Jacob Sickafoose

Lab # 5

Lab Section - 1B

2/21/2020

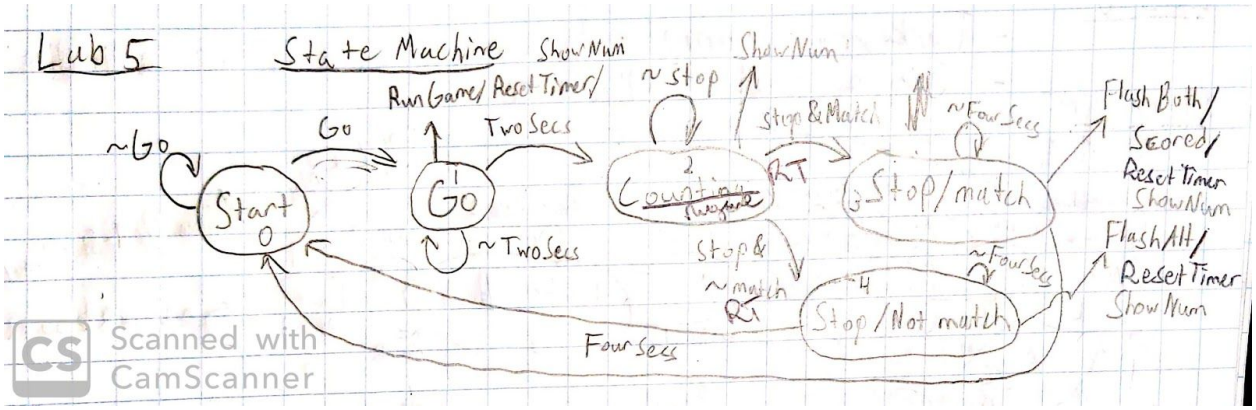
**Description -**

For this lab, we were tasked with creating a game. When the go button was pressed, a random number was displayed and a counter next which incremented every quarter second. If the stop button was pressed when the counter equaled the given number, a point was scored and tracked on the leds. The highest number the counter could reach was just 3F to avoid waiting too long for a full count up. This lab required creating a state machine for keeping track of what state the game was in, and the outputs associated with each of those states. It also involved coding the LSFR which cycled through, seemingly random numbers. When we stored it's output at a random time, it gave us a pseudo-random number.

**Method -**

I started this lab by first implementing the logic for the state machine. I started by drawing out the following flow chart with each state and what variables turned on which states,

as well as their outputs:

