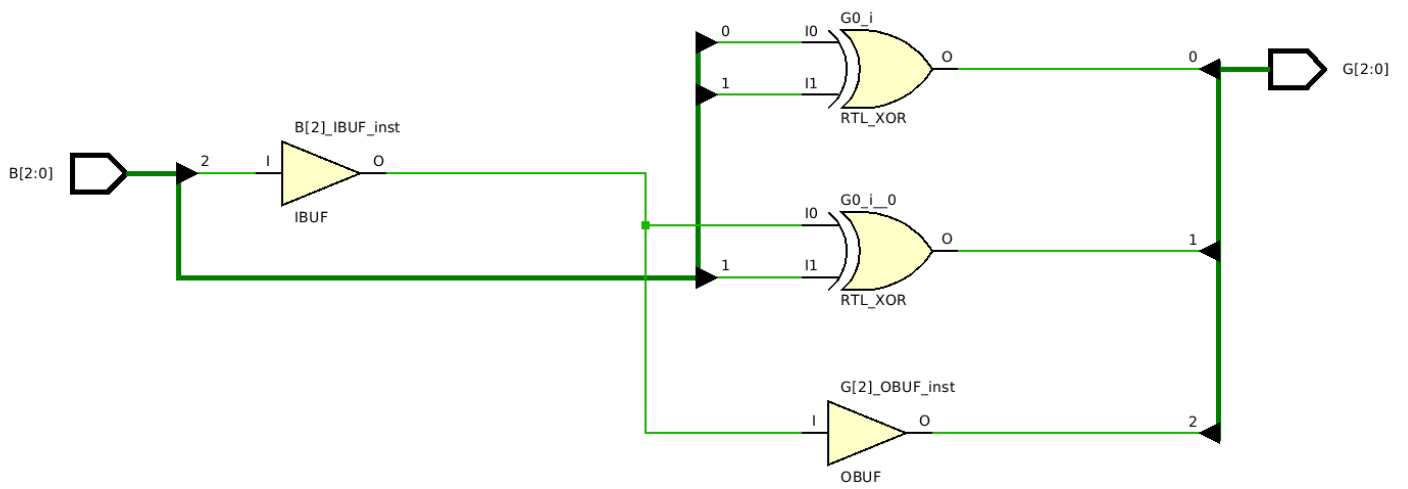










ITI Summer Training

LAB 2



Q1



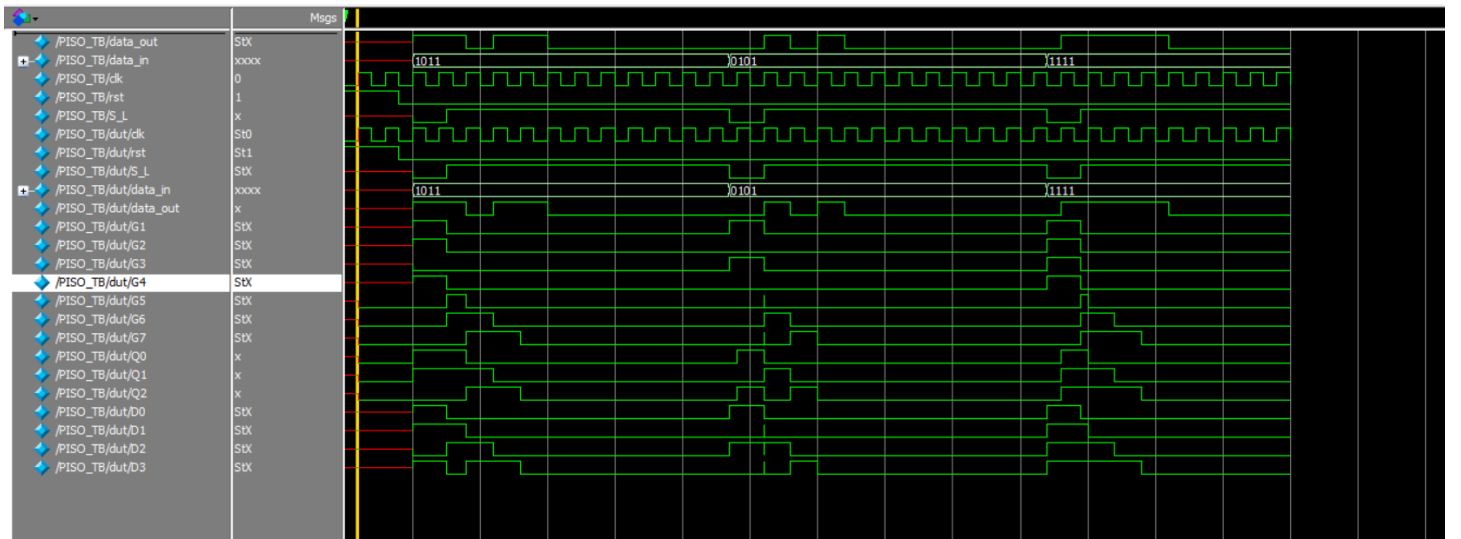
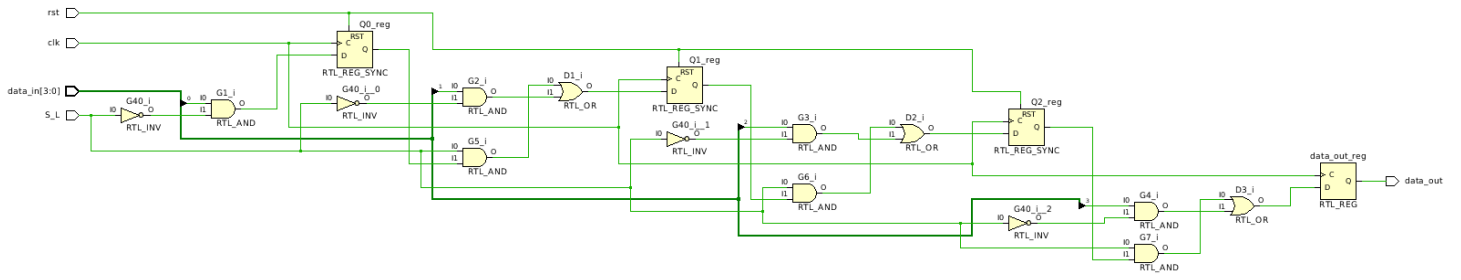
		Msgs															
  /bin2grey_TB/B	010	000	001	010	011	100	101	110	111								
  /bin2grey_TB/G	011	000	001	011	010	110	111	101	100								
  /bin2grey_TB/dut/B	010	000	001	010	011	100	101	110	111								
  /bin2grey_TB/dut/G	011	000	001	011	010	110	111	101	100								

```

1
2 module bin2grey(input[2:0] B,output wire[2:0] G );
3
4
5 assign G[0] = B[0] ^ B[1];
6 assign G[1] = B[2] ^ B[1];
7 assign G[2] = B[2];
8
9 endmodule
10
11
12
13
14 module bin2grey_TB();
15
16 reg[2:0] B;
17 wire[2:0] G;
18
19 bin2grey dut(.B(B),.G(G));
20
21 initial begin
22
23     B= 3'b000;
24     #20
25     B= 3'b001;
26     #20
27     B= 3'b010;
28
29     #20
30     B= 3'b011;
31     #20
32     B= 3'b100;
33     #20
34     B= 3'b101;
35     #20
36     B= 3'b110;
37     #20
38     B= 3'b111;
39
40 end
41
42 endmodule

