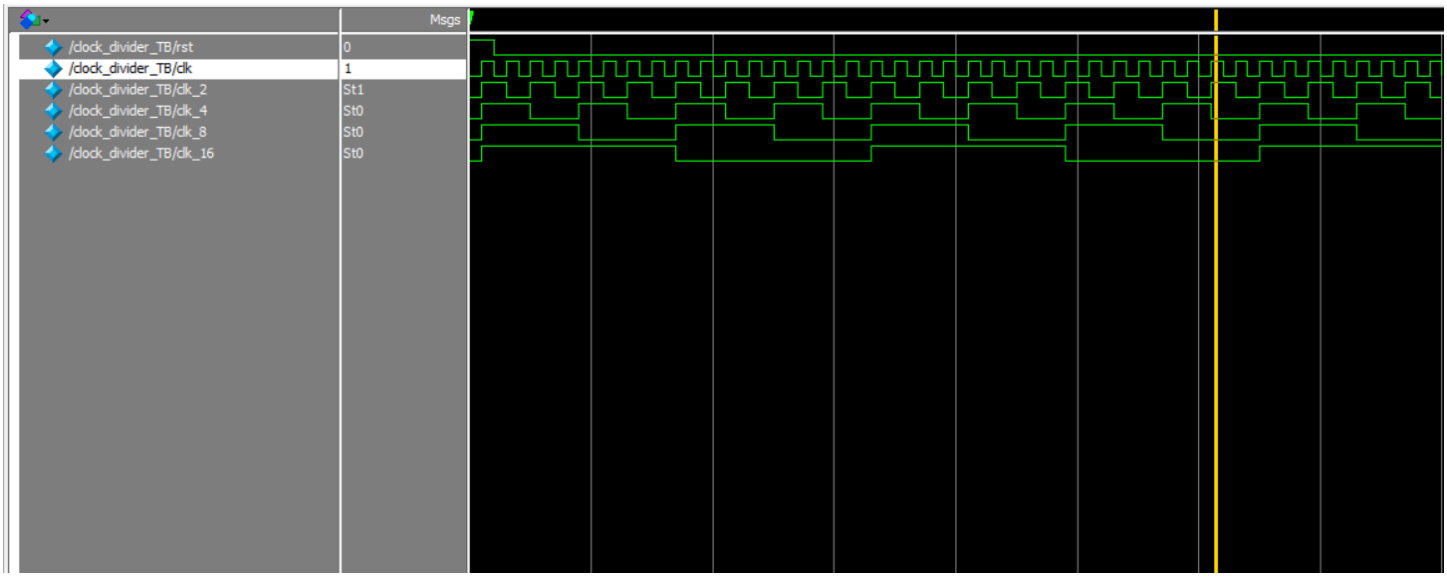
```
                        end

                end


        state_6: begin

                  if(data_in) begin
                              next_state <= state_1;
                              out <= 0;
                              end

                  else        begin
                              next_state <= state_7;
                              out <= 0;
                              end

                end


        state_7:  begin

                  if(data_in) begin
                              next_state <= state_0;
                              out <= 1;
                              end


                  else        begin
                              next_state <= state_0;
                              out <= 0;
                              end

                end



        endcase


```

```
 end


endmodule




module Seq_detector_TB();


reg rst ,clk,data_in;
wire out;


Seq_detector dut(.rst(rst),.clk(clk),.data_in(data_in),.out(out));


initial begin
  clk = 0;
  forever begin

    clk = #5 ~clk;

  end

end


initial begin

 rst = 1;

 #10

 rst = 0;

```

```
data_in = 1 ;

#10
data_in = 0 ;

#10
data_in = 1 ;

#10
data_in = 1 ;

#10
data_in = 0 ;

#10
data_in = 1 ;

#10
data_in = 0 ;

#10
data_in = 1 ;




end



endmodule