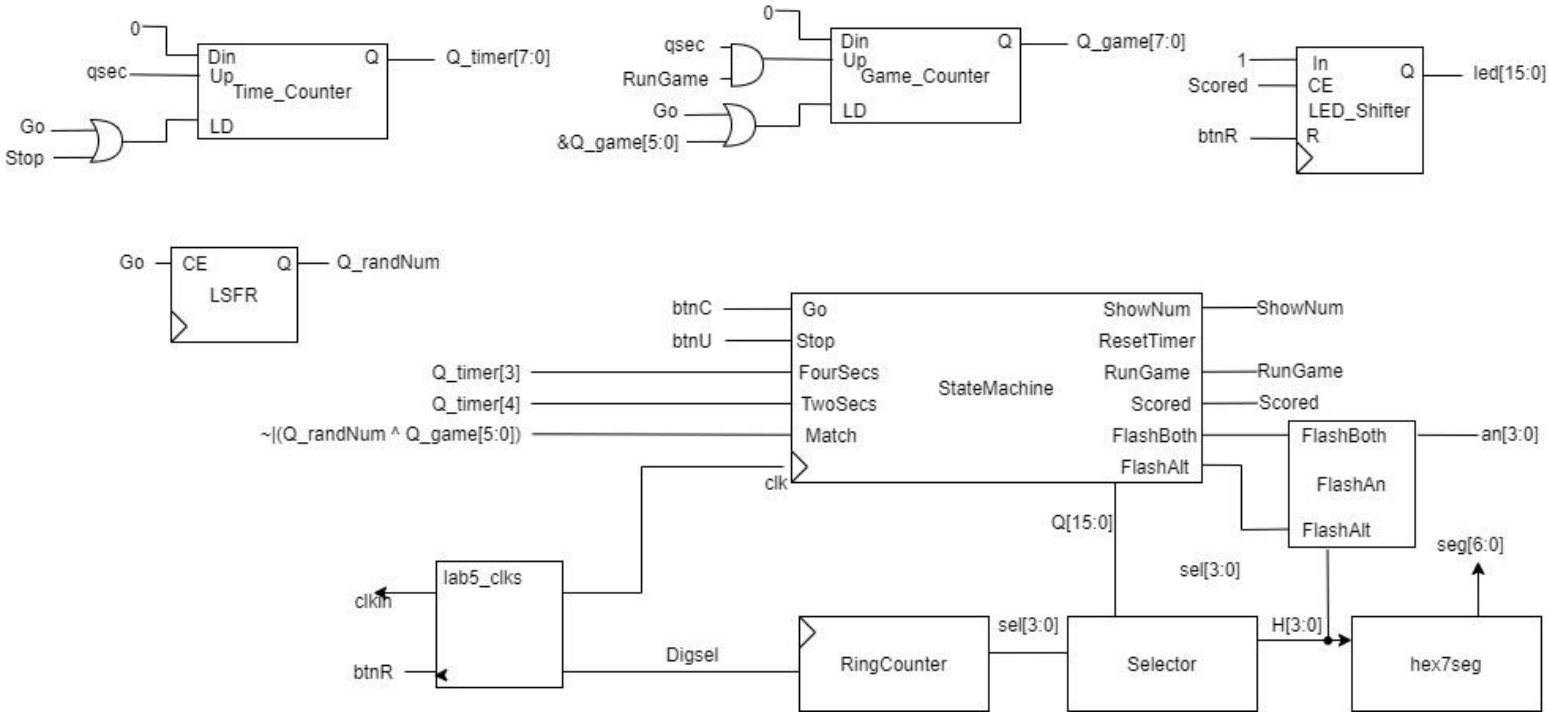



I used the one hot method to implement the states with flip flops. This means that there is a flip flop for each state and only one flip flop is high at a time. The flip flop that is high, is the current state. This made it much easier to implement the logic to switch between states. The LED_Shifter was also fairly easy to implement. I pretty much just had 16 flip flops, set up as a shifter, with the output of the last one acting as the input to the current one. The first flip flop had it's D set to a hard coded high bit. For the counters, I just used two identical Counter8UDL's from the last lab. I had the Din set to 8, 0 bits and the LD was set to reset when it hit 3F. The hardest part of the lab was creating the top level. It took a lot of creativity figuring out what to set some of the state machine inputs to. The TwoSecs and FourSecs were set to high when the Time_Counter's 4th bit, and 5th bit were high. I also had to set the LSFR to continuously cycle through its random number until the Time_Counter was stopped. I also had to make the left and right sides of the display flash, either alternating or at the same time. This depended on if the player scored a point or not. I did this first by making the qsec signal a perfect square wave which was high and low each for a quarter second, with an edge detector. I then used this square signal ANDed with FlashBoth and FlashAlt to make them flash every quarter second. If FlashAlt

was the one high, I NOTed the squared qsec signal for the right side so that it only flashed when the other side didn't.

