Jacob Sickafoose

Lab # 7

Lab Section - 1B

3/13/2020

Description -

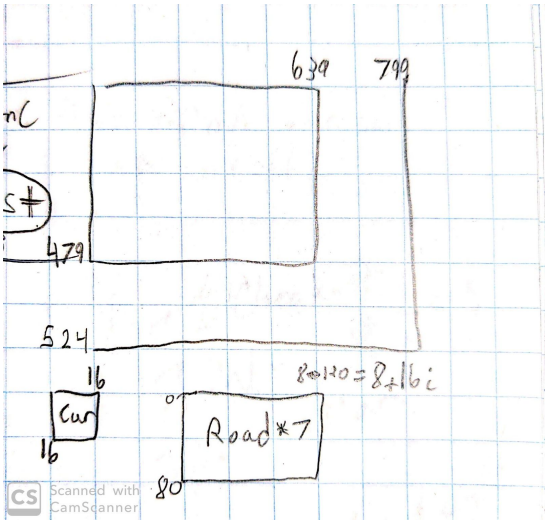
For this lab, we were tasked with creating a game using the VGA port on the boards. This game had a series of blocks, moving from the top to the bottom of the screen. Each block was a different color of the rainbow, and they needed to respawn at the top in a random position. The player was controlling a car which must never fully leave the road. While the game was playing, score was kept on the 7segment display and the leds shifted over. The width of the road segments could also be modified by using switches 6, 5, and 4.

Method -

I started this lab by first implementing the logic for the vga output. This required having two counters. One for the vertical segments and one for the horizontal. These counters represented coordinates on the screen which refreshed individual pixels to certain colors as they counted up. When the horizontal and vertical counters were within a certain pixel range, the Hsync and Vsync signals went low to keep the counters synced with the screen. Using the counters, I was able to set the coordinates for where to draw each of my blocks. I created a separate module which gave the bottom right coordinate of a block, when given the top left coord. This made it easier to draw the range of each block. I then needed 7 counters, one for each block which counted through the vertical pixels, making the blocks move down at 2 pixels per frame. This counter reset when the bottom of the block hit a low enough coord for the top of the

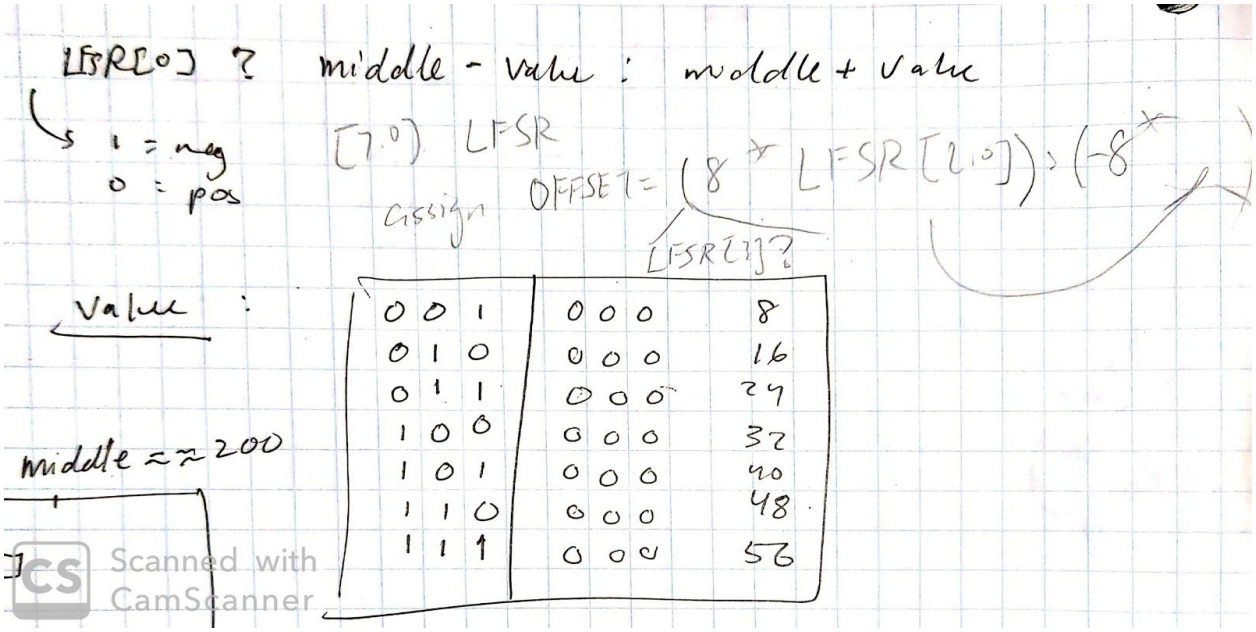
block to be off the screen. This coordinate was fed into the module which drew the rest of the block. I then had to calculate the X coordinate which was to be a new random number each time a block respawned. I took 4 bits from the LFSR. Three were taken and padded with a bottom three 0's to give me 8-56. The 4th bit of the LFSR was used to either subtract or add the 8-56 number from the block before it. If the resulting x would be off the screen, I did the opposite. So if adding would make it off the right of the screen, I would subtract instead and vice versa. I then had to make the logic to detect if the car was outside of a block. This was not hard because I already tracked the coordinates of two corners for both the blocks and the cars, so it was just a long sum of products statement for each of the 4 sides of each block. I lastly created the state machine to keep track of the progress of the game. I used the one hot method to implement the states with flip flops. Making the 7 segment display flash properly as well as the car and the led counter was tricky. I ended up making two frame counters. One to keep track of when TwoSecs had passed by and one for qsec for the flashing. This final step took me the longest to debug.

