EEE 2243

Digital System Design

TEST 1

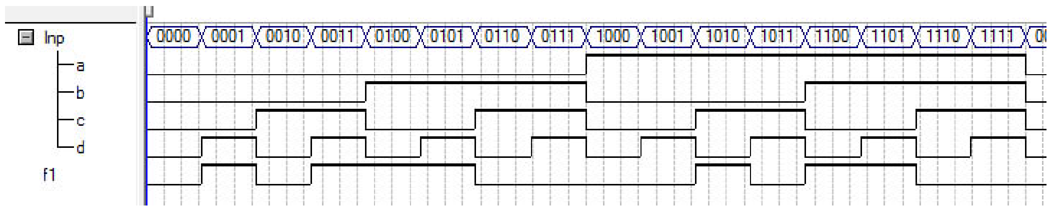
1. a) Write the Verilog code for the following Boolean function WITHOUT minimization using Boolean expression approach: $f = \sum m(1,3,4,5,10,12,13)$

(CO1) [10 marks]

1 ⇨ 0001_2
3 ⇨ 0011_2
4 ⇨ 0100_2
5 ⇨ 0101_2
10 ⇨ 1010_2
12 ⇨ 1100_2
13 ⇨ 1101_2

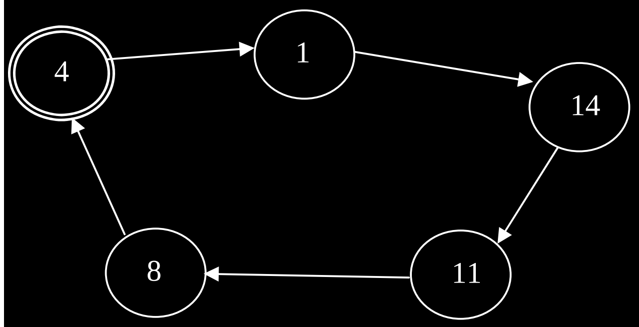
```
module FuncQ1a(a, b, c, d, f1);

input a, b, c, d;
output f1;

assign f1 =   (~a & ~b & ~c &  d)
            | (~a & ~b &  c &  d)
            | (~a &  b & ~c & ~d)
            | (~a &  b & ~c &  d)
            | ( a & ~b &  c & ~d)
            | ( a &  b & ~c & ~d)
            | ( a &  b & ~c &  d);

endmodule
```

Simulation:



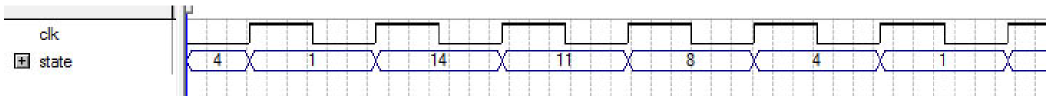
b) Draw the FSM diagram for a 4-bit mod 5 counter starting at 4 and decrement by 3. Then write the Verilog code for it.

(CO1) [15 marks]



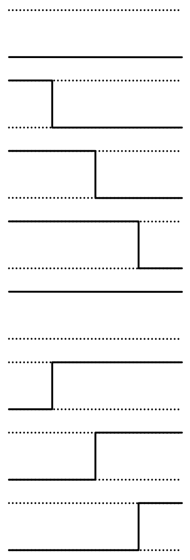
```
module CtrMod5Init4StepNeg3(clk, state);

input clk;
output [3:0] state;
reg [3:0] state;

initial state <= 4;

always@(posedge clk)
begin
   if (state == 8)
      state <= 4;
   else
      state <= state-3;
end

endmodule
```

Simulation:

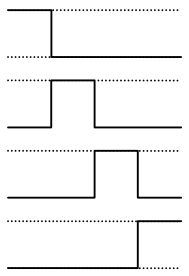


2. Figure Q2 shows the output waveform of 4-bit Johnson and ring counters. Write the Verilog code for them using BOOLEAN expression and BEHAVIORAL approach. (Four code to be written.) [Hint: Draw the logic circuit / diagram first.]

