|CSE 308: Digital System Design|Homework 4 Solution|

Problem 1: Synthesis.
Draw the synthesized circuits described by the Verilog codes below.

(a)

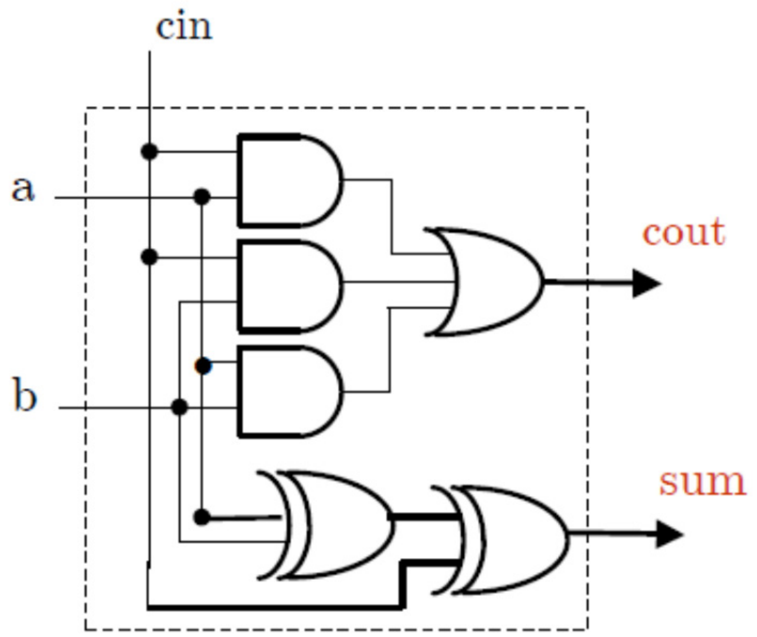
```
module ADD (sum, cout, a, b, cin);

    input a, b, cin;
    output sum, cout;
    reg sum, cout; //registers retain value

    always @ (a or b or cin)
    //anytime a or b or cin CHANGE, run the process
        begin
            sum <= a ^ b ^ cin;
            cout <= (a & b) | (a & cin) | (b & cin);
        end

endmodule
```

Solution:



(b)

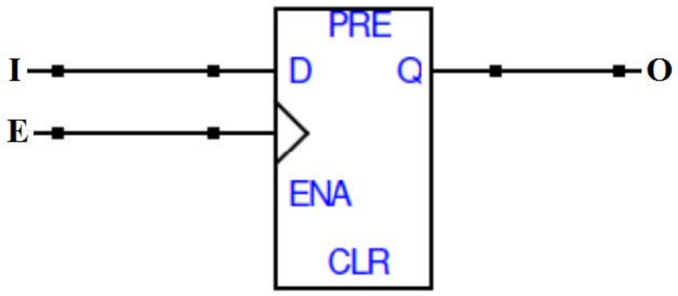
```
module FF (I, E, O);

        input I, E;
        output O;
        reg O;

        always @(posedge E)
                O = I;

endmodule
```

Solution:



(c)

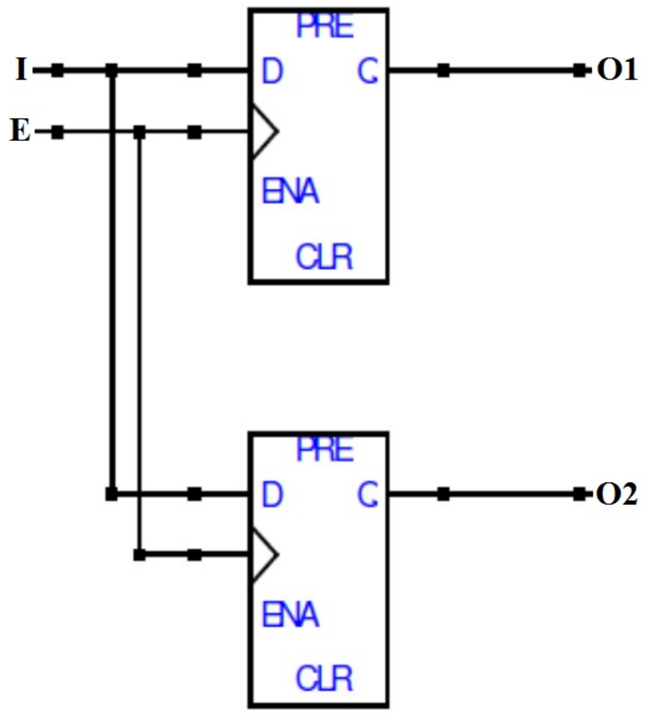
```
module FF_blocking (I, E, O1, O2);

        input   I,    E;
        output O1, O2;
        reg O1, O2;

        always @(posedge E)
                begin
                        //blocking assignment – series execution
                        O1 = I;
                        O2 = O1;
                end

endmodule
```