```

Q6

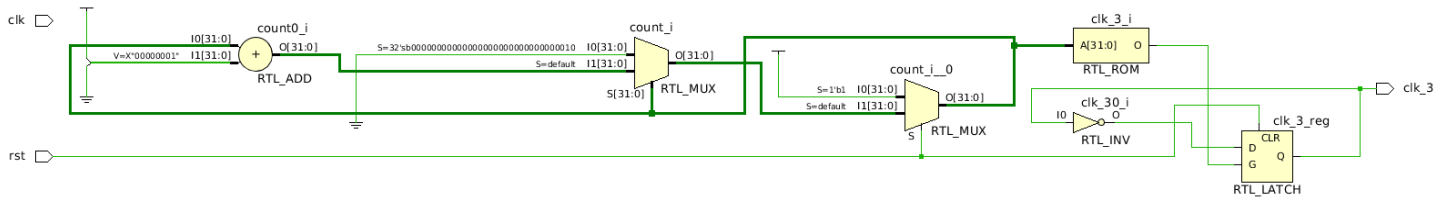

clk_2_reg0_i
RTL_INV
clk_4_reg0_i
RTL_INV
clk_8_reg0_i
RTL_INV
clk_4_reg
RTL_REG
clk_8_reg
RTL_REG
clk_2_reg
RTL_REG
clk_4
clk_8
clk_2
rst
clk
clk_4_reg__0
RTL_REG_SYNC
clk_2_reg__0
RTL_REG_SYNC
clk_8_reg__0
RTL_REG_SYNC
clk_16_reg__0
RTL_REG_SYNC
clk_16
clk_16_reg0_i
RTL_INV
clk_16_reg
RTL_REG

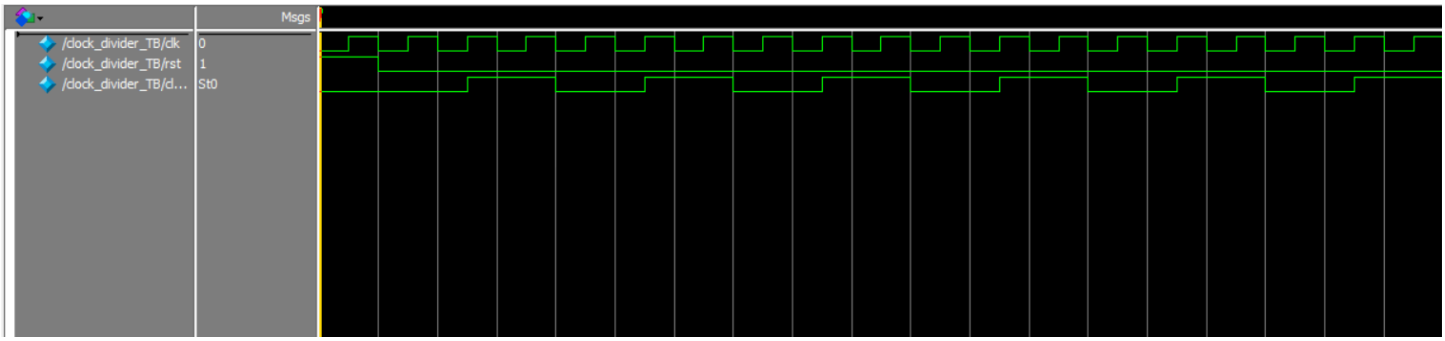

Msgs
/clock_divider_TB/rst 0
/clock_divider_TB/clk 1
/clock_divider_TB/clk_2 St1
/clock_divider_TB/clk_4 St0
/clock_divider_TB/clk_8 St0
/clock_divider_TB/clk_16 St0

```
module clock_divider_1(input clk,rst,output reg clk_2,clk_4,clk_8,clk_16);



  always@(negedge clk) begin

    if(rst) begin
      clk_2 <= 0;
      clk_4 <= 0;
      clk_8 <= 0;
      clk_16 <= 0;
    end

  end




  always@(posedge clk) begin

    clk_2 <= ~clk_2;

  end


  always@(posedge clk_2) begin

    clk_4 <= ~clk_4;

  end


  always@(posedge clk_4) begin

   clk_8 <= ~clk_8;

 end



  always@(posedge clk_8) begin

   clk_16 <= ~clk_16;

 end


endmodule



module clock_divider_TB();

  reg clk,rst;
  wire clk_2,clk_4,clk_8,clk_16;

  clock_divider_1 dut(.clk(clk),.rst(rst),.clk_2(clk_2),.clk_4(clk_4),.clk_8(clk_8),.clk_16(clk_16));

  initial begin
     clk = 0;
    forever begin

        clk = #5 ~clk;

    end

  end
```

```
  initial begin

    rst = 1;

    #10

    rst = 0;




  end

endmodule
```