Design - I made the following guide to help me keep track of the bounds of the screen, blocks, and car:

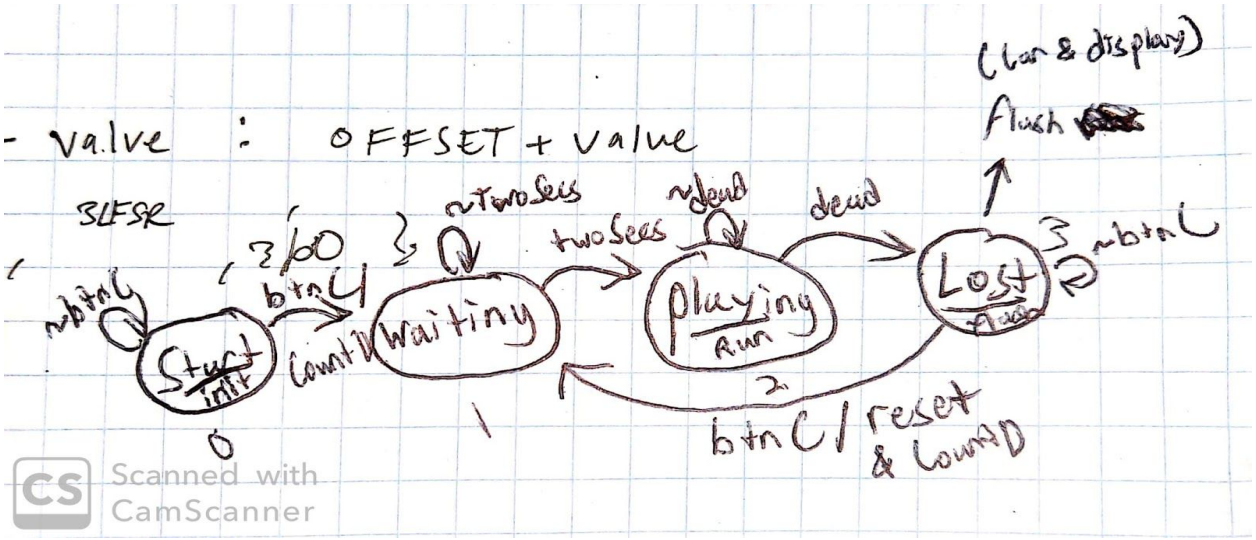


In order to keep track of my pixel count, I made new counters to store a 10bit number which was just enough to count to the bounds needed

for the screen. It also used a proper reset because for the previous labs I reset by loading in a Din of 0 bits and for this lab, the speed that my counters ran at was more important. I used the following logic to get the 8-56 from my LFSR and to add and subtract them I used the 4th LFSR bit.



I also used the following flow chart for my state machine



I implemented the state machine using the 1 hot encoding and it is a Mealy type machine.

