Engr 378 HW #5 2.36, 2.37,2.43, 2.44, 2.48, 2.50

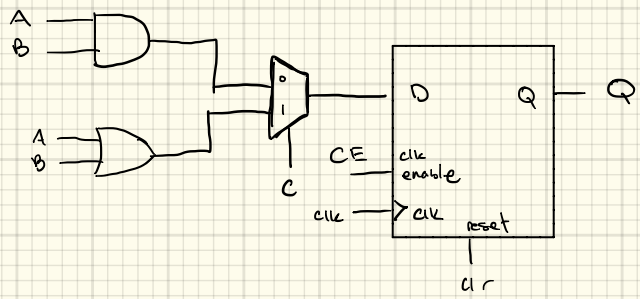
Engineering 378

Farnam Adelkhani
915815724

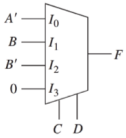
2.36 Draw the circuit represented by the following Verilog process:

```
always @(clk,clr)
begin
    if(clr == 1'b1)
        Q <= 1'b0;
    else if(clk == 1'b0 && CE == 1'b1)
        begin
        if(C == 1'b0)
                Q <= A & B;
        else
                Q <= A | B;
        end
end
```

Why is *clr* on the sensitivity list whereas *C* is not?



2.37 **(a)** Write a conditional signal assignment statement to represent the 4-to-1 MUX shown subsequently. Assume that there is an inherent delay in the MUX that causes the change in output to occur 10 ns after a change in input.
(b) Repeat **(a)** using an if-else statement.
(c) Repeat **(a)** using a case statement.



(a) Conditional Statement for 4 to 1 MUX w/ delay 10 ns.

```
assign #10 F = (C==0)? ((D==0)? ~A:B):((D==0)? ~B:0);
```

(b)

```
always @(*)
begin
  if (C==0 && D==0)
    #10 F = ~A;
  else if (C==0 && D==1)
    #10 F = B;
  else if (C==1 && D==0)
    #10 F = ~B;
  else
    #10 F = 0;
end
```

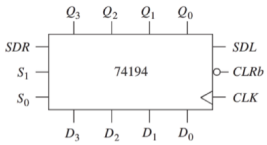
© 4-to-1 MUX w/ inherent delay of 10ns

```
always @ (*)
  begin
    case (sel)
      0: #10 F = ~A;
      1: #10 F = B;
      2: #10 F = ~B;
      3: #10 F = 0;
    end case
  end
```

2.43 Write a Verilog module that describes a 16-bit serial-in, serial-out shift register with inputs *SI* (serial input), *EN* (enable), *CK* (clock, shifts on rising edge), and a serial output (*SO*).

```
Module ShiftReg (SI, EN, CK, SO);
  input SI, EN, CK
  output SO;
  output [15:0] F;
  reg [15:0] R;
  initial begin
    R <= 0
  end
  always @(posedge CK)
    begin if (EN == 1)
      R <= {SI, register[15:1]};
    end
  assign SO = register[0];
  assign F = register;
end module
```

2.44 A description of a 74194 4-bit bidirectional shift register follows. The *CLRb* input is asynchronous and active low and overrides all the other control inputs. All other state changes occur following the rising edge of the clock. If the control inputs $S_1 = S_0 = 1$, the register is loaded in parallel. If $S_1 = 1$ and $S_0 = 0$, the register is shifted right and *SDR* (serial data right) is shifted into Q_3. If $S_1 = 0$ and $S_0 = 1$, the register is shifted left and *SDL* is shifted into Q_0. If $S_1 = S_0 = 0$, no action occurs.

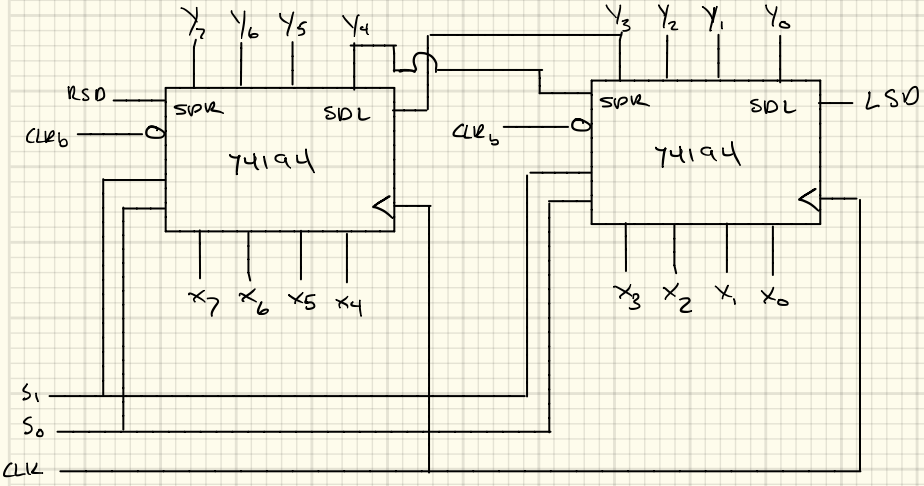


(a) Write a behavioral-level Verilog model for the 74194.
(b) Draw a block diagram and write a Verilog description of an 8-bit bidirectional shift register that uses two 74194s as components. The parallel inputs and outputs to the 8-bit register should be *X*([7:0]) and *Y*([7:0]). The serial inputs should be *RSD* and *LSD*.