**Design -**

This is the schematic for my top level:




I had to figure out how to change the states based on the inputs and using a different flip flop for each state. I ended with the following expressions for the Dins on the flip flops:

$D_0$= ($Q_0$&~Go) | ($Q_4$&FourSecs) | ($Q_3$&FourSecs)

$D_1$= ($Q_0$&Go) | ($Q_1$&~TwoSecs)

$D_2$= ($Q_1$&TwoSecs) | ($Q_2$&~Stop)

$D_3$= ($Q_2$&Stop&Match) | ($Q_3$&~FourSecs)

$D_4$= ($Q_2$&Stop&~Match) | ($Q_4$&~FourSecs)

I also ended up with the following equations for controlling the AN's in my FlashAn module:

```
an[0] = ~(sel[0]&(~(FlashBoth&temp) & ~(FlashAlt&temp)));
an[1] = ~(sel[1]&(~(FlashBoth&temp) & ~(FlashAlt&temp)));
an[2] = ~(ShowNum & sel[2]&(~(FlashBoth&temp) & ~(FlashAlt&~temp)));
an[3] = ~(ShowNum & sel[3]&(~(FlashBoth&temp) & ~(FlashAlt&~temp)));
```

**Conclusion -**

In order to finish this lab, I had to learn how to implement state machines using flip flops. I utilized the one hot method for doing this, which was probably the easiest but less efficient method. I had to figure out how to make the inputs to the state machine, control which state I was in as well as make the state, control the output of the machine. The longest part was figuring out how to plug everything together. It took me awhile just to figure out how the two counters functioned and when they were counting, and when they reset. It was also very time consuming, setting up when the different AN's flashed and debugging why they weren't working for a long time. If I had to do this same lab again, it would probably be much easier now that I have it all figured out.