Conclusion -

Finishing this lab took a lot. It ended up being suprisingly easy to display to the screen for me. I heard from someone that they used a state machine for each of the blocks and I spent way too much time trying to figure out how to implement them with a state machine. It became easy once I made a module to keep track of their bounds and a counter which moved them down the screen. The LFSR took a long time to finish because I kept padding my LFSR output with a {10{}} thinking it just added 0 bits to the front of my number. I tried multiple different methods for implementing the LFSR and they all didn't work because I was dumb. Making the buttons work was easy, it was just another counter which counted up and down, and added to all the X coordinates of the blocks. My Qsec counter was another thing that seemingly did not want to work for me. I got it working through sheer trial and error and I still don't really know what I changed.

Appendix -

```
module top_lab7(
   input clkin,
   input btnR,
   input btnL,
   input btnC,
   input [15:0] sw,
   output [3:0] vgaRed,
   output [3:0] vgaBlue,
   output [3:0] vgaGreen,
   output Hsync,
   output Vsync,
   output [15:0] led,
   output [3:0] an,
   output dp,
   output [6:0] seg
   );
   wire clk, digsel;
   wire [9:0] HQ, VQ;
   wire TwoSecs, Dead, init, CountDown, Run, Flash, reset, init1; // Wires for state machine
```

```
    lab7_clks lab7_clk(.clkin(clkin), .clk(clk), .greset(sw[0]),.digsel(digsel));
    counter10UDL Hcount (.Up(1'b1), .Reset(HQ>=10'd799), .clk(clk), .Q(HQ));
    counter10UDL Vcount (.Up(HQ>=10'd799), .Reset(VQ>=10'd524), .clk(clk), .Q(VQ));
    assign Hsync = ~((HQ >= 10'd655) && (HQ <= 10'd750));
    assign Vsync = ~((VQ >= 10'd489) && (VQ <= 10'd490));

    // Changes X when buttons L or R are pressed
    wire [9:0] BPQX;
    counter10UDL BlockPosX(.Up(btnL & Run), .Dw(btnR & Run), .Reset(reset), .clk(VQ == 0 & HQ ==
0), .Q(BPQX));

    edgeDetector EdgeDetector (.btn(init), .clk(clk), .btnOut(init1));
    // Tracks bottom Y of each block, reseting at the bottom
    wire [9:0] BPQ0, BPQ1, BPQ2, BPQ3, BPQ4, BPQ5, BPQ6, store0, store1, store2, store3, store4,
store5, store6;
    counter10UDL BlockPos0(.Up(~Vsync&HQ == 10'd0&Run), .Reset(BPQ0>=10'd559), .clk(clk /*&
Run*/), .init(10'd0), .LD(init1 | reset), .Q(BPQ0));
    counter10UDL BlockPos1(.Up(~Vsync&HQ == 10'd0&Run), .Reset(BPQ1>=10'd559), .clk(clk /*&
Run*/), .init(10'd80), .LD(init1 | reset), .Q(BPQ1));
    counter10UDL BlockPos2(.Up(~Vsync&HQ == 10'd0&Run), .Reset(BPQ2>=10'd559), .clk(clk /*&
Run*/), .init(10'd160), .LD(init1 | reset), .Q(BPQ2));
    counter10UDL BlockPos3(.Up(~Vsync&HQ == 10'd0&Run), .Reset(BPQ3>=10'd559), .clk(clk /*&
Run*/), .init(10'd240), .LD(init1 | reset), .Q(BPQ3));
    counter10UDL BlockPos4(.Up(~Vsync&HQ == 10'd0&Run), .Reset(BPQ4>=10'd559), .clk(clk /*&