(CO1) [20 marks]



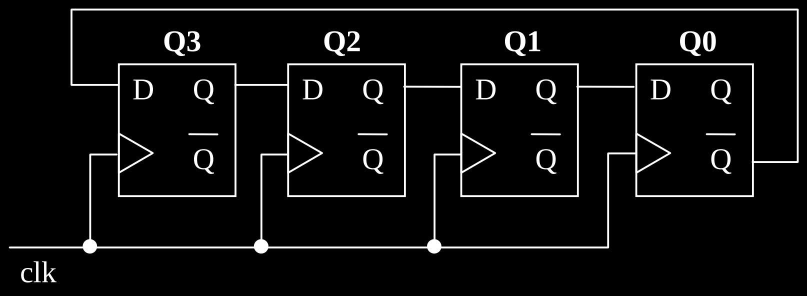
Johnson Counter



Ring Counter

Figure Q2

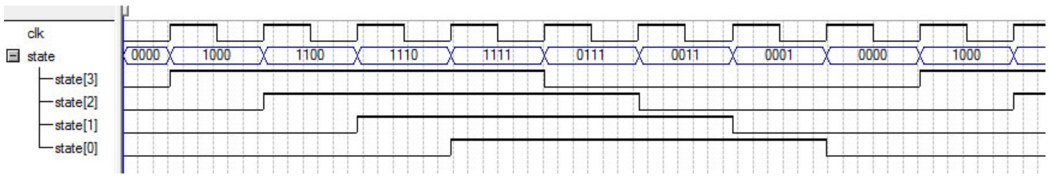
Johnson counter:



Boolean:

```
module JohnsonBoolean(clk, state);

input clk;
output [3:0] state;
reg [3:0] state;

initial state <= 0;

always@(posedge clk)
begin
   state[0] <=  state[1];
   state[1] <=  state[2];
   state[2] <=  state[3];
   state[3] <= ~state[0];
end

endmodule
```

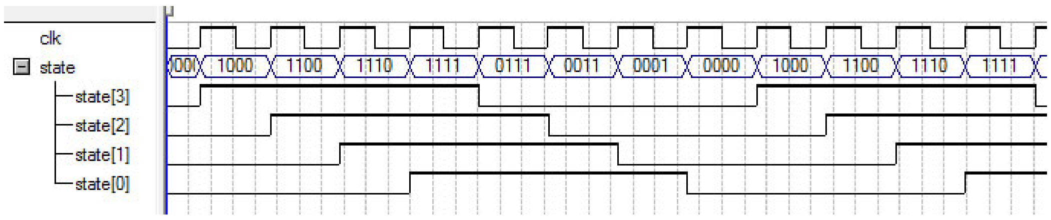
Simulation:



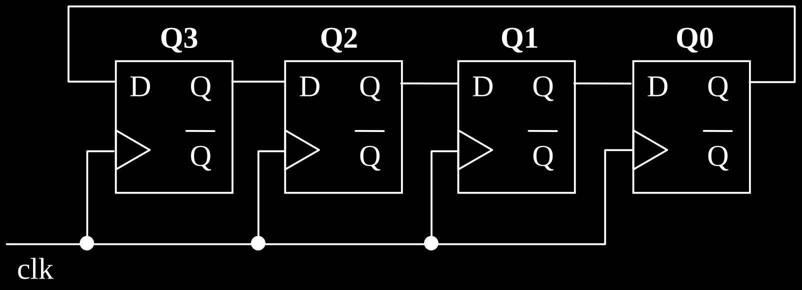
Behavioral:

```
module JohnsonBehavioral(clk, state);

input clk;
output [3:0] state;
reg [3:0] state;

initial state <= 0;

always@(posedge clk)
begin
   case (state)
   4’b0000: state <= 4’b1000;
   4’b1000: state <= 4’b1100;
   4’b1100: state <= 4’b1110;
   4’b1110: state <= 4’b1111;
   4’b1111: state <= 4’b0111;
   4’b0111: state <= 4’b0011;
   4’b0011: state <= 4’b0001;
   4’b0001: state <= 4’b0000;
   endcase
end

endmodule
```

Simulation:



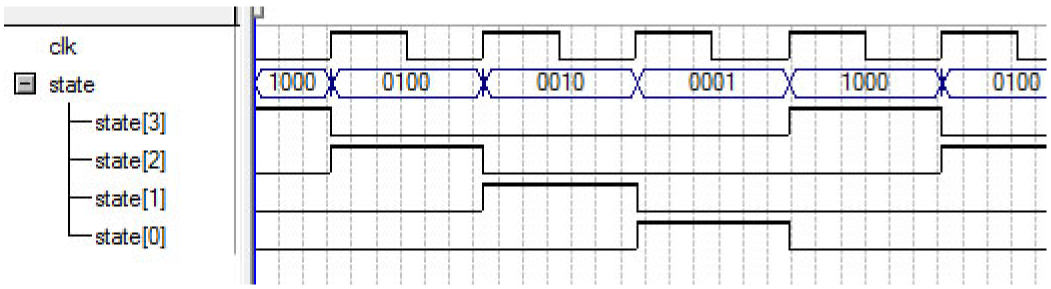
Ring counter:



