***Name:** ______________________________

***Registration:** ______________________________

Department of Computer Systems Engineering
University of Engineering & Technology Peshawar

Digital System Design
CSE 308

Midterm Examination Spring 2024

4 April 2024, Duration: 120 Minutes

****Exam Rules****

Please read carefully before proceeding.

1- This exam is closed books/notes/Internet.
2- Answer all problems on the answer sheet.
3- Problems will not be interpreted during the exam.

Good Luck!

Problem 1. (25 pts.)

Below is a dataflow description for a circuit.

```
module DATAFLOW_CIRCUIT (a, b, c, en, d);

      input a, b, c, en;
      output [0:7] d;

      wire na, nb, nc;

      assign na = !a;
      assign nb = !b;
      assign nc = !c;
      assign d[0] = na && nb && nc && en;
      assign d[1] = na && nb && c && en;
      assign d[2] = na && b && nc && en;
      assign d[3] = na && b && c && en;
      assign d[4] = a && nb && nc && en;
      assign d[5] = a && nb && c && en;
      assign d[6] = a && b && nc && en;
      assign d[7] = a && b && c && en;

endmodule
```

(a) (3 pts., CLO-1)

Give a Verilog statement that instantiates the above DATAFLOW_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.

```
DATAFLOW_CIRCUIT MID (a, b, c, en, d);
```

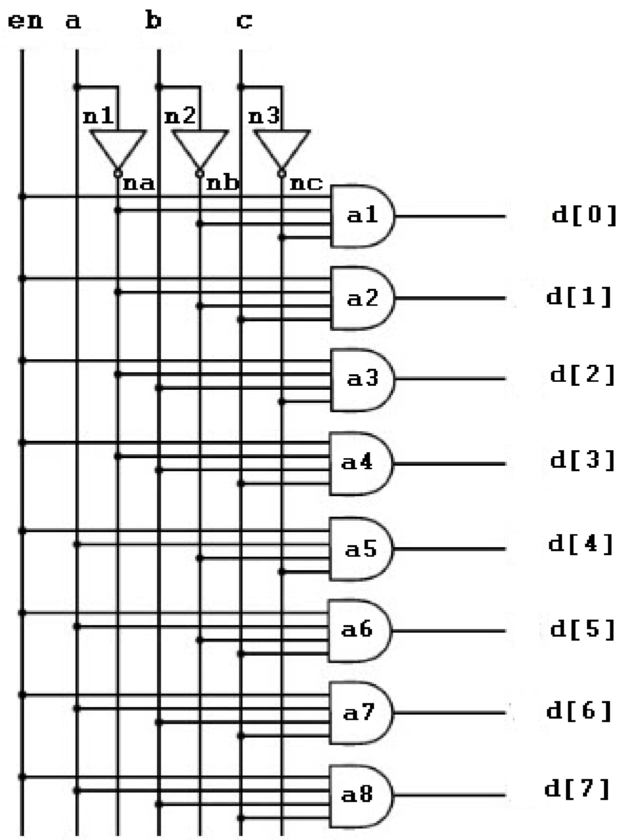
(b) (6 pts., CLO-1)

Rewrite the DATAFLOW_CIRCUIT using Verilog built-in primitives and structural Verilog. Part of the module is done for you.

```
module STRUCT_CIRCUIT (a, b, c, en, d);

      input a, b, c, en;
      output [0:7] d;
      //Write your code here

      wire na, nb, nc;

      not n1 (na, a);
      not n2 (nb, b);
      not n3 (nc, c);
      and a1 (d[0], na, nb, nc, en);
      and a2 (d[1], na, nb, c, en);
      and a3 (d[2], na, b, nc, en);
      and a4 (d[3], na, b, c, en);
      and a5 (d[4], a, nb, nc, en);
      and a6 (d[5], a, nb, c, en);
      and a7 (d[6], a, b, nc, en);
      and a8 (d[7], a, b, c, en);

endmodule
```

(c) (3 pts., CLO-1)

Draw a gate-level circuit for your module in **(b)**. Label all nets on the circuit.



(d) (6 pts., CLO-1)

Rewrite the DATAFLOW_CIRCUIT using behavioral Verilog. Part of the module is done for you.

```
module BEHAV_CIRCUIT (a, b, c, en, d);

      input a, b, c, en;
      output [0:7] d;
      //Write your code here
      reg [0:7] d;
      reg na, nb, nc;

      always @(*)
      begin
            na = !a;
            nb = !b;
            nc = !c;
            d[0] = na && nb && nc && en;
            d[1] = na && nb && c && en;
            d[2] = na && b && nc && en;
            d[3] = na && b && c && en;
            d[4] = a && nb && nc && en;
            d[5] = a && nb && c && en;
            d[6] = a && b && nc && en;
            d[7] = a && b && c && en;
      end

endmodule
```

(e) (7 pts., CLO-1)

Write the output of DATAFLOW_CIRCUIT for the following test bench.

```
//Test Bench for DATAFLOW_CIRCUIT
module TB_DATAFLOW_CIRCUIT;

      reg a, b, c, en;
      wire [0:7] d;
      DATAFLOW_CIRCUIT DF (a, b, c, en, d);

       initial
       begin
            a = 1; b = 1; c = 1; en = 0;
            #5 a = 0; b = 1; c = 1; en = 1;
            #5 a = 1; b = 0; c = 1; en = 1;
            #5 a = 0; b = 1; c = 0; en = 1;
            #5 a = 1; b = 1; c = 0; en = 0;
            #5 a = 0; b = 0; c = 1; en = 1;
            #5 a = 0; b = 0; c = 0; en = 1;
       end

       initial
            $monitor ($time, " OUT = %b", d);

endmodule
```

```
0 OUT = 00000000
5 OUT = 00010000
10 OUT = 00000100
15 OUT = 00100000
20 OUT = 00000000
25 OUT = 01000000
30 OUT = 10000000
```

Problem 2. (40 pts.)