**Appendix -**

```
module top_lab5(
    input clkin,
    input btnR,
    input btnU,
    input btnC,
    output [15:0] led,
    output [3:0] an,
    output dp,
    output [6:0] seg
    );
    /*Clk           */wire digsel, qsec, clk;
    /*StateMachine  */wire Go, Stop, FourSecs, TwoSecs, Match, ShowNum, ResetTimer,
                           RunGame, Scored, FlashBoth, FlashAlt;
    /*Game_Counter  */wire [7:0] Q_game;
    /*Time_Counter  */wire [7:0] Q_timer;
    /*LSFR          */wire [7:0] Q_LSFR;
    /*LSFR stored   */wire [5:0] Q_randNum;
    /*Reset button  */wire R;
    /*Ring Counter  */wire [3:0] sel;
    /*Selector      */wire [3:0] H;
                      wire [15:0] N;

//    assign Go = btnC;
//    assign Stop = btnU;
//    assign R = btnR;
    FDRE #(.INIT(1'b0)) B0_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(btnC), .Q(Go));
    FDRE #(.INIT(1'b0)) B1_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(btnU), .Q(Stop));
    FDRE #(.INIT(1'b0)) B2_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(btnR), .Q(R));

    lab5_clks lab5_clk (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel), .qsec(qsec));
    stateMachine StateMachine (.Go(Go), .Stop(Stop), .FourSecs(FourSecs), .TwoSecs(TwoSecs), .Match(Match), .clk(clk),
        .ShowNum(ShowNum), .ResetTimer(ResetTimer), .RunGame(RunGame), .Scored(Scored), .FlashBoth(FlashBoth), .FlashAlt(FlashAlt));
    counterUD8L Game_Counter (.clk(clk), .Din(8'b00000000), .Up(qsec & RunGame), .Dw(1'b0), .LD(Go | (&Q_game[5:0])), .Q(Q_game));
    counterUD8L Time_Counter (.clk(clk), .Din(8'b00000000), .Up(qsec), .Dw(1'b0),.LD(Go | Stop), .Q(Q_timer));

    assign TwoSecs = Q_timer[3]&~Q_timer[2]&~Q_timer[1]&~Q_timer[0];
    assign FourSecs = Q_timer[4]&~Q_timer[3]&~Q_timer[2]&~Q_timer[1]&~Q_timer[0];

    // Stores the LSFR Output into the 6-bit bus, Q_randNum. Then makes sure it matches with Q_game[5:0]
    randomNum LSFR (.clk(clk), .Q(Q_LSFR));
    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(1'b0), .CE(Go), .D(Q_LSFR[0]), .Q(Q_randNum[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(Go), .D(Q_LSFR[1]), .Q(Q_randNum[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(1'b0), .CE(Go), .D(Q_LSFR[2]), .Q(Q_randNum[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(1'b0), .CE(Go), .D(Q_LSFR[3]), .Q(Q_randNum[3]));
    FDRE #(.INIT(1'b0)) Q4_FF (.C(clk), .R(1'b0), .CE(Go), .D(Q_LSFR[4]), .Q(Q_randNum[4]));
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .R(1'b0), .CE(Go), .D(Q_LSFR[5]), .Q(Q_randNum[5]));
    assign Match = ~|(Q_randNum ^ Q_game[5:0]);

    assign N[7:0] = Q_game;
    assign N[15:8] = Q_randNum;
```

```
    // Sets up the LED Shifter
    wire tp;
    edgeDetector EdgeDetector (.btn(Scored), .clk(clk), .btnOut(tp));
//    FDRE #(.INIT(1'b0)) B3_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(Scored), .Q(tp));
    LED_Shifter LED_Shifter (.In(1'b1), .CE(tp), .R(R), .clk(clk), .Q(led));
    // Sets up the Modules for displaying to the 7seg
    ringCounter RingCounter (.adv(digsel), .clk(clk), .sel(sel));
    selector Selector (.sel(sel), .N(N), .H(H));
    hex7seg hex7seg (.n(H), .seg(seg));
    wire temp;

//    assign temp = 1'b1;
    FDRE #(.INIT(1'b0)) B4_FF (.C(clk), .R(1'b0), .CE(qsec), .D(~temp), .Q(temp));
    assign dp = 1'b1;