```

Q2



```

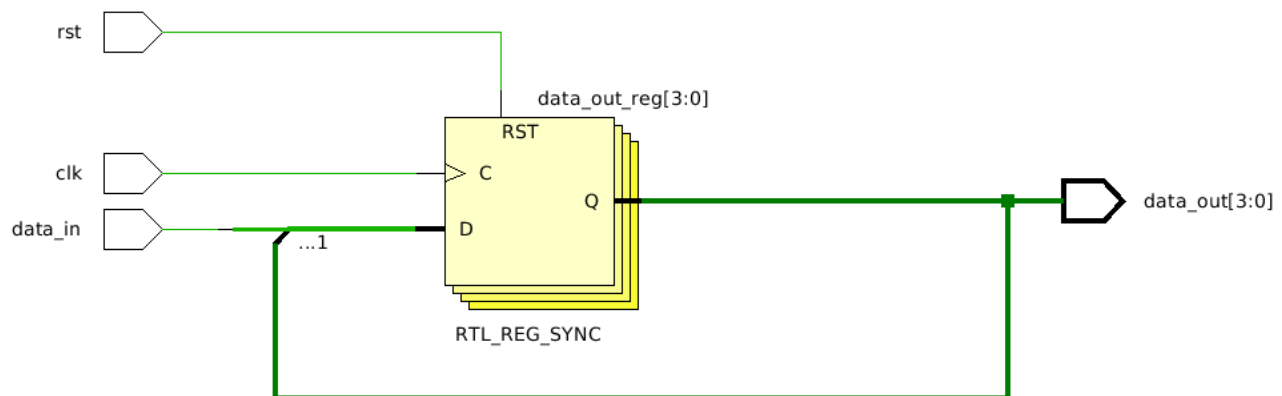
1  module PISO(input clk,rst,S_L,input[3:0] data_in,output reg data_out);
2
3  wire G1,G2,G3,G4,G5,G6,G7;
4  reg Q0,Q1,Q2;
5  wire D0,D1,D2,D3;
6
7
8  assign G1 = data_in[0] & !S_L;
9  assign G2 = data_in[1] & !S_L;
10 assign G3 = data_in[2] & !S_L;
11 assign G4 = data_in[3] & !S_L;
12
13 assign G5 = S_L & Q0;
14 assign G6 = S_L & Q1;
15 assign G7 = S_L & Q2;
16
17 assign D0 = G1;
18 assign D1 = G5 | G2;
19 assign D2 = G6 | G3;
20 assign D3 = G7 | G4;
21
22 always@(posedge clk) begin
23
24     if(rst) begin
25         data_out<=1'bx;
26         Q0 <= 0;
27         Q1 <= 0;
28         Q2 <= 0;
29
30     end
31
32     else begin
33
34         Q0 <= D0;
35         Q1 <= D1;
36         Q2 <= D2;
37         data_out <= D3;
38     end
39
40
41
42 end
43
44 endmodule
45
46
47
48
49
50 module PISO_TB();
51
52     wire data_out;
53     reg[3:0] data_in;
54     reg clk,rst,S_L;
55
56     PISO dut(.clk(clk),.rst(rst),.S_L(S_L),.data_in(data_in),.data_out(data_out));
57
58
59     initial begin
60         clk = 0;
61         forever begin
62
63             clk = #10 ~clk;
64
65         end
66
67     end
68
69
70     initial begin
71
72         rst = 1;
73
74         #40
75
76         rst = 0;
77
78         #10
79         S_L = 0;
80         data_in=4'b1011;
81         #25

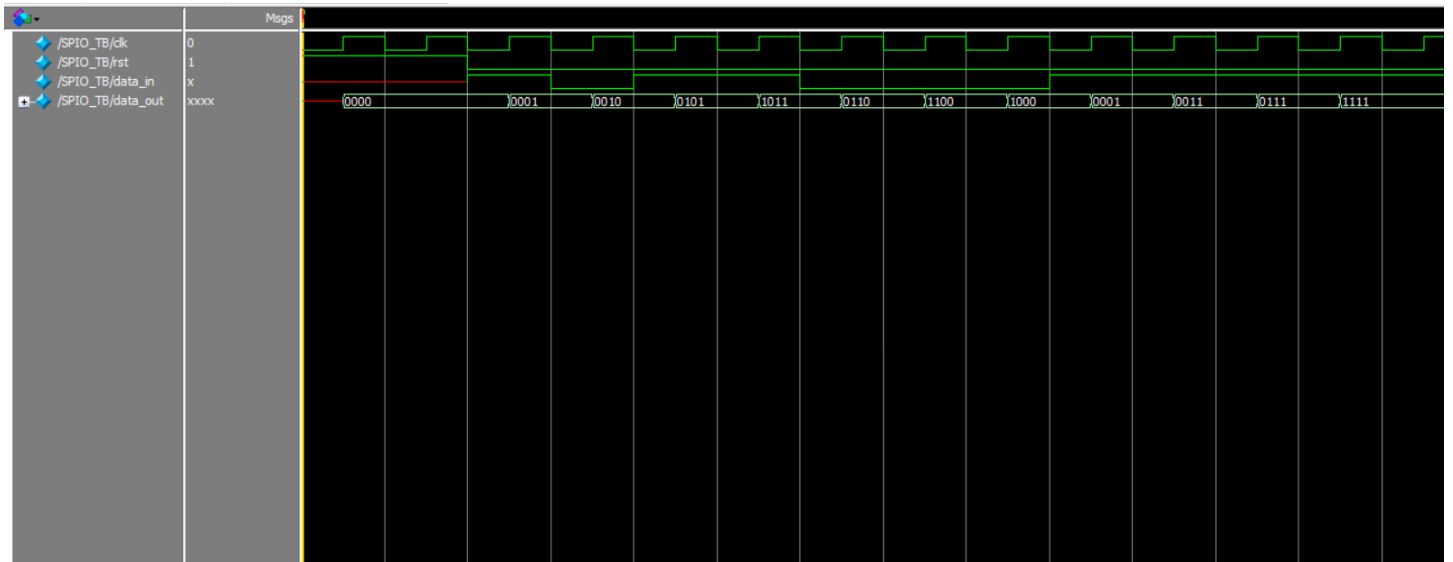
```

```

82     S_L = 1;
83     #200
84
85
86
87     #10
88     S_L = 0;
89     data_in=4'b0101;
90     #25
91     S_L = 1;
92
93     #200
94
95     #10
96     S_L = 0;
97     data_in=4'b1111;
98     #25
99     S_L = 1;
100
101
102     end
103
104
105
106
107     endmodule
108
---
```