Run*/), .init(10'd320), .LD(init1 | reset), .Q(BPQ4));
    counter10UDL BlockPos5(.Up(~Vsync&HQ == 10'd0&Run), .Reset(BPQ5>=10'd559), .clk(clk /*&
Run*/), .init(10'd400), .LD(init1 | reset), .Q(BPQ5));
    counter10UDL BlockPos6(.Up(~Vsync&HQ == 10'd0&Run), .Reset(BPQ6>=10'd559), .clk(clk /*&
Run*/), .init(10'd480), .LD(init1 | reset), .Q(BPQ6));

    // need to take 4 bits of LSFR, 16 possible numbers. Multiply the number by 8 and read in twosComp
    wire [7:0] lsfrOut;
    randomNum LSFR (.clk(clk), .Q(lsfrOut));
    storeNum storeNum0 (.Din((lsfrOut[6]) ? ((store1-lsfrOut[5:0]<=0) ? (store1+lsfrOut[5:0]) :
(store1-lsfrOut[5:0])) : ((store1+lsfrOut[5:0]>=519) ? (store1-lsfrOut[5:0]) : (store1+lsfrOut[5:0]))),
.LD(BPQ0 == 10'd0), .clk(clk), .Q(store0));
    storeNum storeNum1 (.Din((lsfrOut[6]) ? ((store2-lsfrOut[5:0]<=0) ? (store2+lsfrOut[5:0]) :
(store2-lsfrOut[5:0])) : ((store2+lsfrOut[5:0]>=519) ? (store2-lsfrOut[5:0]) : (store2+lsfrOut[5:0]))),
.LD(BPQ1 == 10'd0), .clk(clk), .Q(store1));
    storeNum storeNum2 (.Din((lsfrOut[6]) ? ((store3-lsfrOut[5:0]<=0) ? (store3+lsfrOut[5:0]) :
(store3-lsfrOut[5:0])) : ((store3+lsfrOut[5:0]>=519) ? (store3-lsfrOut[5:0]) : (store3+lsfrOut[5:0]))),
.LD(BPQ2 == 10'd0), .clk(clk), .Q(store2));
    storeNum storeNum3 (.Din((lsfrOut[6]) ? ((store4-lsfrOut[5:0]<=0) ? (store4+lsfrOut[5:0]) :
(store4-lsfrOut[5:0])) : ((store4+lsfrOut[5:0]>=519) ? (store4-lsfrOut[5:0]) : (store4+lsfrOut[5:0]))),
.LD(BPQ3 == 10'd0), .clk(clk), .Q(store3));
    storeNum storeNum4 (.Din((lsfrOut[6]) ? ((store5-lsfrOut[5:0]<=0) ? (store5+lsfrOut[5:0]) :
(store5-lsfrOut[5:0])) : ((store5+lsfrOut[5:0]>=519) ? (store5-lsfrOut[5:0]) : (store5+lsfrOut[5:0]))),
.LD(BPQ4 == 10'd0), .clk(clk), .Q(store4));
```

```
    storeNum storeNum5 (.Din((lsfrOut[6]) ? ((store6-lsfrOut[5:0]<=0) ? (store6+lsfrOut[5:0]) :
(store6-lsfrOut[5:0])) : ((store6+lsfrOut[5:0]>=519) ? (store6-lsfrOut[5:0]) : (store6+lsfrOut[5:0]))),
.LD(BPQ5 == 10'd0), .clk(clk), .Q(store5));
    storeNum storeNum6 (.Din((lsfrOut[6]) ? ((store0-lsfrOut[5:0]<=0) ? (store0+lsfrOut[5:0]) :
(store0-lsfrOut[5:0])) : ((store0+lsfrOut[5:0]>=519) ? (store0-lsfrOut[5:0]) : (store0+lsfrOut[5:0]))),
.LD(BPQ6 == 10'd0), .clk(clk), .Q(store6));

    // When given the X,Y coords of a corner, computes the coords of the block's opposite corner
    wire [9:0] outX0, outY0, outX1, outY1, outX2, outY2, outX3, outY3, outX4, outY4, outX5, outY5,
outX6, outY6;
    wire [9:0] inX0, inX1, inX2, inX3, inX4, inX5, inX6;
    assign inX0 = /*reset ? (10'd320) : */(store0+BPQX);
    assign inX1 = /*reset ? (10'd320) : */(store1+BPQX);
    assign inX2 = /*reset ? (10'd320) : */(store2+BPQX);
    assign inX3 = /*reset ? (10'd320) : */(store3+BPQX);
    assign inX4 = /*reset ? (10'd320) : */(store4+BPQX);
    assign inX5 = /*reset ? (10'd320) : */(store5+BPQX);