This problem involves creating several Verilog designs. Three lower-level modules are written and then two of these are combined to create a top-level design.

For each module, write a complete Verilog description with correct declaration, etc. A completely correct solution will receive the number of points indicated in parentheses. Partially correct solutions will receive partial credit.

(a) Module 1: 3-bit Up/Down Counter. (10 pts., CLO-2)

In this module, design a 3-bit up/down counter, a counter that counts in both up and down directions. The state transition diagram is shown in **Figure 1**.

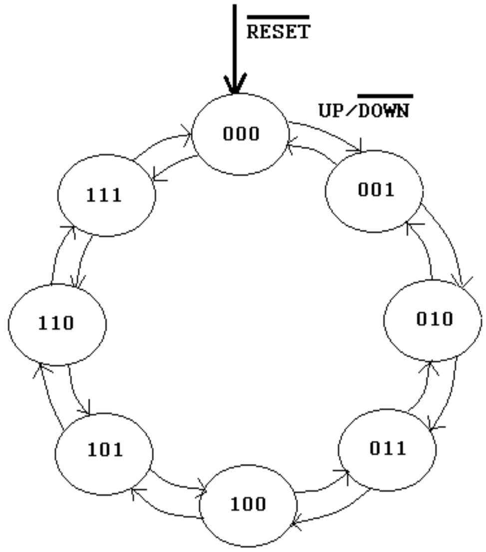


Figure 1. The state transition diagram of the 3-bit up/down counter

The following are the ports of the module:

CLK	1-bit clock input, all actions must be on the rising edge
RESET	1-bit reset input, causes reset on the rising edge
UP_DN	1-bit input (if 1, then count up, if 0, then count down)
COUNT	3-bit output

The skeleton file for the counter has been written below.

```
module COUNTER (CLK, RESET, UP_DN, COUNT);
      //Write your code here

      input CLK, RESET, UP_DN;
      output [2:0] COUNT;

      reg [2:0] COUNT;

      always @(posedge CLK)
            if (~RESET) //Active low RESET
                  COUNT <= 3'b000;
            else
                  if (UP_DN) //Up mode selected
                        COUNT <= COUNT + 1'b1; //Increment COUNT
                  else //Down mode selected
                        COUNT <= COUNT - 1'b1; //Decrement COUNT


endmodule
```

(b) Module 2: 8-to-1 Multiplexer. (10 pts., CLO-1)
In this module, design a byte-wide 8-to-1 multiplexer. In this case, the value on the 3-bit select line will route 1 of 8 inputs to the output. This module is purely combinatorial.

The following are the ports of the module:

SEL	3-bit select line
D0, D1, D2, D3, D4, D5, D6, and D7	8-bit data inputs
OUT	8-bit output

The skeleton file for the multiplexer has been written below.

```
module MUX81 (SEL, D0, D1, D2, D3, D4, D5, D6, D7, OUT);
      //Write your code here

      input [2:0] SEL;
      input [7:0] D0, D1, D2, D3, D4, D5, D6, D7;
      output [7:0] OUT;

      reg [7:0] OUT;

      always @(*)
            case (SEL)
                  3'b000: OUT = D0;
                  3'b001: OUT = D1;
                  3'b010: OUT = D2;
                  3'b011: OUT = D3;
                  3'b100: OUT = D4;
                  3'b101: OUT = D5;
                  3'b110: OUT = D6;
                  3'b111: OUT = D7;
            endcase


endmodule
```

(c) Module 3: Parallel-in, Serial-out Shift Register. (10 pts., CLO-2)
In this module, create a register that loads data in parallel but shifts data out serially, MSB first.

The following are the ports of the module:

CLK	1-bit clock, all operations must be on the falling edge
PI	8-bit parallel data input
SO	1-bit serial output
LD	1-bit input, when high, PI is loaded into the shift register
SHIFT	1-bit shift enable input, when high, contents of the shift register are shifted out on to the serial output SO

The skeleton file for the register has been written below.

```
module PISO (CLK, PI, SO, LD, SHIFT);
      //Write your code here

      input CLK, LD, SHIFT;
      input [7:0] PI;
      output SO;

      reg SO;
      reg [7:0] SHIFTER;

      always @(negedge CLK)
            if (LD)
                  SHIFTER = PI;
            else if (SHIFT)
                  begin
                        SO = SHIFTER[7];
                        SHIFTER = {SHIFTER[6:0], 1'b0};
                  end

endmodule
```

(d) Module 4: Top-level Design. (10 pts., CLO-2)

For this design, combine the 3-bit up/down counter (**Module 1**) with the 8-to-1 multiplexer (**Module 2**) to create a circuit such that the output of the counter controls the select lines of the multiplexer.

This top-level design has the following port definitions:

CLOCK	1-bit clock
CLR	1-bit reset line
UD	1-bit input (if 1, the counter counts up, if 0, the counter counts down)
I0, I1, I2, I3, I4, I5, I6, and I7	8-bit data inputs
DATA_OUT	8-bit data output

The skeleton file for the top-level design has been written below.

```
module TOP (CLOCK, CLR, UD, I0, I1, I2, I3, I4, I5, I6, I7, DATA_OUT);
      //Write your code here

      input CLOCK, CLR, UD;
      input [7:0] I0, I1, I2, I3, I4, I5, I6, I7;
      output [7:0] DATA_OUT;

      wire [2:0] COUNT;

      COUNTER c (CLOCK, CLR, UD, COUNT);
      MUX81 m (COUNT, I0, I1, I2, I3, I4, I5, I6, I7, DATA_OUT);

endmodule
```