Q7

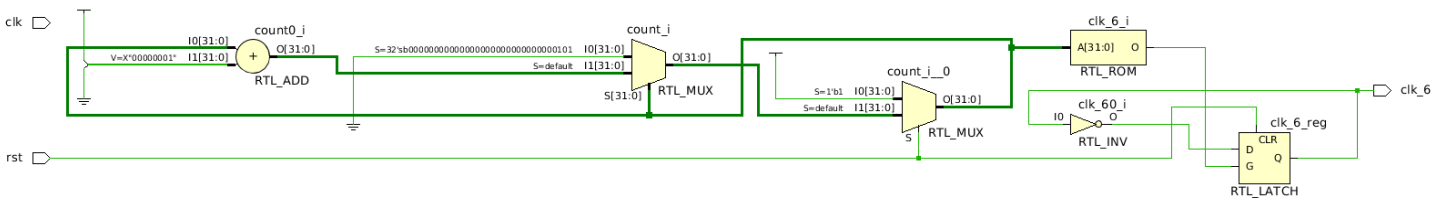


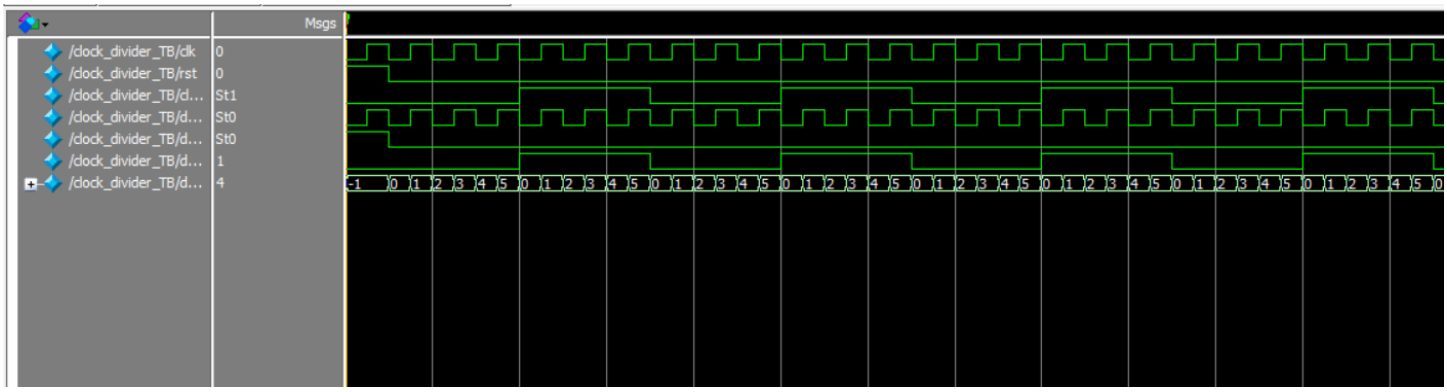


```
module clock_divider_2(input clk,rst,output reg clk_3);


 integer count;

 always@(clk) begin

   if(rst) begin
     clk_3 <= 0;
     count = -1;
   end

   else if(count == 2)begin
    clk_3 <= ~clk_3;
    count = 0;
    end

   else count = count +1;

 end









endmodule
```

```
33  module clock_divider_TB();
34
35    reg clk,rst;
36    wire clk_3;
37
38    clock_divider_2 dut(.clk(clk),.rst(rst),.clk_3(clk_3));
39
40    initial begin
41       clk = 0;
42      forever begin
43
44          clk = #5 ~clk;
45
46      end
47
48    end
49
50
51
52    initial begin
53
54      rst = 1;
55
56      #10
57
58      rst = 0;
59
60
61
62
63
64
65    end
66
67  endmodule
68
69
```