    assign an[0] = ~(sel[0]&(~(FlashBoth&temp) & ~(FlashAlt&temp)));
    assign an[1] = ~(sel[1]&(~(FlashBoth&temp) & ~(FlashAlt&temp)));
    assign an[2] = ~(ShowNum & sel[2]&(~(FlashBoth&temp) & ~(FlashAlt&~temp)));
    assign an[3] = ~(ShowNum & sel[3]&(~(FlashBoth&temp) & ~(FlashAlt&~temp)));
//    assign an = ~sel;
endmodule
```

```
module stateMachine(
    input Go,
    input Stop,
    input FourSecs,
    input TwoSecs,
    input Match,
    input clk,
    output ShowNum,
    output ResetTimer,
    output RunGame,
    output Scored,
    output FlashBoth,
    output FlashAlt
    );
    wire [4:0] D;
    wire [4:0] Q;

    assign D[0] = (Q[0]&~Go) | (Q[4]&FourSecs) | (Q[3]&FourSecs);
    assign D[1] = (Q[0]&Go) | (Q[1]&~TwoSecs);
    assign D[2] = (Q[1]&TwoSecs) | (Q[2]&~Stop);
    assign D[3] = (Q[2]&Stop&Match) | (Q[3]&~FourSecs);
    assign D[4] = (Q[2]&Stop&~Match) | (Q[4]&~FourSecs);

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[3]), .Q(Q[3]));
    FDRE #(.INIT(1'b0)) Q4_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[4]), .Q(Q[4]));

    assign RunGame = Q[2];
    assign ResetTimer = Q[1];
    assign ShowNum = ~Q[0];
    assign FlashBoth = Q[3];
    assign Scored = Stop & Match;
    assign FlashAlt = Q[4];

endmodule
```

```
module randomNum(
    input clk,
    output [7:0] Q
    );
    wire D;

    assign D = (Q[0]^Q[5]^Q[6]^Q[7]);

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(Q[0]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(Q[1]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(Q[2]), .Q(Q[3]));
    FDRE #(.INIT(1'b0)) Q4_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(Q[3]), .Q(Q[4]));
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(Q[4]), .Q(Q[5]));
    FDRE #(.INIT(1'b0)) Q6_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(Q[5]), .Q(Q[6]));
    FDRE #(.INIT(1'b0)) Q7_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(Q[6]), .Q(Q[7]));

endmodule
```

```
module LED_Shifter(
    input In,
    input CE,
    input R,
    input clk,
    output [15:0] Q
    );

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(R), .CE(CE), .D(In), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(R), .CE(CE), .D(Q[0]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(R), .CE(CE), .D(Q[1]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(R), .CE(CE), .D(Q[2]), .Q(Q[3]));
    FDRE #(.INIT(1'b0)) Q4_FF (.C(clk), .R(R), .CE(CE), .D(Q[3]), .Q(Q[4]));
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .R(R), .CE(CE), .D(Q[4]), .Q(Q[5]));
    FDRE #(.INIT(1'b0)) Q6_FF (.C(clk), .R(R), .CE(CE), .D(Q[5]), .Q(Q[6]));
    FDRE #(.INIT(1'b0)) Q7_FF (.C(clk), .R(R), .CE(CE), .D(Q[6]), .Q(Q[7]));
    FDRE #(.INIT(1'b0)) Q8_FF (.C(clk), .R(R), .CE(CE), .D(Q[7]), .Q(Q[8]));
    FDRE #(.INIT(1'b0)) Q9_FF (.C(clk), .R(R), .CE(CE), .D(Q[8]), .Q(Q[9]));
    FDRE #(.INIT(1'b0)) Q10_FF (.C(clk), .R(R), .CE(CE), .D(Q[9]), .Q(Q[10]));
    FDRE #(.INIT(1'b0)) Q11_FF (.C(clk), .R(R), .CE(CE), .D(Q[10]), .Q(Q[11]));
    FDRE #(.INIT(1'b0)) Q12_FF (.C(clk), .R(R), .CE(CE), .D(Q[11]), .Q(Q[12]));
    FDRE #(.INIT(1'b0)) Q13_FF (.C(clk), .R(R), .CE(CE), .D(Q[12]), .Q(Q[13]));
    FDRE #(.INIT(1'b0)) Q14_FF (.C(clk), .R(R), .CE(CE), .D(Q[13]), .Q(Q[14]));
    FDRE #(.INIT(1'b0)) Q15_FF (.C(clk), .R(R), .CE(CE), .D(Q[14]), .Q(Q[15]));
endmodule
```