Boolean:

```
module RingBoolean(clk, state);

input clk;
output [3:0] state;
reg [3:0] state;

initial state <= 4’b1000;

always@(posedge clk)
begin
   state[0] <= state[1];
   state[1] <= state[2];
   state[2] <= state[3];
   state[3] <= state[0];
end

endmodule
```

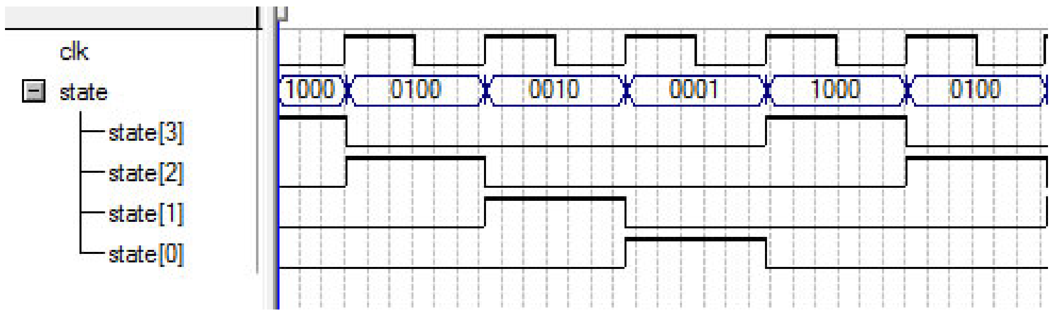
Simulation:



Behavioral:

```
module RingBehavioral(clk, state);

input clk;
output [3:0] state;
reg [3:0] state;

initial state <= 4’b1000;

always@(posedge clk)
begin
   case (state)
   4’b1000: state <= 4’b0100;
   4’b0100: state <= 4’b0010;
   4’b0010: state <= 4’b0001;
   4’b0001: state <= 4’b1000;
   endcase
end

endmodule
```

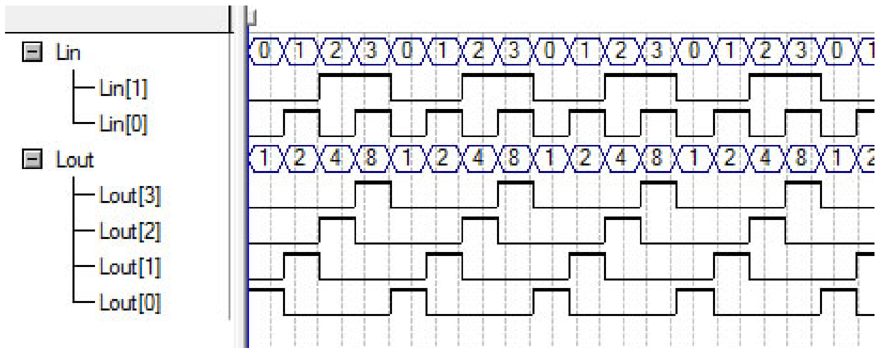
Simulation:



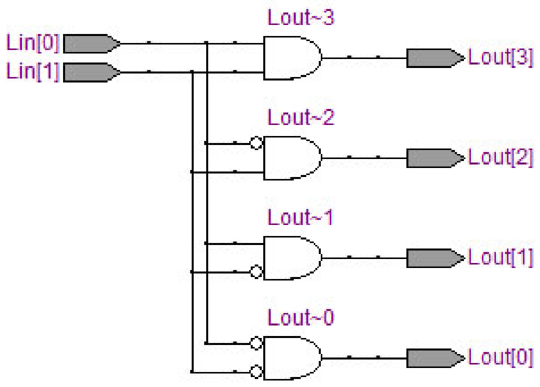
3. a) Write the Verilog code for a 2-to-4 decoder WITHOUT any enable input.
(CO1) [5 marks]

```
module Decoder_2_4(Lin, Lout);

input [1:0] Lin;
output [3:0] Lout;

assign Lout[0] = ~Lin[1] & ~Lin[0];
assign Lout[1] = ~Lin[1] &  Lin[0];
assign Lout[2] =  Lin[1] & ~Lin[0];
assign Lout[3] =  Lin[1] &  Lin[0];

endmodule
```