Solution:



(d)

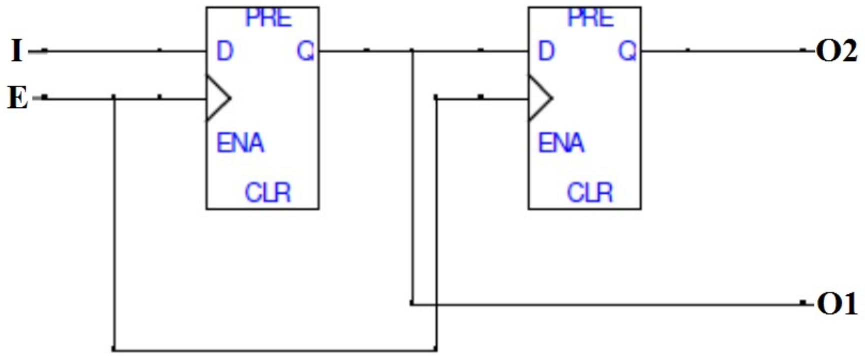
```
module FF_non_blocking (I, E, O1, O2);

    input    I,    E;
    output   O1,   2;
    reg O1, O2;

    always @(posedge E)
        begin
            //non-blocking assignment - can be done in
            //parallel (or any order)
            O1 <= I;
            O2 <= O1;
        end

endmodule
```

Solution:



(e)

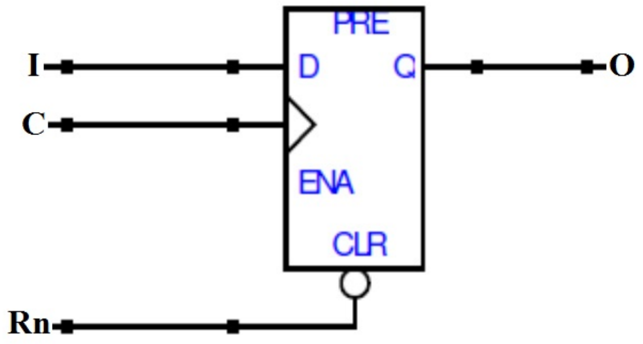
```
module FF_reset (I, C, Rn, O);

        input I, C, Rn;
        output O;
        reg O;

        always @(negedge Rn or posedge C)
                if (!Rn)
                        O <= 0;
                else
                        O <= I;

endmodule
```

Solution:



Problem 2: Clock Generation.

Complete the following clk_gen module, which generates a clock signal that initially goes to zero for 10 ns, then goes to one for 3 ns, and then repeats this pattern indefinitely. Your module can only use one initial statement.

```
module clk_gen;

    reg clk;

    initial
        begin
        // complete code here





        end

endmodule
```

Solution:

```
`timescale 1ns / 1ns
 module clk_gen;
       reg clk;

       initial
       begin
       clk = 1'b0;
       forever
                 begin
                        #10 clk = ~ clk;
                        #3  clk = ~ clk;
                 end
       end
 endmodule
```

Problem 3: Serial-in, Serial-out Left-shift Register.