## Modules used from previous labs :

```
module edgeDetector(
    input btn,
    input clk,
    output btnOut
    );
    wire [1:0] D;
    wire [1:0] Q;

    assign D[0] = btn;
    assign D[1] = Q[0];

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[1]), .Q(Q[1]));

    assign btnOut = ~Q[1]&Q[0];
endmodule
```

```
module counterUD8L(
    input clk,
    input [7:0] Din,
    input Dw,
    input Up,
    input LD,
    output UTC,
    output DTC,
    output [7:0] Q
    );
    wire [1:0] utc;
    wire [1:0] dtc;

    countUD4L count0 (.Up(Up), .Dw(Dw), .LD(LD), .Din(Din[3:0]), .clk(clk), .UTC(utc[0]), .DTC(dtc[0]), .Q(Q[3:0]));
    countUD4L count1 (.Up(utc[0]&Up | Up&UTC), .Dw(dtc[0]&Dw | Dw&DTC), .LD(LD), .Din(Din[7:4]), .clk(clk), .UTC(utc[1]), .DTC(dtc[1]), .Q(Q[7:4]));

    assign UTC = utc[0]&utc[1];
    assign DTC = dtc[0]&dtc[1];
endmodule
```

```
module countUD4L(
    input Up,
    input Dw,
    input LD,
    input [3:0] Din,
    input clk,
    output UTC,
    output DTC,
    output [3:0] Q
    );
    wire [3:0] D;

    assign D[0] = (Q[0]^Up^Dw)&~LD | Din[0]&LD;
    assign D[1] = (Q[1]^(Q[0]&Up)^(~Q[0]&Dw))&~LD | Din[1]&LD;
    assign D[2] = (Q[2]^(Q[0]&Q[1]&Up)^(~Q[0]&~Q[1]&Dw))&~LD | Din[2]&LD;
    assign D[3] = (Q[3]^(Up&Q[2]&Q[1]&Q[0])^(Dw&~Q[2]&~Q[1]&~Q[0]))&~LD | Din[3]&LD;

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(1'b0), .CE(1'b1), .D(D[3]), .Q(Q[3]));

    assign UTC = Q[0]&Q[1]&Q[2]&Q[3];
    assign DTC = ~Q[0]&~Q[1]&~Q[2]&~Q[3];

endmodule
```

```
module ringCounter(
    input adv,
    input clk,
    output [3:0] sel
    );
//    wire [3:0] D;
    wire [3:0] Q;

//    assign D[0] = (Q[0]&~adv) | (Q[3]&adv);
//    assign D[1] = (Q[1]&~adv) | (Q[0]&adv);
//    assign D[2] = (Q[2]&~adv) | (Q[1]&adv);
//    assign D[3] = (Q[3]&~adv) | (Q[2]&adv);

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .R(1'b0), .CE(adv), .D(Q[3]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(1'b0), .CE(adv), .D(Q[0]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(1'b0), .CE(adv), .D(Q[1]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(1'b0), .CE(adv), .D(Q[2]), .Q(Q[3]));

    assign sel[0] = Q[0];
    assign sel[1] = Q[1];
    assign sel[2] = Q[2];
    assign sel[3] = Q[3];
endmodule
```

```
module selector(
    input [3:0] sel,
    input [15:0] N,
    output [3:0] H
    );

    assign H[0] = (~sel[3]&~sel[2]&~sel[1]&sel[0]&N[0]) | (~sel[3]&~sel[2]&sel[1]&~sel[0]&N[4]) | (~sel[3]&sel[2]&~sel[1]&~sel[0]&N[8]) | (sel[3]&~sel[2]&~sel[1]&~sel[0]&N[12]);
    assign H[1] = (~sel[3]&~sel[2]&~sel[1]&sel[0]&N[1]) | (~sel[3]&~sel[2]&sel[1]&~sel[0]&N[5]) | (~sel[3]&sel[2]&~sel[1]&~sel[0]&N[9]) | (sel[3]&~sel[2]&~sel[1]&~sel[0]&N[13]);
    assign H[2] = (~sel[3]&~sel[2]&~sel[1]&sel[0]&N[2]) | (~sel[3]&~sel[2]&sel[1]&~sel[0]&N[6]) | (~sel[3]&sel[2]&~sel[1]&~sel[0]&N[10]) | (sel[3]&~sel[2]&~sel[1]&~sel[0]&N[14]);
    assign H[3] = (~sel[3]&~sel[2]&~sel[1]&sel[0]&N[3]) | (~sel[3]&~sel[2]&sel[1]&~sel[0]&N[7]) | (~sel[3]&sel[2]&~sel[1]&~sel[0]&N[11]) | (sel[3]&~sel[2]&~sel[1]&~sel[0]&N[15]);

endmodule
```