    assign inX6 = /*reset ? (10'd320) : */(store6+BPQX);
    blockSize BlockSize0(.inX(inX0), .inY(BPQ0), .width(sw[6:4]), .outX(outX0), .outY(outY0));
    blockSize BlockSize1(.inX(inX1), .inY(BPQ1), .width(sw[6:4]), .outX(outX1), .outY(outY1));
    blockSize BlockSize2(.inX(inX2), .inY(BPQ2), .width(sw[6:4]), .outX(outX2), .outY(outY2));
    blockSize BlockSize3(.inX(inX3), .inY(BPQ3), .width(sw[6:4]), .outX(outX3), .outY(outY3));
    blockSize BlockSize4(.inX(inX4), .inY(BPQ4), .width(sw[6:4]), .outX(outX4), .outY(outY4));
    blockSize BlockSize5(.inX(inX5), .inY(BPQ5), .width(sw[6:4]), .outX(outX5), .outY(outY5));
    blockSize BlockSize6(.inX(inX6), .inY(BPQ6), .width(sw[6:4]), .outX(outX6), .outY(outY6));

    wire [9:0] frameQ, qsec;

    counter10UDL frame (.Up(1'b1), .Reset(frameQ > 10'd121), .clk(VQ == 0 & HQ == 0), .Q(frameQ));
    counter10UDL Qsec (.Up(1'b1), .Reset(qsec > 10'd15), .clk(VQ == 0 & HQ == 0), .Q(qsec));
    wire qsec1;
    edgeDetector EdgeDetector1 (.btn(qsec>=15), .clk(clk), .btnOut(qsec1));

    assign TwoSecs = (frameQ >= 10'd120);
    assign Dead = ~((10'd328>=inX0 & 10'd312<=outX0 & 10'd392<=BPQ0 & 10'd408>=outY0) |
(10'd328>=inX1 & 10'd312<=outX1 & 10'd392<=BPQ1 & 10'd408>=outY1) | (10'd328>=inX2 &
10'd312<=outX2 & 10'd392<=BPQ2 & 10'd408>=outY2) | (10'd328>=inX3 & 10'd312<=outX3 &
10'd392<=BPQ3 & 10'd408>=outY3) | (10'd328>=inX4 & 10'd312<=outX4 & 10'd392<=BPQ4 &
10'd408>=outY4) | (10'd328>=inX5 & 10'd312<=outX5 & 10'd392<=BPQ5 & 10'd408>=outY5) |
(10'd328>=inX6 & 10'd312<=outX6 & 10'd392<=BPQ6 & 10'd408>=outY6));
    stateMachine StateMachine (.clk(clk), .btnC(btnC), .TwoSecs(TwoSecs), .Dead(Dead), .init(init),
.CountDown(CountDown), .Run(Run), .Flash(Flash), .reset(reset));
    wire [3:0] ringLed;
    RingCounterLed ringCounterLED0 (.adv(Run&qsec1), .clk(clk), .sel(ringLed), .reset(reset));

    assign led[15] = (frameQ >= 30)&CountDown | Run&ringLed[3];
    assign led[11] = (frameQ >= 60)&CountDown | Run&ringLed[3];
    assign led[7] = (frameQ >= 90)&CountDown | Run&ringLed[3];
```

```
    assign led[3] = (frameQ >= 120)&CountDown | Run&ringLed[3];

    assign led[14:12] = {3{Run}}&ringLed[2:0];
    assign led[10:8] = {3{Run}}&ringLed[2:0];
    assign led[6:4] = {3{Run}}&ringLed[2:0];
    assign led[2:0] = {3{Run}}&ringLed[2:0];

    assign vgaRed = ((4'h0&{4{HQ>10'd639 | VQ>10'd479}}) | (4'hf&{4{HQ>=10'd312 & HQ<=10'd328
& VQ>=10'd392 & VQ<=10'd408 & (~Flash | Flash&(frameQ<10'd30) |
Flash&(frameQ<10'd90)&(frameQ>=10'd60))}}) | (4'hf&{4{HQ>=(inX0) & HQ<=outX0 & VQ<=BPQ0
& VQ>=outY0}}) | (4'hf&{4{HQ>=(inX1) & HQ<=outX1 & VQ<=BPQ1 & VQ>=outY1}}) |
(4'hf&{4{HQ>=(inX2) & HQ<=outX2 & VQ<=BPQ2 & VQ>=outY2}}) | /*(4'h0&{4{HQ>=10'd280 &
HQ<=outX3 & VQ<=BPQ3 & VQ>=outY3}}) | (4'h0&{4{HQ>=10'd280 & HQ<=outX4 & VQ<=BPQ4
