Aidan Smith

11/22/19

Section 01D

Lab 6 Report

Description:

For this lab we had to make a turkey counter. The program would count turkeys passing by, adding or subtracting one for each turkey passing based on its direction. Since we did not use sensors, we used buttons to emulate a turkey blocking one sensor. We had to create a state diagram to represent the states the turkey could be in while crossing.

Methods:

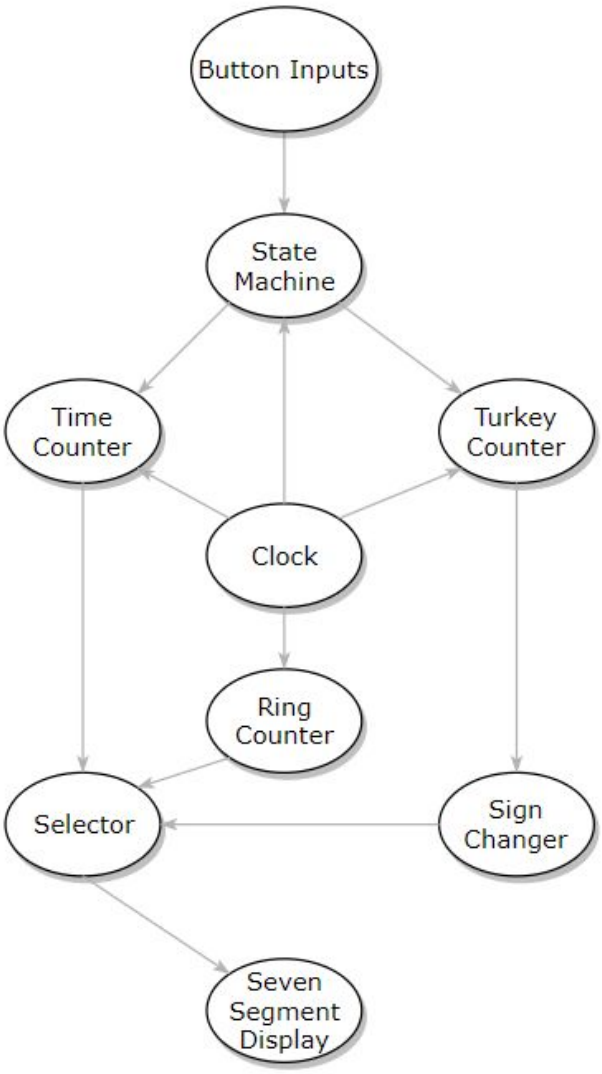
To create this program I had to create a state machine for the turkey's states while crossing the sensors. This was the center of the lab, and everything else had to work around it. The primary inputs are the buttons L and R, the "sensors" that the turkeys are crossing. Using our state diagram logic, we can determine if one passes from left to right, right to left, or turns around before making a complete pass. From being able to determine what the turkey is doing, we can use a counter to keep track of how many more turkeys passed from one direction than the other. Our seven segment display would display the turkey count, a range of values from -127 to 127, and the time that a turkey has spent in one state that includes blocking a sensor. Once the turkey has blocked the sensor for 4 seconds, the timer stops counting and flashes until it moves. LEDs 8 and 15 would also go on corresponding to the right or left button being pressed down.

Results:

Breakdown of section

- **Lab6** (Lab6.v) (9)
  - slowit : lab6_clks (lab6_clks.v) (2)
  - StateMachine : StateMachine (StateMachine.v) (1)
    - logic : StateLogic (StateLogic.v)
  - timeCount : countUD16L (countUD16L.v) (4)
    - count3_0 : countUD3L (countUD3L.v)
    - count5_1 : countUD5L (countUD5L.v)
    - count5_2 : countUD5L (countUD5L.v)
    - count3_3 : countUD3L (countUD3L.v)
  - turkeyCount : countUD16L (countUD16L.v) (4)
  - signPicker : SignChanger (SignChanger.v) (9)
    - add0 : Adder (Adder.v) (2)
      - mux4sum : m4_1e (m4_1e.v)
      - mux4carry : m4_1e (m4_1e.v)
    - add1 : Adder (Adder.v) (2)
    - add2 : Adder (Adder.v) (2)
    - add3 : Adder (Adder.v) (2)
    - add4 : Adder (Adder.v) (2)
    - add5 : Adder (Adder.v) (2)
    - add6 : Adder (Adder.v) (2)
    - add7 : Adder (Adder.v) (2)
    - mux2pick : m2_1x8 (m2_1x8.v)
  - ring : ring (ring.v)
  - sel1 : selector (selector.v)
  - qFlash : counter4L (counter4L.v)
  - seg1 : hex7seg (hex7seg.v) (7)
    - mux8A : m8_1e (m8_1e.v)
    - mux8B : m8_1e (m8_1e.v)
    - mux8C : m8_1e (m8_1e.v)
    - mux8D : m8_1e (m8_1e.v)
    - mux8E : m8_1e (m8_1e.v)
    - mux8F : m8_1e (m8_1e.v)
    - mux8G : m8_1e (m8_1e.v)