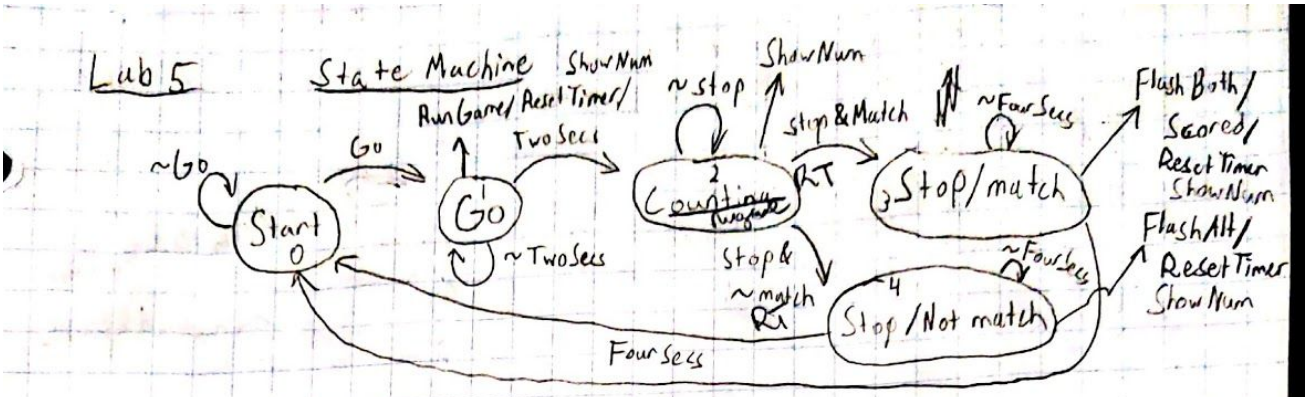
```
module m8_1(
    input [7:0] in,
    input [2:0] sel,
    output o
    );

    assign o = ((~sel[2]&~sel[1]&~sel[0]&in[0]) | (~sel[2]&~sel[1]&sel[0]&in[1]) |
    (~sel[2]&sel[1]&~sel[0]&in[2]) | (~sel[2]&sel[1]&sel[0]&in[3]) | (sel[2]&~sel[1]&~sel[0]&in[4]) |
    (sel[2]&~sel[1]&sel[0]&in[5]) | (sel[2]&sel[1]&~sel[0]&in[6]) | (sel[2]&sel[1]&sel[0]&in[7]));
endmodule

module hex7seg(
    input [3:0] n,
    output [6:0] seg
    );

    m8_1 a(.in({1'b0,n[0],n[0], 1'b0, 1'b0, ~n[0], 1'b0, n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[0]));
    m8_1 b(.in({1'b1,~n[0],n[0], 1'b0, ~n[0], n[0], 1'b0, 1'b0}), .sel({n[3], n[2], n[1]}), .o(seg[1]));
    m8_1 c(.in({1'b1,~n[0],1'b0, 1'b0, 1'b0, 1'b0, ~n[0], 1'b0}), .sel({n[3], n[2], n[1]}), .o(seg[2]));
    m8_1 d(.in({n[0],1'b0,~n[0], n[0], n[0], ~n[0], 1'b0, n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[3]));
    m8_1 e(.in({1'b0,1'b0,1'b0, n[0], n[0], 1'b1, n[0], n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[4]));
    m8_1 f(.in({1'b0,n[0],1'b0, 1'b0, n[0], 1'b0, 1'b1, n[0]}), .sel({n[3], n[2], n[1]}), .o(seg[5]));
    m8_1 g(.in({1'b0,~n[0],1'b0, 1'b0, n[0], 1'b0, 1'b0, 1'b1}), .sel({n[3], n[2], n[1]}), .o(seg[6]));
endmodule
```

**Lab Book :**




$D_0 = (Q_0 \& \sim Go) \mid (Q_4 \& FourSecs) \mid (Q_3 \& FourSecs)$

$D_1 = (Q_0 \& Go) \mid (Q_1 \& \sim TwoSecs)$

$D_2 = (Q_1 \& TwoSecs) \mid (Q_2 \& \sim stop)$

$D_3 = (Q_2 \& Stop \& Match) \mid (Q_3 \& \sim FourSecs)$

$D_4 = (Q_2 \& Stop \& \sim Match) \mid (Q_4 \& \sim FourSecs)$

an when ~ShowNum
→ 1100
else 0000 Flash both =

an 0 =

an 1 =

an 2 = ShowNum & (sel2

an 3 =

AS
1:35pm
2/14/20
5126

Scanned with CamScanner