& VQ>=outY4}}) |*/ (4'h2&{4{HQ>=(inX5) & HQ<=outX5 & VQ<=BPQ5 & VQ>=outY5}}) |
(4'h8&{4{HQ>=(inX6) & HQ<=outX6 & VQ<=BPQ6 & VQ>=outY6}}));
    assign vgaBlue = ((4'h0&{4{HQ>10'd639 | VQ>10'd479}}) | (4'hf&{4{HQ>=10'd312 &
HQ<=10'd328 & VQ>=10'd392 & VQ<=10'd408 & (~Flash | Flash&(frameQ<10'd30) |
Flash&(frameQ<10'd90)&(frameQ>=10'd60))}}) | /*(4'h0&{4{HQ>=10'd280 & HQ<=outX0 &
VQ<=BPQ0 & VQ>=outY0}}) |*/ (4'h7&{4{HQ>=(inX1) & HQ<=outX1 & VQ<=BPQ1 &
VQ>=outY1}}) | (4'hf&{4{HQ>=(inX2) & HQ<=outX2 & VQ<=BPQ2 & VQ>=outY2}}) |
(4'hf&{4{HQ>=(inX3) & HQ<=outX3 & VQ<=BPQ3 & VQ>=outY3}}) | /*(4'h0&{4{HQ>=10'd280 &
HQ<=outX4 & VQ<=BPQ4 & VQ>=outY4}}) |*/ (4'h2&{4{HQ>=(inX5) & HQ<=outX5 &
VQ<=BPQ5 & VQ>=outY5}}) /*| (4'h0&{4{HQ>=10'd280 & HQ<=outX6 & VQ<=BPQ6 &
VQ>=outY6}})*/);
    assign vgaGreen = ((4'h0&{4{HQ>10'd639 | VQ>10'd479}}) | (4'hf&{4{HQ>=10'd312 &
HQ<=10'd328 & VQ>=10'd392 & VQ<=10'd408 & (~Flash | Flash&(frameQ<10'd30) |
Flash&(frameQ<10'd90)&(frameQ>=10'd60))}}) | /*(4'h0&{4{HQ>=10'd280 & HQ<=outX0 &
VQ<=BPQ0 & VQ>=outY0}}) | (4'h0&{4{HQ>=10'd280 & HQ<=outX1 & VQ<=BPQ1 &
VQ>=outY1}}) | (4'h0&{4{HQ>=10'd280 & HQ<=outX2 & VQ<=BPQ2 & VQ>=outY2}}) |
(4'h0&{4{HQ>=10'd280 & HQ<=outX3 & VQ<=BPQ3 & VQ>=outY3}}) | */(4'hf&{4{HQ>=(inX4) &
HQ<=outX4 & VQ<=BPQ4 & VQ>=outY4}}) | (4'h5&{4{HQ>=(inX5) & HQ<=outX5 & VQ<=BPQ5
& VQ>=outY5}}) | (4'hf&{4{HQ>=(inX6) & HQ<=outX6 & VQ<=BPQ6 & VQ>=outY6}}));
//    &(VQ == 0 & HQ == 0)

    wire [15:0] counterOut;
    counter10UDL gameCounter (.clk(clk), .Up(Run&qsec1), .Reset(reset), .Q(counterOut));
    wire [3:0] sel, H;
    ringCounter RingCounter (.adv(digsel), .clk(clk), .sel(sel));
    selector Selector (.sel(sel), .N(counterOut), .H(H));
    hex7seg hex7seg (.n(H), .seg(seg));
    assign an[3] = ~(sel[3] &(~Flash | Flash&(frameQ<10'd30) |
Flash&(frameQ<10'd90)&(frameQ>=10'd60)));
    assign an[2] = ~(sel[2] &(~Flash | Flash&(frameQ<10'd30) |
Flash&(frameQ<10'd90)&(frameQ>=10'd60)));
    assign an[1] = ~(sel[1] &(~Flash | Flash&(frameQ<10'd30) |
Flash&(frameQ<10'd90)&(frameQ>=10'd60)));
    assign an[0] = ~(sel[0] &(~Flash | Flash&(frameQ<10'd30) |
Flash&(frameQ<10'd90)&(frameQ>=10'd60)));
```

```
   assign dp = 1'b1;
Endmodule


module counter10UDL(
   input Up,
   input Reset,
   input Dw,
   input clk,
   input [9:0] init,
   input LD,
   output [9:0] Q
   );
   wire [4:0] utc;
   wire [4:0] dtc;
//    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]));
   count2UDL count0 (.Up(Up), .Reset(Reset), .Dw(Dw), .clk(clk), .init(init[1:0]), .LD(LD),
.UTC(utc[0]), .Q(Q[1:0]), .DTC(dtc[0]));
   count2UDL count1 (.Up(Up&utc[0]), .Reset(Reset), .Dw(dtc[0]&Dw), .clk(clk), .init(init[3:2]),
.LD(LD), .UTC(utc[1]), .Q(Q[3:2]), .DTC(dtc[1]));
   count2UDL count2 (.Up(Up&utc[1]&utc[0]), .Reset(Reset), .Dw(dtc[0]&dtc[1]&Dw), .clk(clk),
.init(init[5:4]), .LD(LD), .UTC(utc[2]), .Q(Q[5:4]), .DTC(dtc[2]));