Q3





```

1  module SPIO(input clk,rst,input data_in,output reg [3:0]data_out);
2
3
4
5
6  always@(posedge clk) begin
7
8      if(rst) begin
9          data_out<=0;
10         end
11
12         else begin
13
14             data_out[0] <= data_in;
15             data_out[1] <= data_out[0];
16             data_out[2] <= data_out[1];
17             data_out[3] <= data_out[2];
18         end
19
20
21
22     end
23
24 endmodule
25
26
27
28
29 module SPIO_TB();
30
31
32     reg clk,rst,data_in;
33
34     wire[3:0] data_out;
35
36
37     SPIO dut(.clk(clk),.rst(rst),.data_in(data_in),.data_out(data_out));
38
39     initial begin
40         clk =0;
41         forever begin

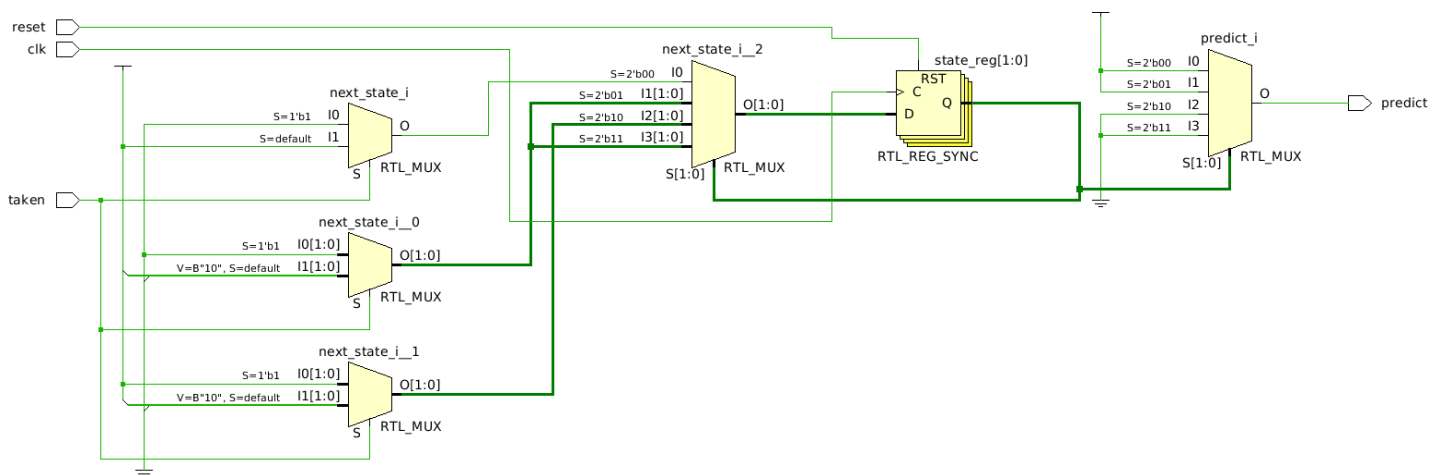
```

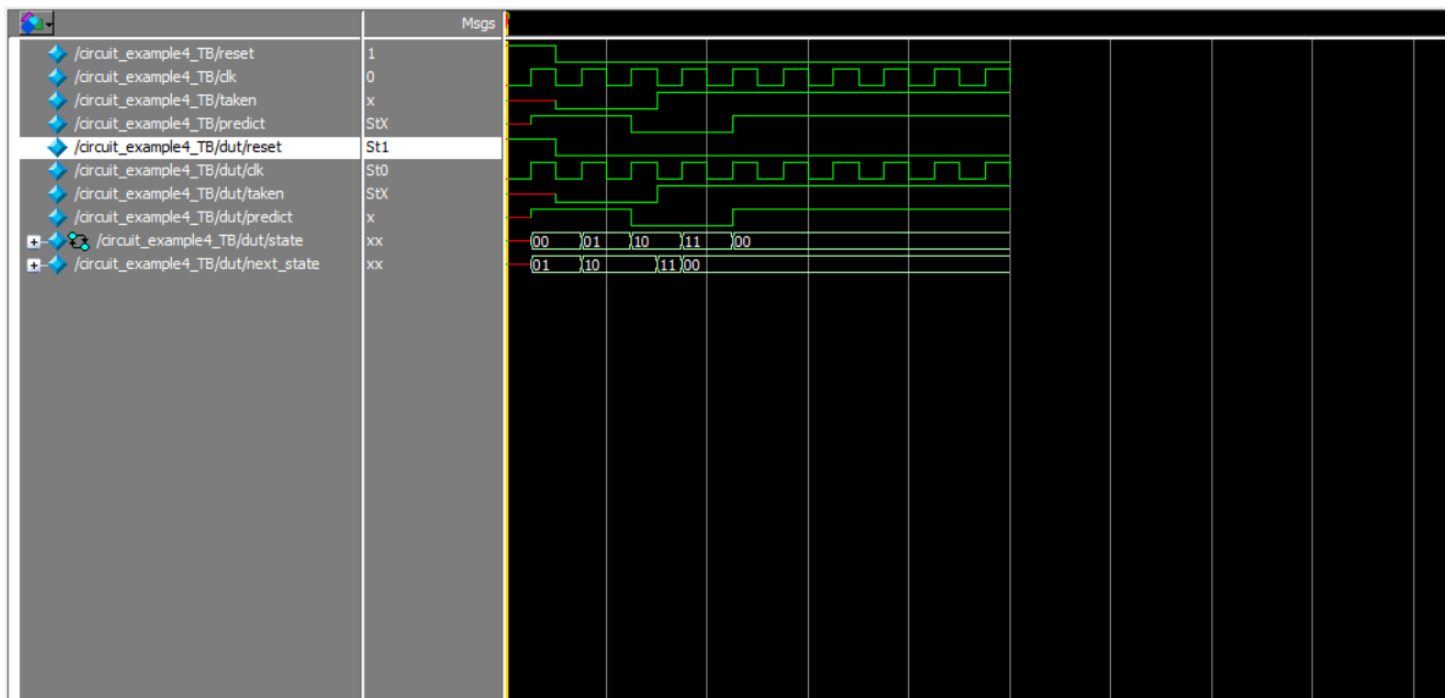
```

41     forever begin
42         clk = #5 ~clk;
43     end
44
45 end
46
47
48
49 initial begin
50
51     rst = 1;
52     #20
53     rst = 0;
54
55
56     data_in = 1;
57     #10
58     data_in = 0;
59     #10
60     data_in = 1;
61     #10
62     data_in = 1;
63     #10
64
65
66
67
68
69     data_in = 0;
70     #10
71     data_in = 0;
72     #10
73     data_in = 0;
74     #10
75     data_in = 1;
76

