Vincent Han

EE 4301

Homework 5

6.4 The next-state equations for a sequential circuit with two flip-flops (Q_1 and Q_2), input signals R, S, T, and an output P are

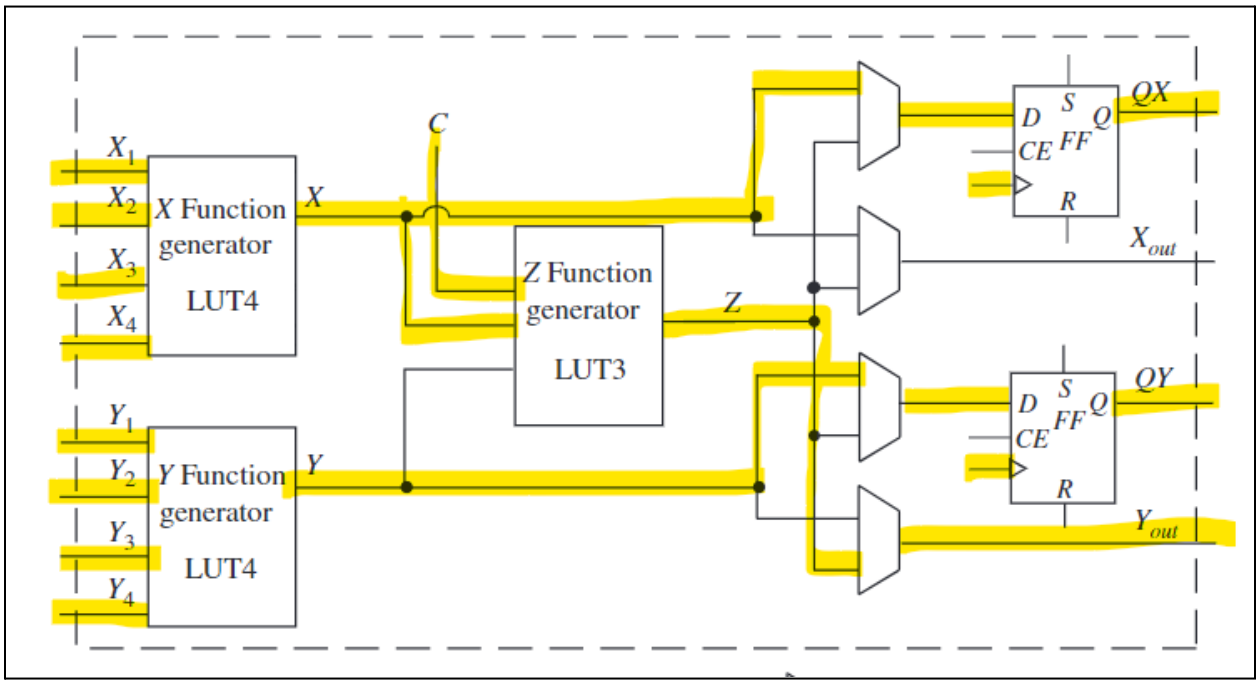
$$D_1 = Q_1^+ = Q_2R + Q_1S$$
$$D_2 = Q_2^+ = Q_1 + Q_2'T$$

The output equation is $P = Q_2RT + Q_1ST$

(a) Explain how this sequential circuit can be implemented using a single Figure 6-3 logic block. Write the equation that each function generator in the block will implement.

(b) Mark (highlight) the input signals, the state and output variables, and the activated paths on a copy of Figure 6-3.

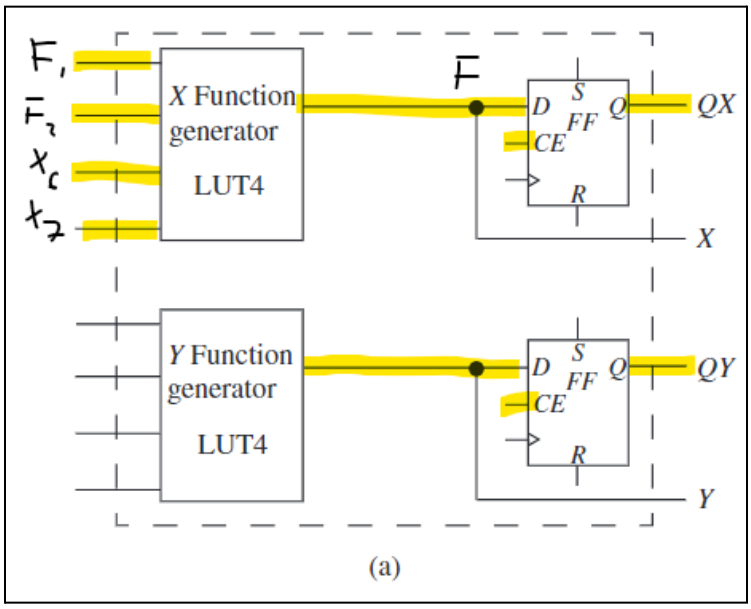
```
a. If you use the X function gen and R, S, Q1 and Q2 as inputs you can show the next
state of Q1. Then if you use the Y function gen with the inputs of T, Q1 and Q2,
you can show the next state of Q2. We can show the output P if we use the Z function
gen with the inputs T and then also the X function gen.

b.
```