Q8

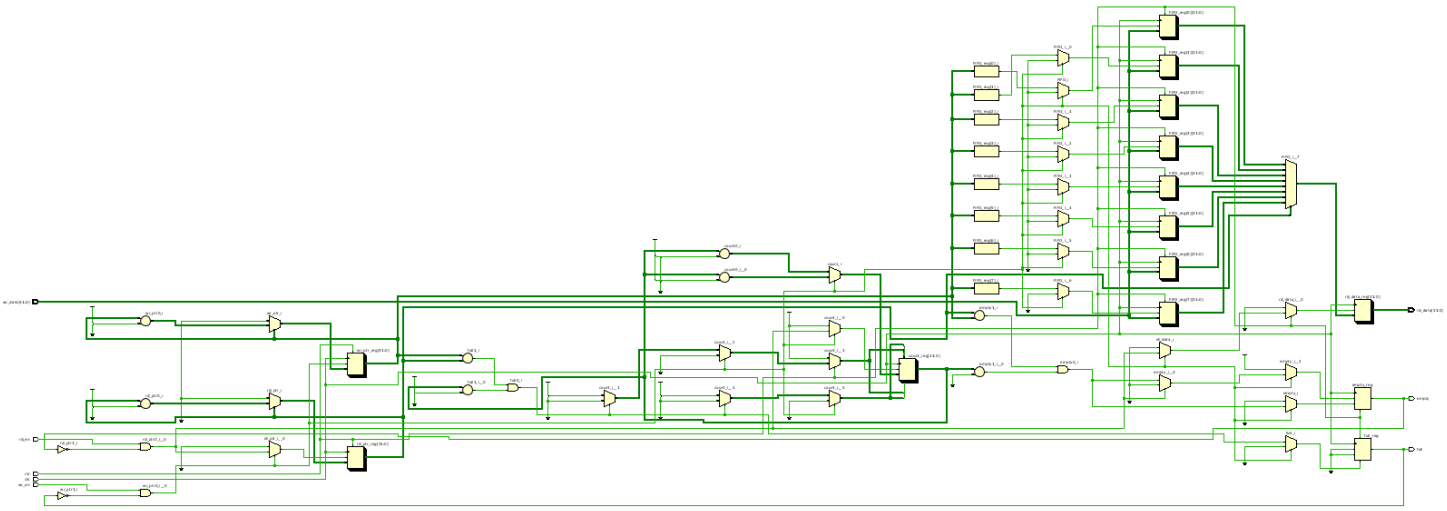


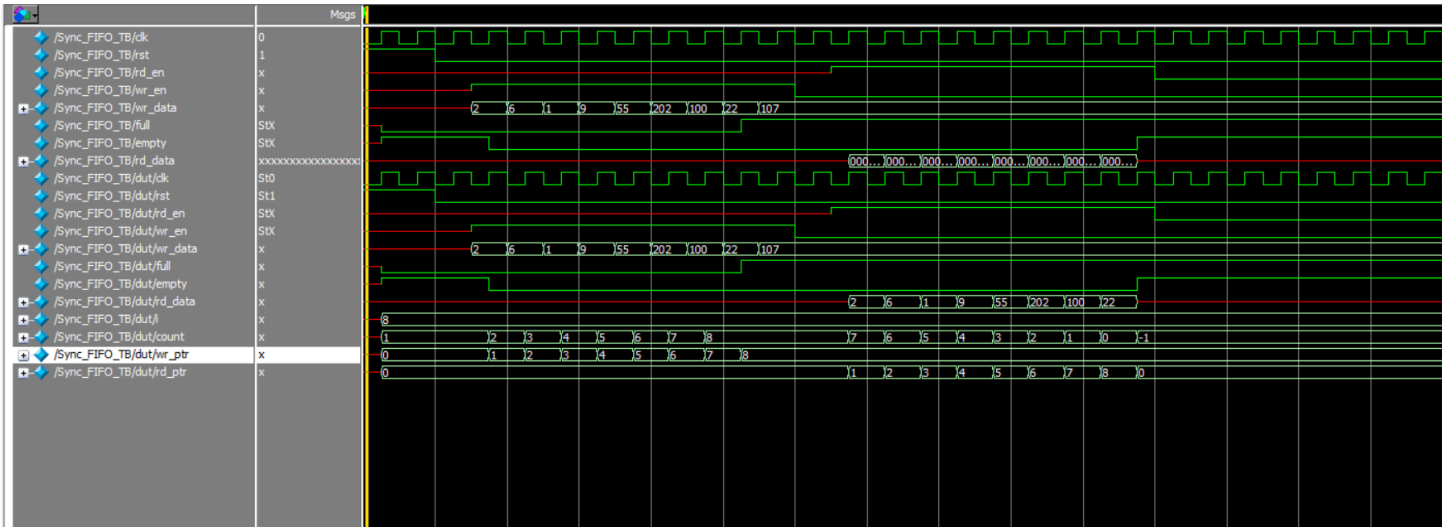



```
module clock_divider_2(input clk,rst,output reg clk_6);


 integer count;

 always@(clk) begin

   if(rst) begin
     clk_6 <= 0;
     count = -1;
   end

   else if(count == 5)begin
    clk_6 <= ~clk_6;
    count = 0;
    end

   else count = count +1;

 end









endmodule
```

```
module clock_divider_TB();

  reg clk,rst;
  wire clk_6;

  clock_divider_2 dut(.clk(clk),.rst(rst),.clk_6(clk_6));

  initial begin
     clk = 0;
    forever begin

        clk = #5 ~clk;

    end

  end



  initial begin

    rst = 1;

    #10

    rst = 0;





  end

endmodule
```