```

Q4





```

1  module circuit_example4(input reset ,clk,taken,output reg predict);
2
3      reg[1:0] state,next_state;
4
5      always@(posedge clk)begin
6
7          if(reset) begin
8
9              state <= 2'b00;
10
11          end
12
13          else
14              state <= next_state;
15
16      end
17
18
19
20      always@(taken or state) begin
21
22          case(state)
23
24              2'b00: begin
25
26                  if(taken)
27                      next_state = 2'b00;
28
29                  else
30                      next_state = 2'b01;
31
32                  end
33
34
35              2'b01: begin
36
37                  if(taken)
38                      next_state = 2'b00;
39
40

```

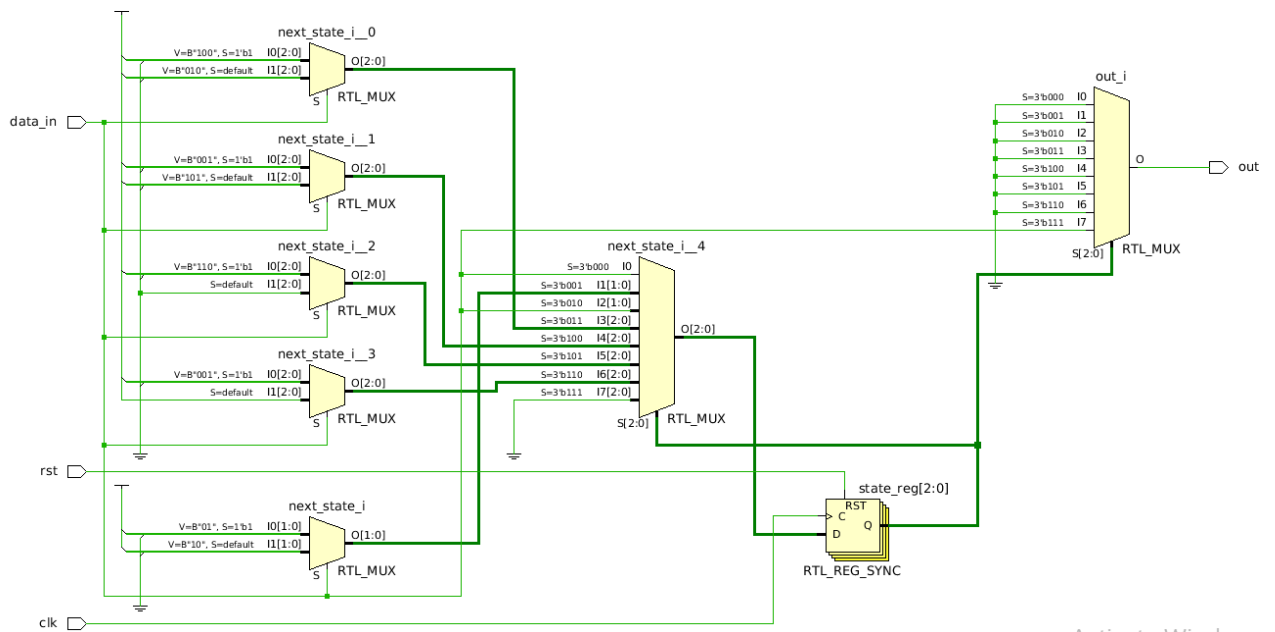
```
41         else
42             next_state = 2'b10;
43
44         end
45
46
47
48     2'b10: begin
49
50         if(taken)
51             next_state = 2'b11;
52
53         else
54             next_state = 2'b10;
55
56         end
57
58
59     2'b11: begin
60
61         if(taken)
62             next_state = 2'b00;
63
64         else
65             next_state = 2'b10;
66
67         end
68
69
70     endcase
71
72
73 end
74
75
76
```

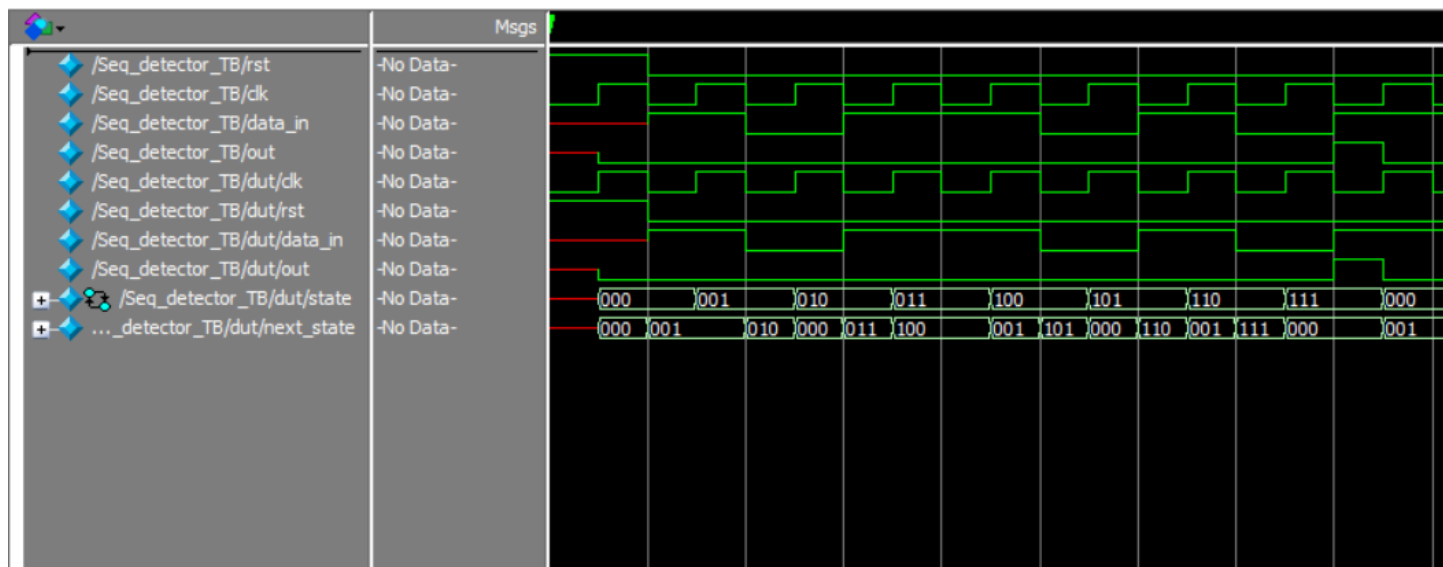
```

77
78     always@(state) begin
79
80         case(state)
81
82             2'b00: predict = 1 ;
83
84             2'b01: predict = 1 ;
85
86             2'b10: predict = 0 ;
87
88             2'b11: predict = 0 ;
89
90
91         endcase
92
93
94     end
95
96
97
98 endmodule
99