(a)

```
module biShiftReg74194(D,S,SDR,SDL,CLRb,CLK,Q);
input [3:0] D;
input [1:0] S;
input SDR,SDL,CLRb,CLK;
output reg [3:0] Q;
initial begin
  Q<=0;
end
always @(CLK,CLRb)
begin
  if(CLRb==0)
  Q<=4'b0000;
  else if (CLK==1)
    begin
      case (S)
        0: Q<= Q;
        1: Q<={Q[2:0],SDL};
        2: Q<={SDR,Q[3:1]};
        3: Q<=D;
      endcase
  end
end
endmodule
```

(b) Draw a block diagram and write a Verilog description of an 8-bit bidirectional shift register that uses two 74194s as components. The parallel inputs and outputs to the 8-bit register should be $X([7:0])$ and $Y([7:0])$. The serial inputs should be *RSD* and *LSD*.




```
module biShiftReg 8bit (X, S, RSD, LSD, CLRb, CLK, Y);
  input [7:0] X;
  input [1:0] S;
  input RSD, LSD, CLRb, CLK;
    output [7:0] Y;
  biShiftRegister 74194 s1 (X[3:0], S, Y[4], LSD, CLRb, CLK,
    Y[3:0]);
  biShiftRegister 74194 s2 (X[7:4], S, RSD, Y[3], CLRb, CLK,
    Y[7:4]);
  end module
```

2.48 Complete the following Verilog code to implement a counter that counts in the following sequence: $Q = 1000, 0111, 0110, 0101, 0100, 0011, 1000, 0111, 0110, 0101, 0100, 0011, \ldots$ (repeats). The counter is synchronously loaded with 1000 when *Ld8* = 1. It goes through the prescribed sequence when *Enable* = 1. The counter outputs $S_5 = 1$ whenever it is in state 0101. Do not change the provided structure of the following module in any way. Your code must be synthesizable.

```
module countQ1(clk,Ld8,Enable,S5,Q);
input clk,Ld8,Enable;
output reg S5;
output reg[3:0] Q;
.
.
.
endmodule
```

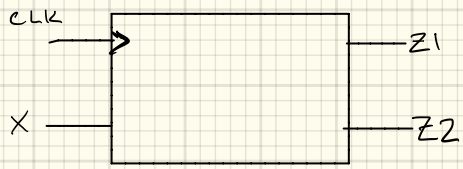
```
module countQ1 (clk, Ld8, Enable, S5, Q);
  input clk, Ld8, Enable;
    output S5;
    output [3:0] Q;
  reg [3:0] Qtemp;
initial begin
    Qtemp=0
end
always @(posedge clk)
  begin
    if (Ld8 == 1)
    Qtemp <=4'b1000;
  else if (Enable==1)
  begin
    if(Qtemp == 4'b1000)
      Qtemp <= 4'b1000;
    else
     Qtemp <= Qtemp -1;
    end
    end
    assign S5= (Qtemp == 4'b0101)? 1:0;
    assign Q = Qtemp;
end module
```

2.50 Examine the following Verilog code and answer the following questions

```
module Problem(X,CLK,Z1,Z2);
input X,CLK;
output Z1,Z2;

reg [1:0]State,Nextstate;

initial
begin
     State = 2'b00;
     Nextstate = 2'b00;
end

always @(State,X)
begin
    case(State)
     0:begin
           if(X == 1'b0)begin
                 Z1 = 1'b1;
                 Z2 = 1'b0;
                 Nextstate = 2'b00;
           end
           else begin
                 Z1 = 1'b0;
                 Z2 = 1'b0;
                 Nextstate = 2'b01;
           end
     end
     1:begin
           if(X == 1'b0)begin
                 Z1 = 1'b0;
                 Z2 = 1'b1;
                 Nextstate = 2'b01;
           end
           else begin
                 Z1 = 1'b0;
                 Z2 = 1'b1;
                 Nextstate = 2'b10;
           end
     end
     2:begin
           if(X == 1'b0)begin
                 Z1 = 1'b0;
                 Z2 = 1'b1;
                 Nextstate = 2'b10;
           end
           else begin
                 Z1 = 1'b0;

                 Z2 = 1'b1;
                 Nextstate = 2'b11;
           end
     end
     3:begin
           if(X == 1'b0)begin
                 Z1 = 1'b0;
                 Z2 = 1'b0;
                 Nextstate = 2'b00;
           end
           else begin
                 Z1 = 1'b1;
                 Z2 = 1'b0;
                 Nextstate = 2'b01;
           end
     end
           endcase
     end

     always @(posedge CLK)
     begin
         State <= Nextstate;
     end
     endmodule
```

(a) Draw a block diagram of the circuit implemented by this code.
(b) Write the state table that is implemented by this code.

(a)



(b)

current state	Next state					
	X=0	X=1	Z1	Z2	Z1	Z2
S0	S0	S1	1	0	0	1
S1	S1	S2	0	1	0	1
S2	S2	S3	0	1	0	1
S3	S0	S1	0	0	1	0