Simulation:



RTL View:

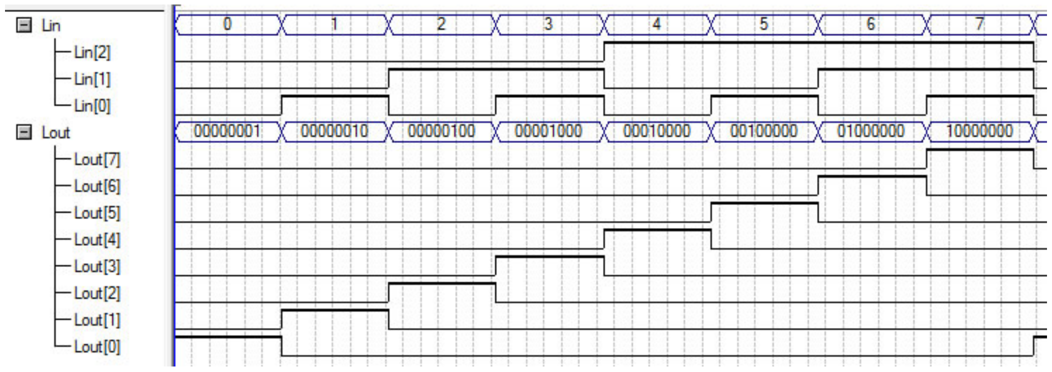


b) Re-use the code in (a) to write a code for 3-to-8 decoder with or without additional components. [Hint: You need to figure out how to put all outputs of one of the decoders to all 0-s when it is not selected.]

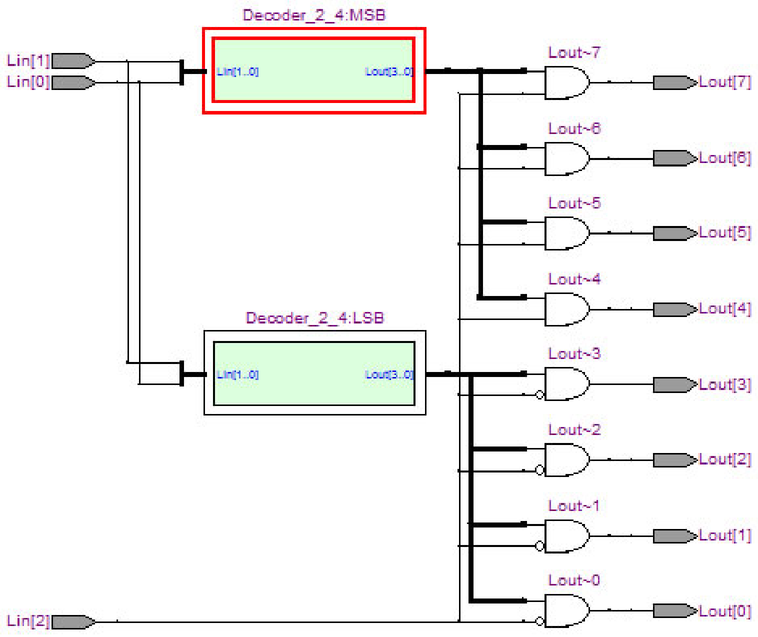
(CO1) [15 marks]

```
module Decoder_3_8(Lin, Lout);

input [2:0] Lin;
output [7:0] Lout;
wire [7:0] w;

Decoder_2_4 MSB(.Lin({Lin[1], Lin[0]}),
                .Lout({w[7], w[6], w[5], w[4]}));

Decoder_2_4 LSB(.Lin({Lin[1], Lin[0]}),
                .Lout({w[3], w[2], w[1], w[0]}));

assign Lout[0] = w[0] & ~Lin[2];
assign Lout[1] = w[1] & ~Lin[2];
assign Lout[2] = w[2] & ~Lin[2];
assign Lout[3] = w[3] & ~Lin[2];
assign Lout[4] = w[4] &  Lin[2];
assign Lout[5] = w[5] &  Lin[2];
assign Lout[6] = w[6] &  Lin[2];
assign Lout[7] = w[7] &  Lin[2];

endmodule
```

Simulation:



RTL View:



4. a) Explain the steps you use to perform bit-by-bit magnitude comparison.