```

Q5





```

1  module Seq_detector (input clk,rst,data_in,output reg out);
2
3
4      parameter state_0 = 3'b000;
5      parameter state_1 = 3'b001;
6      parameter state_2 = 3'b010;
7      parameter state_3 = 3'b011;
8      parameter state_4 = 3'b100;
9      parameter state_5 = 3'b101;
10     parameter state_6 = 3'b110;
11     parameter state_7 = 3'b111;
12
13
14     reg[2:0] state,next_state;
15
16
17
18
19     always@(posedge clk) begin
20
21         if(rst)
22             state <= state_0;
23
24         else
25             state <= next_state;
26
27     end
28
29
30
31     always@(state or data_in) begin
32
33
34         case(state)
35
36             state_0: begin
37
38                 if(data_in) begin
39                     next_state <= state_1;
40                     out <= 0;
41                 end
42
43             end
44
45         endcase
46     end
47
48 endmodule

```

```
41         end
42
43     else    begin
44         next_state <= state_0;
45         out <= 0;
46     end
47 end
48
49
50 state_1: begin
51
52     if(data_in) begin
53         next_state <= state_1;
54         out <= 0;
55     end
56
57     else    begin
58         next_state <= state_2;
59         out <= 0;
60     end
61
62 end
63
64 state_2: begin
65
66     if(data_in) begin
67         next_state <= state_3;
68         out <= 0;
69     end
70
71
72     else    begin
73         next_state <= state_0;
74         out <= 0;
75     end
76
77 end
78
79
80 state_3: begin
```

```
82
83         if(data_in) begin
84             next_state <= state_4;
85             out <= 0;
86         end
87
88         else begin
89             next_state <= state_2;
90             out <= 0;
91         end
92
93     end
94
95
96 state_4: begin
97
98         if(data_in) begin
99             next_state <= state_1;
100             out <= 0;
101         end
102
103         else begin
104             next_state <= state_5;
105             out <= 0;
106         end
107
108     end
109
110
111
112 state_5: begin
113
114         if(data_in) begin
115             next_state <= state_6;
116             out <= 0;
117         end
118
119         else begin
120             next_state <= state_0;
121             out <= 0;
```

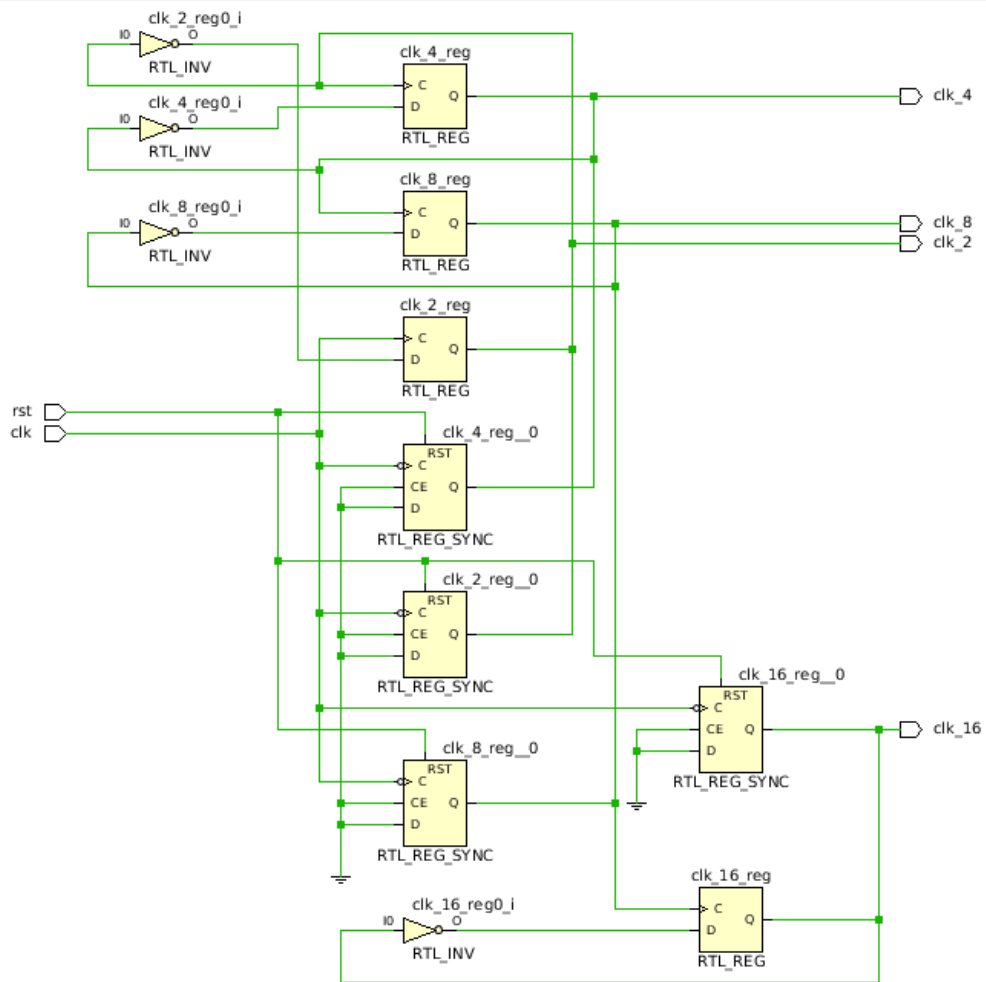
```
122                                     end
123
124             end
125
126
127     state_6: begin
128
129         if(data_in) begin
130             next_state <= state_1;
131             out <= 0;
132             end
133
134         else begin
135             next_state <= state_7;
136             out <= 0;
137             end
138
139     end
140
141
142     state_7: begin
143
144         if(data_in) begin
145             next_state <= state_0;
146             out <= 1;
147             end
148
149         else begin
150             next_state <= state_0;
151             out <= 0;
152             end
153
154     end
155
156
157
158
159 endcase
160
161
```

```
162     end
163
164
165 endmodule
166
167
168
169
170
171 module Seq_detector_TB();
172
173
174     reg rst ,clk,data_in;
175     wire out;
176
177
178     Seq_detector dut(.rst(rst),.clk(clk),.data_in(data_in),.out(out));
179
180
181     initial begin
182         clk = 0;
183         forever begin
184
185             clk = #5 ~clk;
186
187         end
188     end
189 end
190
191
192     initial begin
193
194         rst = 1;
195
196         #10
197
198         rst = 0;
199
```