In this module, create an 8-bit shift-left register with a positive-edge clock, asynchronous clear, serial in, and serial out.

The following are the ports of the module:

CLK	1-bit Positive-Edge Clock
SI	1-bit Serial Input
SO	1-bit Serial Output
CLR	1-bit Asynchronous Clear (active High)
SHIFT	1-bit shift enable input, when high, contents of the shift register are shifted out on to the serial output SO

Solution:

```
module PISO_SHIFT (
                   CLK,
                   CLR,
                   SHIFT,
                   SI,
                   SO);
     input CLK;
     input CLR;
     input SHIFT;
     input SI;
     output SO;

     reg SO;
     reg [7:0] SHIFTER;

     always @ (posedge CLK, CLR)
          if (CLR)
               SHIFTER = 0;
          else if (SHIFT)
               begin
                    SO = SHIFTER[7];
                    SHIFTER = {SHIFTER[6:0], SI};
               end
endmodule
```

Problem 4: Up-Down, Loadable Counter.

In this module, create a counter that counts in both the up and down directions. The preset is to be set to decimal value 3. That is, upon asserting the reset signal RESET low, the register value should be reset to 3. Also, upon asserting the load signal LD high, the register value should be set to the input value DIN. Both the reset and the load are synchronous and the module should count on the falling edge of the clock.

The following are the ports of the module:

CLK	1-bit Positive-Edge Clock
RESET	1-bit preset (asynchronous)
UP_DN	1-bit input (if '1', then count down, if '0', then count up)
LD	1-bit load enable input, loads not synchronized with CLK rising edge
DIN	3-bit input data for loading counter value
Q	3-bit result

Solution:

```
module UP_DN_LD_CNT (
                     CLK,
                     RESET,
                    UP_DN,
                     LD,
                     DIN,
                     Q);
      input CLK;
      input RESET;
      input UP_DN;
      input LD;
      input [2:0] DIN;
      output [2:0] Q;

      reg [2:0] Q;

      always @ (posedge CLK, RESET, LD)
            if (~RESET)
                  Q = 3'b011;
            else
                  if (LD)
                        Q = DIN;
                  else if (UP_DN)
                        Q = Q - 1;
                  else
                        Q = Q + 1;
endmodule
```

Problem 5: State Diagrams in Verilog.

You are a Hardware Design Engineer working for DCSE (suppose only!). They want you to design a circuit that will automate their peon Peerzada's movement on the UET campus. The HoD wirelessly transmits the travel plans to Peer, and then Peer moves according to that information.

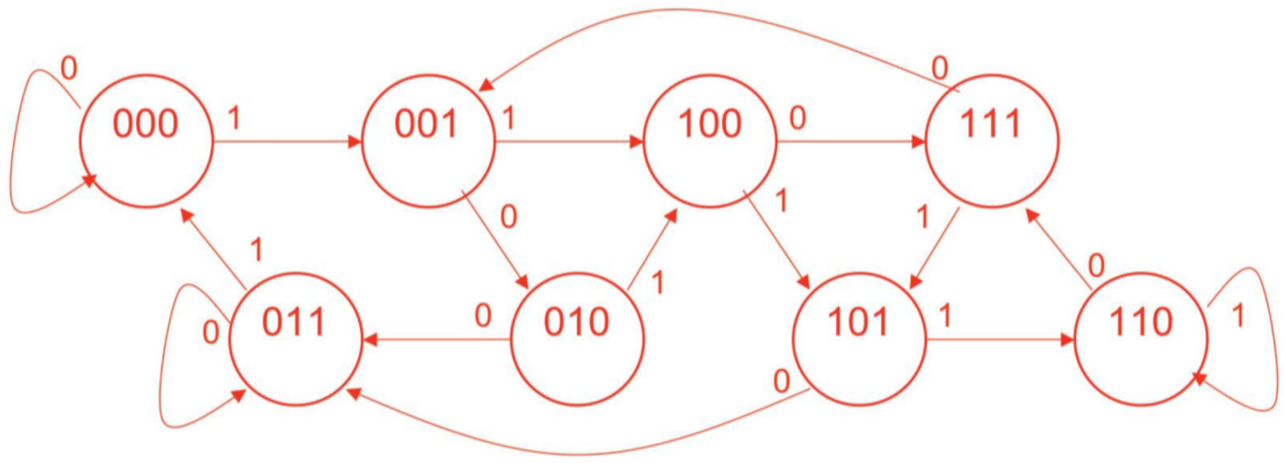
To design your circuit, you first select the following locations around the UET campus and assign each location with a state in 3-bit binary representation: HoD's office[000], Lab-1[001], Lab-2[010], Lab-3[011], Main office[100], Dean's office[101], Registrar's office[110], and the VC's office[111].