   count2UDL count3 (.Up(Up&utc[2]&utc[1]&utc[0]), .Reset(Reset), .Dw(dtc[0]&dtc[1]&dtc[2]&Dw),
.clk(clk), .init(init[7:6]), .LD(LD), .UTC(utc[3]), .Q(Q[7:6]), .DTC(dtc[3]));
   count2UDL count4 (.Up(Up&utc[3]&utc[2]&utc[1]&utc[0]), .Reset(Reset),
.Dw(dtc[0]&dtc[1]&dtc[2]&dtc[3]&Dw), .clk(clk), .init(init[9:8]), .LD(LD), .UTC(utc[4]), .Q(Q[9:8]),
.DTC(dtc[4]));

endmodule

module count2UDL(
   input Up,
   input Reset,
   input Dw,
   input clk,
   input [1:0] init,
   input LD,
   output UTC,
   output DTC,
   output [1:0] Q
   );

   wire [1:0] D;
   assign D[0] = ~LD&(Up^Q[0]^Dw) | LD&init[0];
   assign D[1] = ~LD&(Q[1]^(Up&Q[0])^(Dw&~Q[0])) | LD&init[1];

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

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

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

module storeNum(
    input [9:0] Din,
    input LD,
    input reset,
    input clk,
    output [9:0] Q
    );
    wire [9:0] D;
    assign D = (~{10{LD}} & Q)&(~{10{reset}}) | ({10{LD}} & Din)&(~{10{reset}});

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[3]), .Q(Q[3]));
    FDRE #(.INIT(1'b0)) Q4_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[4]), .Q(Q[4]));
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[5]), .Q(Q[5]));
    FDRE #(.INIT(1'b1)) Q6_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[6]), .Q(Q[6]));
    FDRE #(.INIT(1'b0)) Q7_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[7]), .Q(Q[7]));
    FDRE #(.INIT(1'b1)) Q8_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[8]), .Q(Q[8]));
    FDRE #(.INIT(1'b0)) Q9_FF (.C(clk), .R(reset), .CE(1'b1), .D(D[9]), .Q(Q[9]));

endmodule

module blockSize(
    input [9:0] inX,
    input [9:0] inY,
    input [2:0] width,
    output [9:0] outX,
    output [9:0] outY
    );

//    assign outX = inX+120;
    assign outX = inX+8+(16*width);
    assign outY = inY<80 ? 10'd0 : inY-80;
endmodule

module stateMachine(
    input clk,
    input btnC,
    input TwoSecs,
    input Dead,
    output init,
```

```
    output CountDown,
    output Run,
    output Flash,
    output reset
    );
    wire [4:0] D;
    wire [4:0] Q;

    assign D[0] = (Q[0]&~btnC);
    assign D[1] = (Q[0]&btnC) | (Q[1]&~TwoSecs) | (Q[3]&btnC);
    assign D[2] = (Q[1]&TwoSecs) | (Q[2]&~Dead);
    assign D[3] = (Q[2]&Dead) | (Q[3]&~btnC);

    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]));

    assign init = Q[0];
    assign CountDown = Q[1];
    assign Run = Q[2];
    assign Flash = Q[3] | Q[1];
    assign reset = Q[3] & btnC;
endmodule


module RingCounterLed(
    input adv,
    input clk,
    input tick,
    input reset,
    output [3:0] sel
    );

    wire [3:0] Q;

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

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

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 :