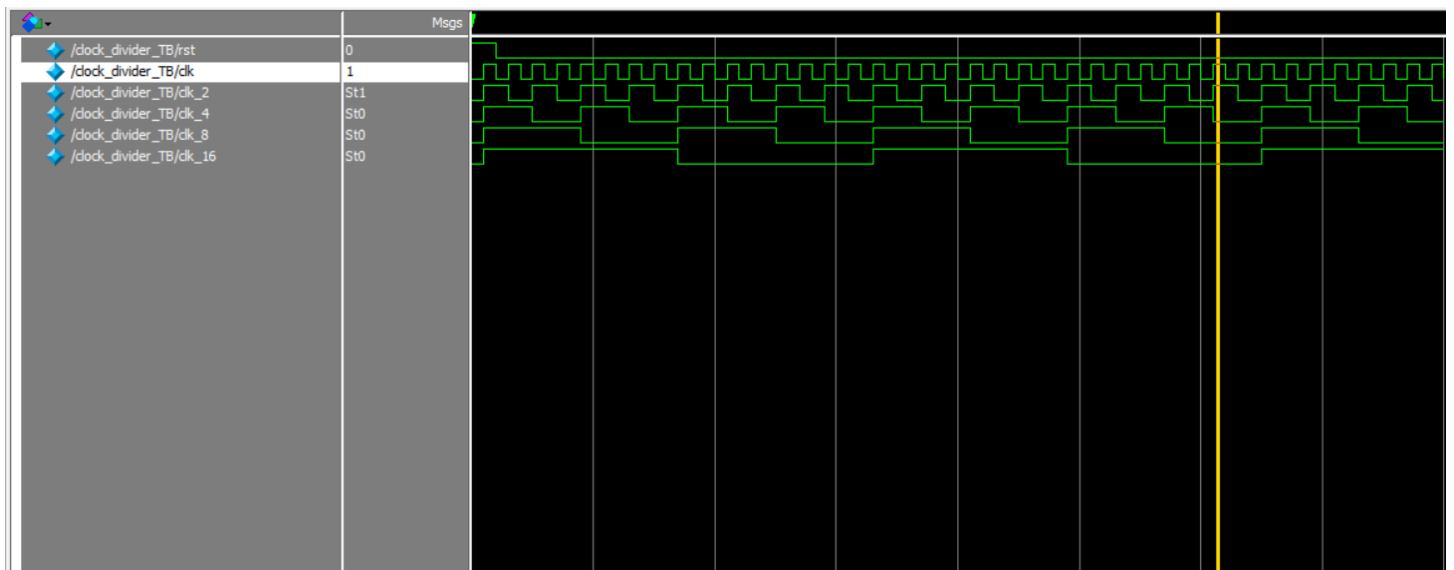


```
203     data_in = 1 ;
204
205     #10
206     data_in = 0 ;
207
208     #10
209     data_in = 1 ;
210
211     #10
212     data_in = 1 ;
213
214     #10
215     data_in = 0 ;
216
217     #10
218     data_in = 1 ;
219
220     #10
221     data_in = 0 ;
222
223     #10
224     data_in = 1 ;
225
226
227
228
229
230     end
231
232
233
234
235     endmodule
236
```