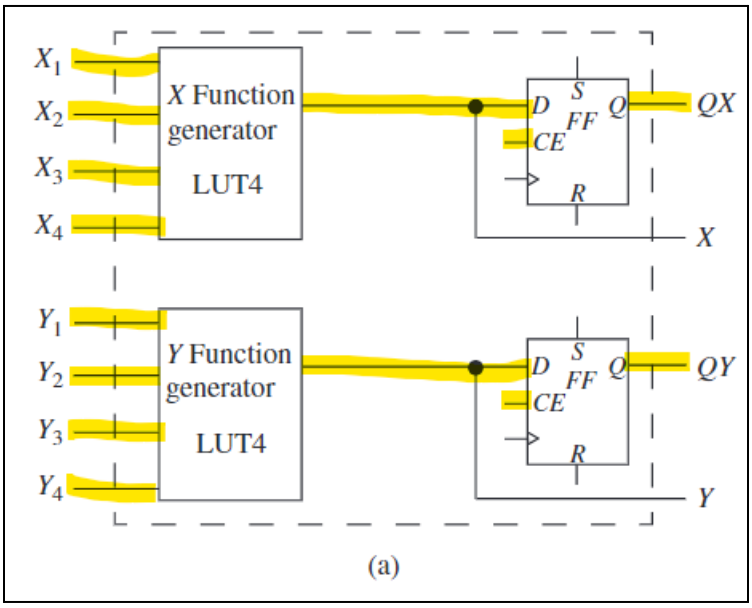
6.10 Show how to realize the following combinational function using two Figure 6-1(a) logic blocks. Show the connections on a copy of Figure 6-1(a) and give the functions X and Y for both blocks.

$$F = X_1'X_2X_3'X_6 + X_2'X_3'X_4X_6' + X_2X_3'X_4' + X_2X_3X_4'X_6 + X_3'X_4X_5X_6' + X_7$$

```
F = X6F1 + X'6F2 + X7
```



(a)



(a)

```
For inputs X1, X2, X3, and X4 on X:

    F1 = X'1X2X'3 + X2X'3X'4 + X2X3X'4


For inputs X2, X3, X4, and X5 on Y:

    F2 = X'2X'3X4 + X2X'3X'4 + X'3X4X5

For inputs F1, F2, X6, and X7 on X in the second block:

    F = X6F1 + X'6F2 + X7
```

7.16 Two floating-point numbers are added to form a floating-point sum:

$$(F_1 \times 2^{E1}) + (F_2 \times 2^{E2}) = F \times 2^E$$

Assume that F_1 and F_2 are normalized and the result should be normalized.

(a) List the steps required to carry out floating-point addition, including all special cases.

(b) Illustrate these steps for $F_1 = 1.0101$, $E_1 = 1001$, $F_2 = 0.1010$, and $E_2 = 1000$. Note that the fractions are 5 bits, including sign, and the exponents are 4 bits, including sign.

(c) Write a Verilog description of the system.

Answer was directly from the textbook:

a.

In summary, the steps required to carry out floating-point addition are as follows:

1. Compare exponents. If the exponents are not equal, shift the fraction with the smaller exponent right and add 1 to its exponent; repeat until the exponents are equal.
2. Add the fractions (significands).
3. If the result is 0, set the exponent to the appropriate representation for 0 and exit.
4. If fraction overflow occurs, shift right and add 1 to the exponent to correct the overflow.
5. If the fraction is unnormalized, shift left and subtract 1 from the exponent until the fraction is normalized.
6. Check for exponent overflow. Set overflow indicator, if necessary.
7. Round to the appropriate number of bits. Is it still normalized? If not, go back to step 4.

b.

1. Make exponents the same	F1: 1.0101	E1: 1001	(-11/16 * 2^-6)
	F2: 0.0101	E2: 1001	(5/16 2^-6)
2. Add fractions	F: 1.1010	E: 1001	(-6/16 * 2^-6)
3. Normalize fractions	F: 1.0100	E: 1000	(-12/16 * 2^-7)
4. Check for overflow	None		

c.

