*Name: Halwa

*Registration: EasyPizzy-007

Department of Computer Systems Engineering
University of Engineering & Technology Peshawar

Digital System Design
CSE 308

Midterm Examination Spring 2020
3 July 2020, Duration: 150 Minutes

Start Time: 2:30 P.M. (Sharp)
End Time: 5:00 P.M. (Sharp)

****Exam Rules****
Please read carefully before proceeding.
1- This exam is open books/notes/Internet.
2- It's good to share but sharing of books, notes, and other materials during this exam is not permitted.
3- Answer all problems.
4- Problems will not be interpreted during the exam.

Good Luck!

Problem 1. (25 pts.)
Below is an RTL (or Dataflow) description for a circuit.

```
module RTL_circuit (x, y, a, b, c);

     parameter x_delay = 5, y_delay = 10;
     input a, b, c;
     output x, y;
     wire a, b, c, x, y;
     wire na, nb, nc, t3, t4, t5;

     assign na = !a;
     assign nb = !b;
     assign nc = !c;
     assign t3 = na && b && c;
     assign t4 = a && nb && c;
     assign t5 = a && b && nc;
     assign #x_delay x = t3 || t5;
     assign #y_delay y = a || t4;

endmodule
```

1(a) (3 pts.) Give a Verilog statement that instantiates the above RTL_circuit, with the instance name MID. When you instantiate the circuit, use the same names for wires as is used in the module port list.

Solution:

```
RTL_circuit MID (x, y, a, b, c);
```

1(b) (6 pts.) Rewrite the RTL_circuit using Verilog built-in primitives and structural Verilog. Your design should have the same overall delays (from module inputs to module outputs) as the original code. It is fine to let some gates have zero delay. Part of the module is done for you.

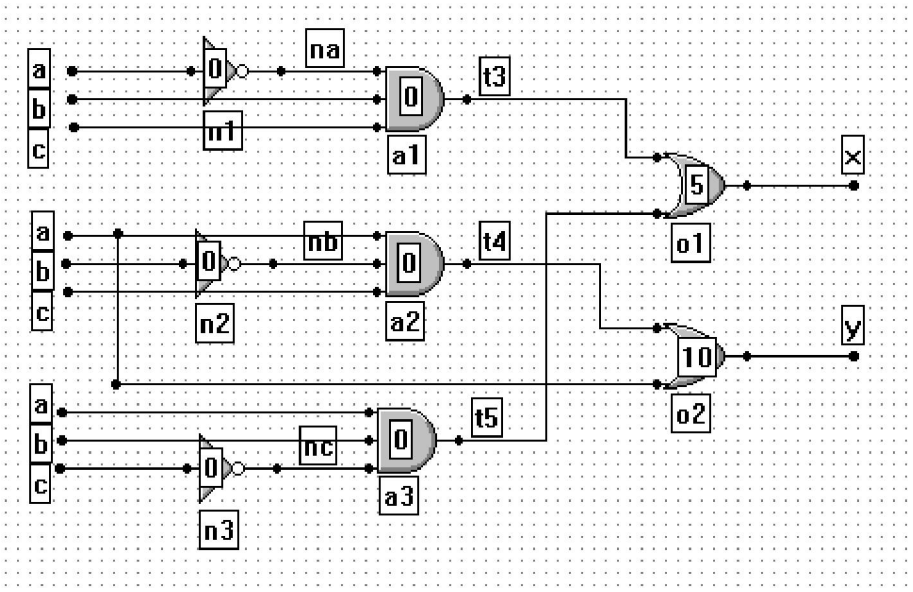
```
module struct_circuit (x, y, a, b, c);

     parameter x_delay = 5, y_delay = 10;
     input a, b, c;
     output x, y;
     //Write your code here




endmodule
```

Solution:

```
module struct_circuit (x, y, a, b, c);

     parameter x_delay = 5, y_delay = 10;
     input a, b, c;
     output x, y;
     //Write your code here

     wire a, b, c, x, y;
     wire na, nb, nc, t3, t4, t5;
     not n1(na, a);
     not n2(nb, b);
     not n3(nc, c);
     and a1(t3, na, b, c);
     and a2(t4, a, nb, c);
     and a3(t5, a, b, nc);
     or #x_delay o1(x, t3, t5);
     or #y_delay o2(y, a, t4);

endmodule
```

1(c) (3 pts.) Draw a gate-level diagram for your module in **1(b)**. Label all nets on the diagram and write the delay of each gate inside the gate.