Q6



Activato 1



```

3  module clock_divider_1(input clk,rst,output reg clk_2,clk_4,clk_8,clk_16);
4
5
6
7
8  always@(negedge clk) begin
9
10     if(rst) begin
11         clk_2 <= 0;
12         clk_4 <= 0;
13         clk_8 <= 0;
14         clk_16 <= 0;
15     end
16
17 end
18
19
20
21
22
23 always@(posedge clk) begin
24
25     clk_2 <= ~clk_2;
26
27 end
28
29
30 always@(posedge clk_2) begin
31
32     clk_4 <= ~clk_4;
33
34 end
35
36
37
38 always@(posedge clk_4) begin
39
40     clk_8 <= ~clk_8;
41
42 end
43
44
45
46 always@(posedge clk_8) begin
47
48     clk_16 <= ~clk_16;
49
50 end
51
52
53 endmodule
54
55
56
57
58 module clock_divider_TB();
59
60     reg clk,rst;
61     wire clk_2,clk_4,clk_8,clk_16;
62
63     clock_divider_1 dut(.clk(clk),.rst(rst),.clk_2(clk_2),.clk_4(clk_4),.clk_8(clk_8),.clk_16(clk_16));
64
65     initial begin
66         clk = 0;
67         forever begin
68
69             clk = #5 ~clk;
70
71         end
72
73     end
74
75
76

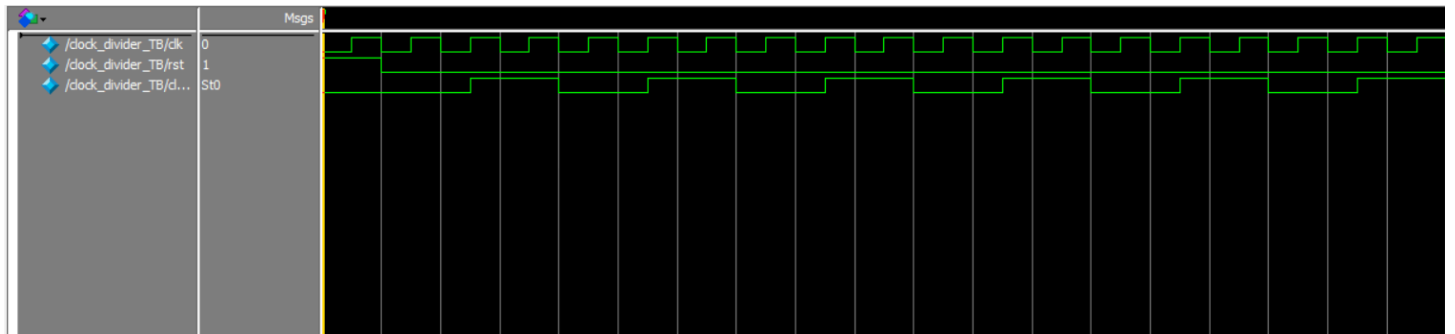
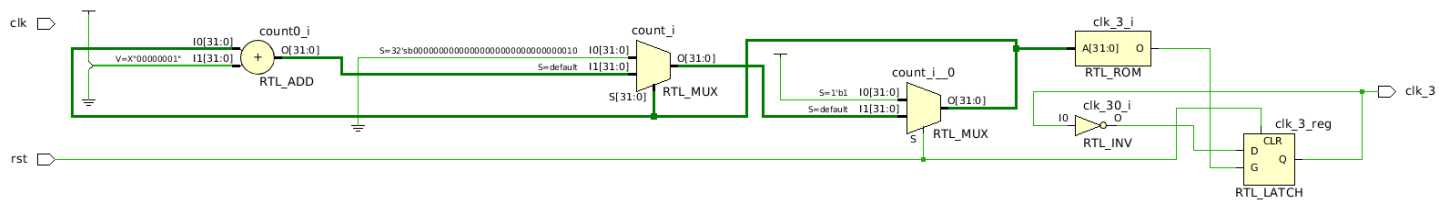
```

```

77     initial begin
78
79         rst = 1;
80
81         #10
82
83         rst = 0;
84
85
86
87
88
89
90     end
91
92 endmodule

```

Q7



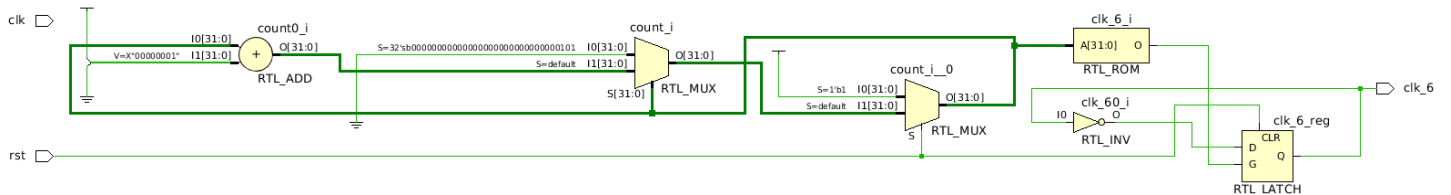
```
1  module clock_divider_2(input clk,rst,output reg clk_3);
2
3
4  integer count;
5
6  always@(clk) begin
7
8      if(rst) begin
9          clk_3 <= 0;
10         count = -1;
11     end
12
13     else if(count == 2)begin
14         clk_3 <= ~clk_3;
15         count = 0;
16     end
17
18     else count = count +1;
19
20 end
21
22
23
24
25
26
27
28 endmodule
29
30
31
```

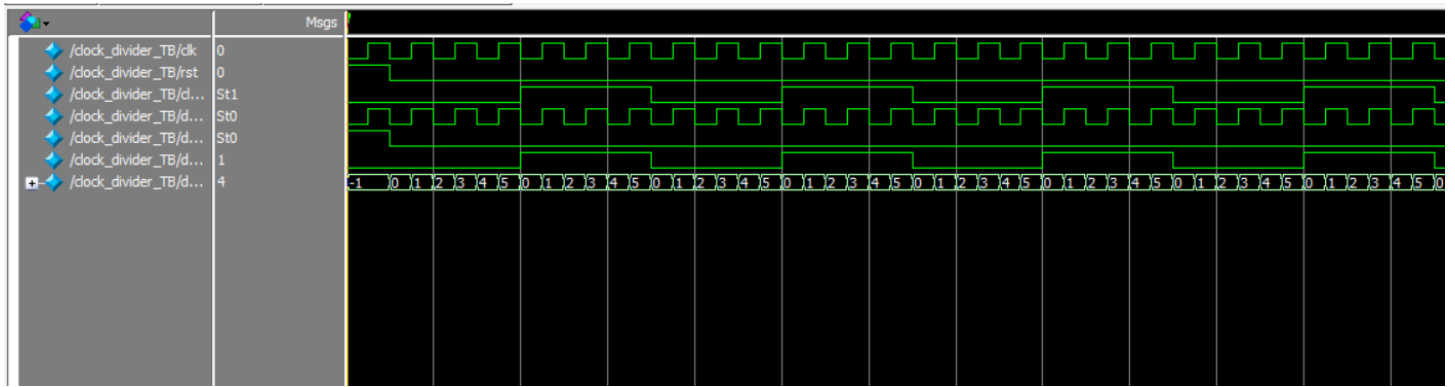
```

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     end
48 end
49
50
51     initial begin
52
53         rst = 1;
54
55         #10
56
57         rst = 0;
58
59
60
61
62
63
64
65     end
66
67 endmodule
68
69