To simplify your design, you inform the HoD to send Peer a binary sequence for travel plans (e.g. '1-0-0-0-1' to cause Peer to move five times). In other words, Peer receives either '0' or '1' for each move and travels to the next destination as specified below. Peer starts off at HoD's office, and your circuit should output Peer's current location.

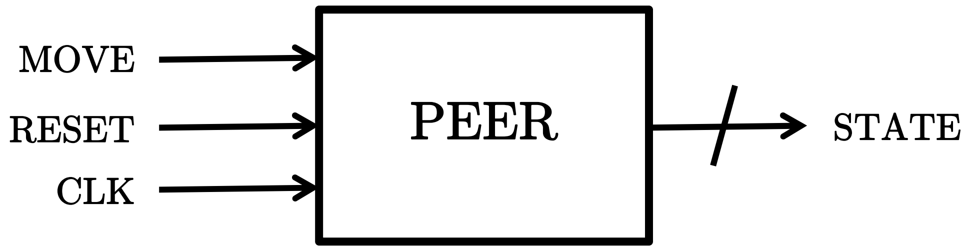
Current location	**Next location**	
HoD's office[000]:	If 0, stay at HoD's office.	If 1, go to Lab-1.
Lab-1[001]:	If 0, go to Lab-2.	If 1, go to Main office.
Lab-2[010]:	If 0, go to Lab-3.	If 1, go to Main office.
Lab-3[011]:	If 0, stay at Lab-3.	If 1, go to HoD's office. (Laiq sb. chai ghwardi!)
Main office[100]:	If 0, go to VC's office.	If 1, go to Dean's office.
Dean's office[101]:	If 0, go to Lab-3.	If 1, go to Registrar's office.
Registrar's office[110]:	If 0, go to VC's office.	If 1, stay at Registrar's office.
VC's office[111]:	If 0, go to Lab-1.	If 1, go to Dean's office.

(a) Draw the state transition diagram for Peer. Please use a scratch page for scratch work and make a neat copy of your final diagram here.

Solution:



(b) Design a module in Verilog for Peer – the following schematic shows the appropriate inputs and outputs. Please use a scratch page as scratch space and write a neat copy of the final code here.



Solution:

```
module PEER (
             CLK,
             RESET,
             MOVE,
             STATE);

     input CLK, RESET, MOVE;
     output reg [2:0] STATE;

     always @ (posedge CLK)
          begin
               if (RESET)
                    STATE <= 3'b000;
               else
                    case (STATE)
                         3'b000: STATE <= MOVE ? 3'b001 : 3'b000;
                         3'b001: STATE <= MOVE ? 3'b100 : 3'b010;
                         3'b010: STATE <= MOVE ? 3'b100 : 3'b011;
                         3'b011: STATE <= MOVE ? 3'b000 : 3'b011;
                         3'b100: STATE <= MOVE ? 3'b101 : 3'b111;
                         3'b101: STATE <= MOVE ? 3'b110 : 3'b011;
                         3'b110: STATE <= MOVE ? 3'b110 : 3'b111;
                         3'b111: STATE <= MOVE ? 3'b101 : 3'b001;
                         default: STATE <= 3'b000;
                    endcase
          end
endmodule
```