Solution:



1(d) (6 pts.) Rewrite the RTL_circuit using behavioral Verilog. Your design should have the same delays as the original code. Part of the module is done for you.

```
module behav_circuit (x, y, a, b, c);

      parameter x_delay = 5, y_delay = 10;
      input a, b, c;
      output x, y;
      //Write your code here









endmodule
```

Solution:

```
module behav_circuit (x, y, a, b, c);

      parameter x_delay = 5, y_delay = 10;
      input a, b, c;
      output x, y;
      //Write your code here

      wire a, b, c;
      reg x, y;
      reg t3, t4, t5;
      always @( a or b or c ) t3 <= !a & b & c;
      always @( a or b or c ) t4 <= a & !b & c;
      always @( a or b or c ) t5 <= a & b & !c;
      always @( t3 or t5 ) #x_delay x <= t3 | t5;
      always @( a or t4 ) #y_delay y <= a | t4;

endmodule
```

1(e) (7 pts.) Write the output of RTL_circuit for the following test bench.

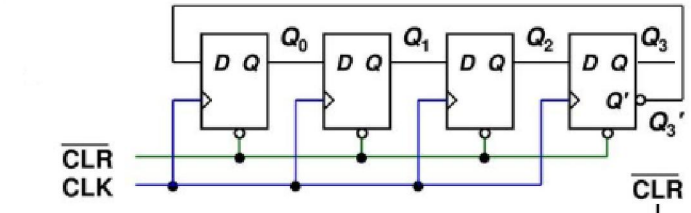
```
//test bench for RTL_circuit
module test_RTL_circuit;

     reg a, b, c;
     wire x, y;

     RTL_circuit RTLC (x, y, a, b, c);

     Initial begin
          a = 1; b = 1; c = 1;
          #30 a = 0; b = 1; c = 1;
     end

     initial begin
          $display ($time, " x = %b and y = %b", x, y);
          #6 $display ($time, " x = %b and y = %b", x, y);
          #5 $display ($time, " x = %b and y = %b", x, y);
          #20 $display ($time, " x = %b and y = %b", x, y);
          #5 $display ($time, " x = %b and y = %b", x, y);
          #6 $display ($time, " x = %b and y = %b", x, y);
          #8 $display ($time, " x = %b and y = %b", x, y);
     end

endmodule
```

Solution:

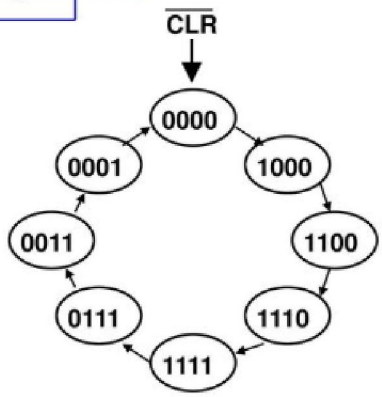
```
0 x = x and y = x
6 x = 0 and y = x
11 x = 0 and y = 1
31 x = 0 and y = 1
36 x = 1 and y = 1
42 x = 1 and y = 0
50 x = 1 and y = 0
```

Problem 2. (30 pts.)

2(a) (10 pts.) In this problem, design a 4-bit (Mod-8) Johnson counter (the circuit and state diagrams are given below).