```

Q8





```

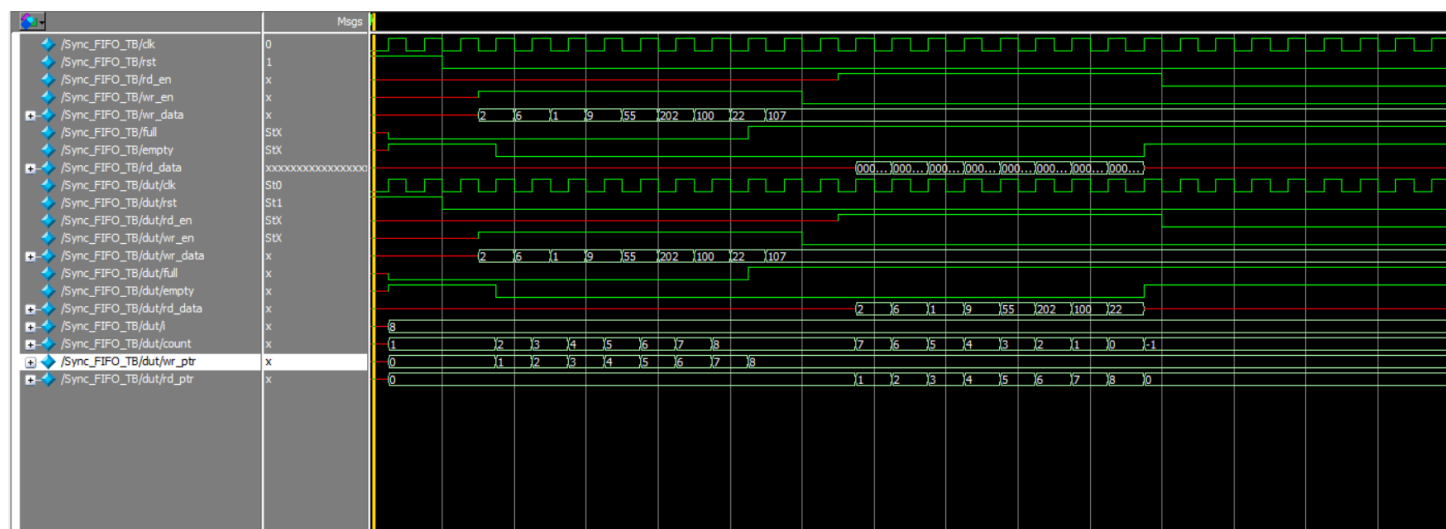
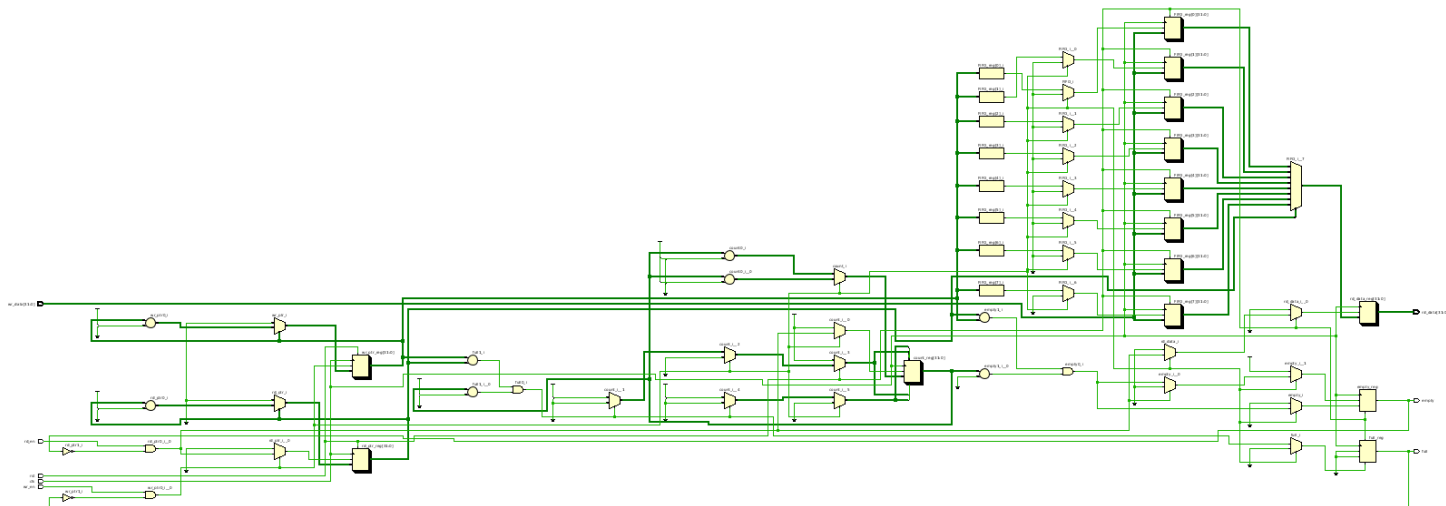
1  module clock_divider_2(input clk,rst,output reg clk_6);
2
3
4      integer count;
5
6      always@(clk) begin
7
8          if(rst) begin
9              clk_6 <= 0;
10             count = -1;
11         end
12
13         else if(count == 5)begin
14             clk_6 <= ~clk_6;
15             count = 0;
16         end
17
18         else count = count +1;
19
20     end
21
22
23
24
25
26
27
28     endmodule
29
30
31
32

```

```

33 module clock_divider_TB();
34
35     reg clk,rst;
36     wire clk_6;
37
38     clock_divider_2 dut(.clk(clk),.rst(rst),.clk_6(clk_6));
39
40     initial begin
41         clk = 0;
42         forever begin
43
44             clk = #5 ~clk;
45
46         end
47     end
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

```

```

1  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);
2
3
4
5
6  reg[31:0] FIFO [0:7];
7  integer i,count;
8  integer wr_ptr,rd_ptr;
9
10
11
12
13  always@(posedge clk) begin
14
15      if(rst) begin
16
17          wr_ptr <= 0;
18          rd_ptr <= 0;
19          empty <= 1;
20          full <= 0;
21          count <= 1;
22          for(i = 0 ; i<8 ; i = i+1)
23              FIFO[i] <= 0;
24
25      end
26
27
28      else if (wr_en & !full) begin
29
30          FIFO[wr_ptr] <= wr_data;
31          wr_ptr <= wr_ptr + 1;
32          empty <= 0;
33          if( wr_ptr == 8) wr_ptr <= wr_ptr % 8;
34
35          count <= count + 1;
36
37          if(wr_ptr > rd_ptr && count==8) begin
38              full <= 1;
39              count <= 8;
40          end
41

```

```

41
42     end
43
44
45     else if (rd_en & !empty) begin
46
47         rd_data <= FIFO[rd_ptr];
48         rd_ptr <= rd_ptr + 1;
49         count <= count - 1;
50         if( rd_ptr == 8 ) rd_ptr <= rd_ptr % 8;
51
52         if(rd_ptr == wr_ptr && count==0)
53             empty <= 1;
54
55     end
56 end
57
58 endmodule
59
60
61
62
63 module Sync_FIFO_TB();
64
65 reg clk,rst,rd_en,wr_en;
66 reg[31:0] wr_data;
67 wire full,empty;
68 wire[31:0] rd_data ;
69
70
71 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));
72
73 initial begin
74     clk = 0;
75     forever begin
76         clk = #5 ~clk;
77     end
78
79 end
80
81
82

```

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