(CO2) [10 marks]

Algorithm to compare A and B starting at MSB:

- If the bit are the same repeat to compare next LSB
- Else if bit A is 1 and bit B is 0 then stop by giving condition A > B
- Else if bit A is 0 and bit B is 1 then stop by giving condition A < B

b) Figure 4(b) shows a block diagram of a single stage of a bit-by-bit magnitude comparator. Explain (with drawing) how you construct a 4-bit magnitude comparator out of it.

(CO2) [10 marks]

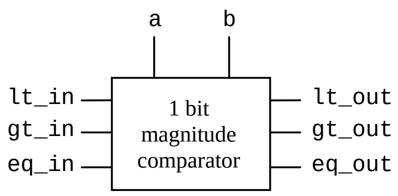
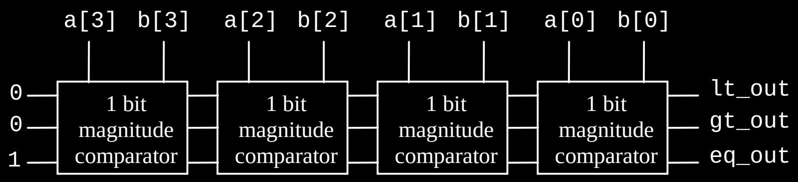


Figure Q4(b)

4 bit comparator is constructed by cascading the 1 bit comparators `lt_out` to `lt_in`, `gt_out` to `gt_in` and `eq_out` to `eq_in` from MSB to LSB. The inputs at MSB is set as: `lt_in` = 0, `gt_in` = 0 and `eq_out` = 1. This way the comparison can propogate to the next bit. The outputs at LSB is the output of the 4 bit comparator.

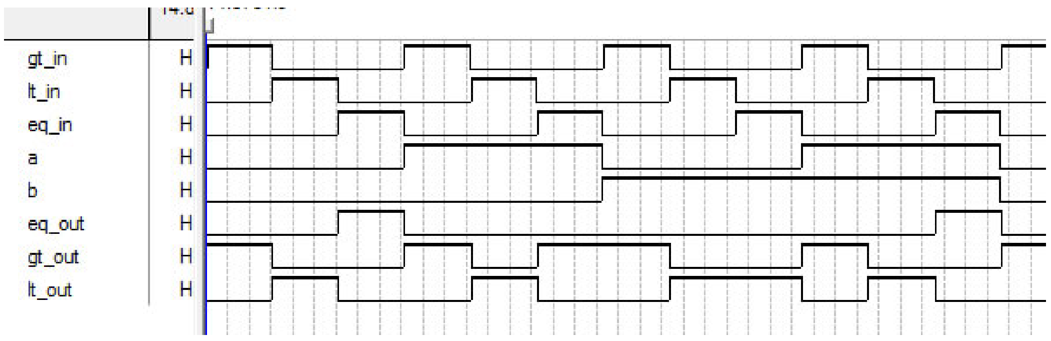

a[3] b[3]
a[2] b[2]
a[1] b[1]
a[0] b[0]
0
0
1
1 bit magnitude comparator
1 bit magnitude comparator
1 bit magnitude comparator
1 bit magnitude comparator
lt_out
gt_out
eq_out

c) Based on the information you have or gave in (a) and (b), write the Verilog code for the 1-bit magnitude comparator in BEHAVIORAL mode.

(CO1) [15 marks]

```
module Comp(lt_in, gt_in, eq_in,
            lt_out, gt_out, eq_out,
            a, b);

input lt_in, gt_in, eq_in, a, b;
output lt_out, gt_out, eq_out;
reg lt_out, gt_out, eq_out;

always@(*)
begin
   if (lt_in)
   begin
      lt_out <= 1;
      gt_out <= 0;
      eq_out <= 0;
   end
   else if (gt_in)
   begin
      lt_out <= 0;
      gt_out <= 1;
      eq_out <= 0;
   end
   else if (eq_in)
   begin
      if (a == b)
      begin
         lt_out <= 0;
         gt_out <= 0;
         eq_out <= 1;
      end
      else if (a < b)
      begin
         lt_out <= 1;
         gt_out <= 0;
         eq_out <= 0;
      end
      else
      begin
         lt_out <= 0;
         gt_out <= 1;
         eq_out <= 0;
      end
   end
end

endmodule
```

Simulation:


gt_in
lt_in
eq_in
a
b
eq_out
gt_out
lt_out
H