CLK	Q_0	Q_1	Q_2	Q_3
→0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1



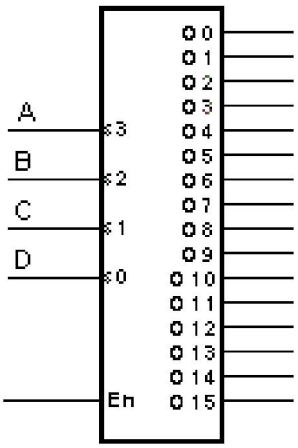
The suggested skeleton file has been written below. The module has 2 inputs – CLK and CLR which is active low. The output is Q which is 4-bit in size.

```
module JS_counter (CLK, CLR, Q);

      input CLK, CLR;
      output [3:0] Q;
      //Place your code here



endmodule
```

Solution:

```
module JS_counter (CLK, CLR, Q);

     input CLK, CLR;
     output [3:0] Q;
     //Place your code here

     reg [3:0] Q;
     parameter S0 = 4'b0000,S1 = 4'b1000,S2 = 4'b1100,S3 = 4'b1110,
               S4 = 4'b1111,S5 = 4'b0111,S6 = 4'b0011,S7 = 4'b0001;

     always @ (posedge CLK)
          if (~CLR)
               Q <= S0;
          else
               case (Q)
                    S0: Q <= S1;
                    S1: Q <= S2;
                    S2: Q <= S3;
                    S3: Q <= S4;
                    S4: Q <= S5;
                    S5: Q <= S6;
                    S6: Q <= S7;
                    S7: Q <= S0;
                    default: Q <= S0;
               endcase

endmodule
```

2(b) (10 pts.) In this problem, design a 4-to-16 Decoder with Enable input (the block diagram is given below).



The suggested skeleton file has been written below. The module has 5 inputs – A, B, C, D and En which is active high. The output is O which is 16-bit in size.

```
module Dec_4x16 (A, B, C, D, En, O);

     input A, B, C, D, En; //Assume A is MSB and D is LSB
     output [15:0] O;
     //Place your code here



endmodule
```

Solution:

```
module Dec_4x16 (A, B, C, D, En, O);

     input A, B, C, D, En; //Assume A is MSB and D is LSB
     output [15:0] O;
     //Place your code here

     reg [15:0] O;
     always @ (A or B or C or D or En)
          if (~En)
               O = 16'b0;
          else
               case ({A, B, C, D})
                    0: O = 16'h0001;
                    1: O = 16'h0002;
                    2: O = 16'h0004;
                    3: O = 16'h0008;
                    4: O = 16'h0010;
                    5: O = 16'h0020;
                    6: O = 16'h0040;
                    7: O = 16'h0080;
                    8: O = 16'h0100;
                    9: O = 16'h0200;
                    10: O = 16'h0400;
                    11: O = 16'h0800;
                    12: O = 16'h1000;
                    13: O = 16'h2000;
                    14: O = 16'h4000;
                    15: O = 16'h8000;
               endcase

endmodule
```

2(c) (10 pts.) In this problem, combine the Johnson counter (from **2(a)**) with the decoder (from **2(b)**) to create a circuit such that the output of the Johnson counter controls the data lines of the decoder. Assume that the decoder is enabled (i.e. the En input is tied to VCC or 1).

The top-level design has the following port definitions:

CLK	1-bit clock
RESET	1-bit reset line
OUT_DATA	16-bit data output

The suggested skeleton file has been written below.

```
module Top (CLK, RESET, OUT_DATA);

      input CLK, RESET;
      output [15:0] OUT_DATA;
      //Place your code here




endmodule
```

Solution:

```
module Top (CLK, RESET, OUT_DATA);

      input CLK, RESET;
      output [15:0] OUT_DATA;
      //Place your code here

      wire [3:0] Q;

      JS_counter js1 (CLK, RESET, Q);
      Dec_4x16 dec1 (Q[3], Q[2], Q[1], Q[0], 1'b1, OUT_DATA);

endmodule
```