c.

```
module Prob7_16(CLK, St, F1, F2, E1, E2, V, Done);

    input CLK, St;
    input [4:0] F1, F2;
    input [3:0] E1, E2;
    output reg V, Done;
    reg [5:0] RegF1;
    reg [4:0] RegF2;
    reg [4:0] RegE1;
    reg [3:0] RegE2;
    wire [5:0] addout;
    wire [4:0] subout;
    reg Load, Ad, LM8, RSX1, LS, RSX2, INC1, INC2, DEC1, LdTwoToOne;
    wire LT, GT, E1G, E2G, EV, SubV;
    wire FZ, FV, Fnorm;
    reg [1:0] state, nextstate;
```

```
initial begin
    V = 0;
    Done = 0;
    RegF1 = 0;
    RegF2 = 0;
    RegE1 = 0;
    RegE2 = 0;
    Load = 0;
    Ad = 0;
    LM8 = 0;
    RSX1 = 0;
    LS = 0;
    RSX2 = 0;
    INC1 = 0;
    INC2 = 0;
    DEC1 = 0;
    LdTwoToOne = 0;
    state = 0;
    nextstate = 0;
end

assign subout = RegE1 - {RegE2[3], RegE2};
assign addout = RegF1 + {RegF2[4], RegF2};
assign SubV = (subout[4] != subout[3])? 1 : 0;
assign EV = (RegE1[4] != RegE1[3])? 1 : 0;
assign LT = subout[3];
assign GT = (subout[3] == 0 && subout[3:0] != 3'b000)? 1 : 0;
assign E1G = ((SubV == 0 && subout[3] == 0 && subout[2] == 1 && (~(subout[1:0])==2'b00))||(SubV == 1 && RegE1[3] == 0 && RegE2[3] == 1))? 1 : 0;

assign E2G = ((SubV == 0 && subout[3] == 1 && subout[2] == 0) || (SubV == 1 && RegE1[3] == 1 && RegE2[3] == 0))? 1 : 0;

assign FZ = (RegF1 == 0)? 1 : 0;
assign FV = (RegF1[5] != RegF1[4])? 1 : 0;
assign Fnorm = (RegF1[4] != RegF1[3])? 1 : 0;

always @(state, St, LT, GT, E1G, E2G, FZ, FV, Fnorm, EV)
    begin
        Load = 0; Ad = 0; LM8 = 0; RSX1 = 0; LS = 0; V = 0;
        RSX2 = 0; INC1 = 0; INC2 = 0; DEC1 = 0; Done = 0;
        LdTwoToOne = 0;
        case(state)
        0: begin
        if(St == 1) begin
        Load = 1;
        nextstate = 1;
    end
```

```
    else begin
        nextstate = 0;
    end
end

    1: begin
        if(E1G == 1) begin
            nextstate = 3;
        end

        else if(E2G == 1) begin
            LdTwoToOne = 1;
            nextstate = 3;
        end

        else if(GT == 1) begin
            RSX2 = 1;
            INC2 = 1;
            nextstate = 1;
        end

        else if(LT == 1) begin
            RSX1 = 1;
            INC1 = 1;
            nextstate = 1;
        end

        else begin
            Ad = 1;
            nextstate = 2;
        end
    end

    2: begin
        if(FZ == 1) begin
            LM8 = 1;
            nextstate = 3;
        end

        else if(FV == 1) begin
            RSX1 = 1;
            INC1 = 1;
            nextstate = 3;
        end
```

```
        else if(Fnorm == 0) begin
            LS = 1;
            DEC1 = 1;
            nextstate = 2;
        end

        else begin
            nextstate = 3;
        end
    end

    3: begin
        Done =1 ;

        if(St == 0)
            nextstate = 0;

        else
            nextstate = 3;

            if(EV == 1)
                V = 1;
            end
    endcase
end

always @(posedge CLK)
    begin
        state <= nextstate;
        if(Load == 1) begin
        RegF1 <= {F1[4],F1};
        RegF2 <= F2;
        RegE1 <= {E1[3],E1};
        RegE2 <= E2;
    end

    else begin

    //RegE1
    if(LM8 == 1)
        RegE1 <= 5'b11000;
        if(LdTwoToOne == 1) begin
        RegE1 <= {RegE2[3], RegE2};
        RegF1 <= {RegF2[4], RegF2};
    end
```

```
        else begin

        //RegE1
        if(LM8 == 1)
            RegE1 <= 5'b11000;
            if(LdTwoToOne == 1) begin
            RegE1 <= {RegE2[3], RegE2};
            RegF1 <= {RegF2[4], RegF2};
        end

         if(INC1 == 1)
            RegE1 <= RegE1 + 1;

        else if(DEC1 == 1)
            RegE1 <= RegE1 - 1;

        //RegE2
        if(INC2 == 1)
            RegE2 <= RegE2 + 1;

        //REGF1
        if(LS == 1)
            RegF1 <= {RegF1[4:0], 1'b0};

        else if(RSX1 == 1)
            RegF1 <= {RegF1[5], RegF1[5:1]};

        else if(Ad == 1)
            RegF1 <= addout;

        //RegF2
        if(RSX2 == 1)
            RegF2 <= {RegF2[4], RegF2[4:1]};
        end
    end
endmodule
```

7.22 Assume that A, B, and C are floating-point numbers expressed in IEEE single precision floating-point format and that floating-point addition is performed. If $A = 2^{40}$, $B = -2^{40}$, $C = 1$, then

(a) What is A + (B + C)? (B + C is done first, and then A is added to it.)
(b) What is (A + B) + C? (A + B is done first, and then C is added to it.)

```
7.22

a.

B + C = -2^-40 + 1

(-1 * 2^40) + (2^40)

2^40 + -2^40

1 + (-1) * (2^40) = 0

A + (B + C) = 0


b.

2^40 + (-2^40)

1 + (-2) * 2^40 = 0

0 + 1 = 1

(A + B) + C = 1
```