Q9






```
module Sync_FIFO(input clk,rst,rd_en,wr_en,input[31:0] wr_data,output reg full,empty,output reg[31:0] rd_data);



  reg[31:0] FIFO [0:7];
  integer i,count;
  integer wr_ptr,rd_ptr;



  always@(posedge clk) begin

     if(rst) begin

       wr_ptr <= 0;
       rd_ptr <= 0;
       empty <= 1;
       full <= 0;
       count <= 1;
       for(i = 0 ; i<8 ; i = i+1)
             FIFO[i] <= 0;

      end


     else if (wr_en & !full) begin

       FIFO[wr_ptr] <= wr_data;
       wr_ptr <= wr_ptr + 1;
       empty <= 0;
       if( wr_ptr == 8) wr_ptr <= wr_ptr % 8;

       count <= count + 1;

       if(wr_ptr > rd_ptr && count==8) begin
                           full <= 1;
                           count <= 8;
                        end

```

```
      end


      else if (rd_en & !empty) begin

        rd_data <= FIFO[rd_ptr];
        rd_ptr <= rd_ptr + 1;
        count <= count - 1;
        if( rd_ptr == 8 ) rd_ptr <= rd_ptr % 8;

        if(rd_ptr == wr_ptr && count==0)
                            empty <= 1;

      end
   end

endmodule




module Sync_FIFO_TB();

reg clk,rst,rd_en,wr_en;
reg[31:0] wr_data;
wire full,empty;
wire[31:0] rd_data ;


 Sync_FIFO dut(.clk(clk),.rst(rst),.rd_en(rd_en),.wr_en(wr_en),.wr_data(wr_data),.full(full),.empty(empty),.rd_data(rd_data));

 initial begin
    clk = 0;
   forever begin
     clk = #5 ~clk;
   end

end
```

```
83  initial begin
84
85    rst =1;
86
87    #20
88
89    rst = 0;
90
91    #10
92
93    wr_en = 1;
94    wr_data = 2;
95
96    #10
97
98    wr_en = 1;
99    wr_data = 6;
100
101   #10
102
103   wr_en = 1;
104   wr_data = 1;
105
106   #10
107
108   wr_en = 1;
109   wr_data = 9;
110
111   #10
112
113   wr_en = 1;
114   wr_data = 55;
115
116   #10
117
118   wr_en = 1;
119   wr_data = 202;
120
```

```
121     #10
122
123     wr_en = 1;
124     wr_data = 100;
125
126
127      #10
128
129     wr_en = 1;
130     wr_data = 22;
131
132     #10
133
134     wr_en = 1;
135     wr_data = 107;
136
137     #10
138     wr_en = 0;
139
140     #10
141
142     rd_en = 1;
143
144     #10
145
146     rd_en = 1;
147     #10
148
149     rd_en = 1 ;
150     #10
151
152     rd_en = 1 ;
153     #10
154
155     rd_en = 1 ;
156     #10
157
158     rd_en = 1 ;
159     #10
160
```

```
161      rd_en = 1 ;
162      #10
163
164      rd_en = 1 ;
165
166      #10
167
168
169      rd_en = 1 ;
170     #10
171       rd_en = 0;
172   end
173
174
175   endmodule
176
177
178
```