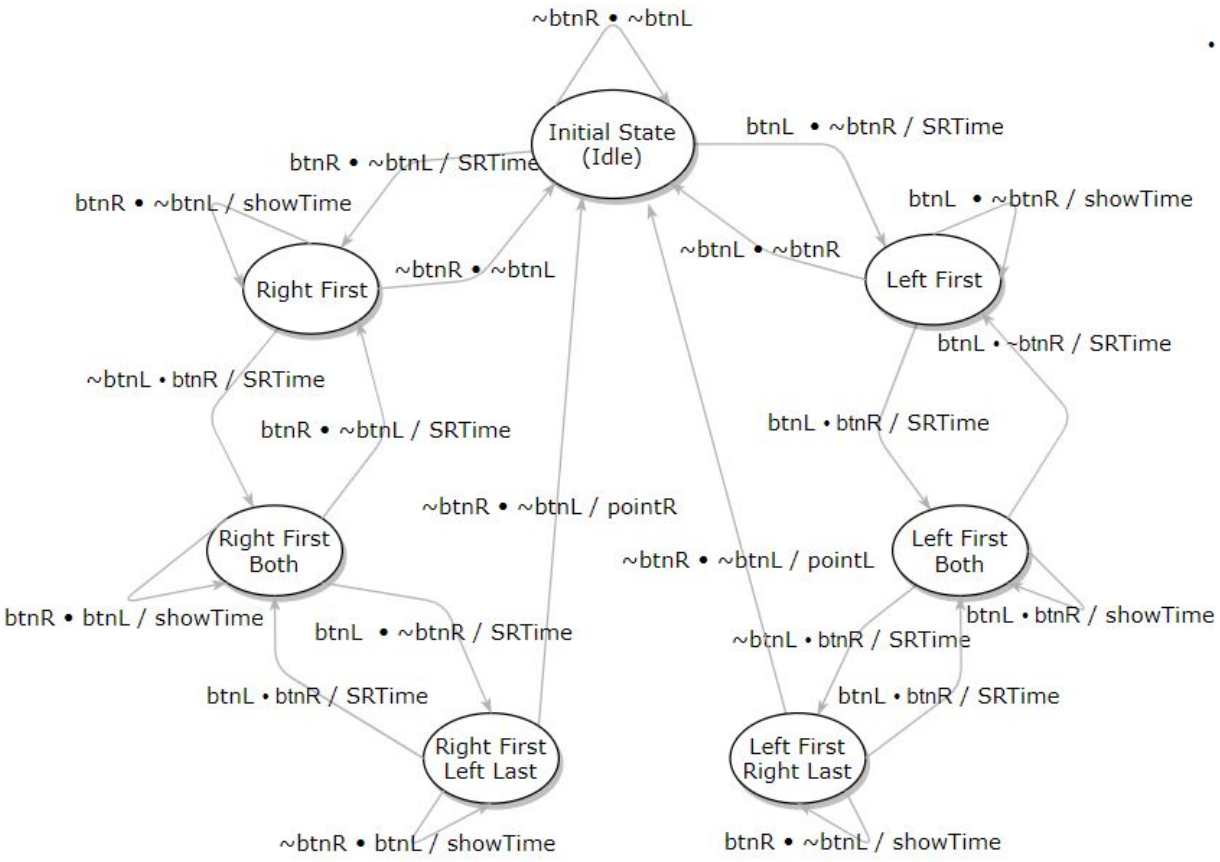
Diagram of top level



The button inputs tell whether a turkey has is passing or has passed. This is interpreted in the state machine. The state machine contains logic for changing states, and has outputs based on its current state transitions, as a Mealy machine. Each time a turkey enters the sensor, the timer counts up using a counter. This is sent to the selector as the 4 bit number to be printed on the leftmost display. The state machine also has an output to the turkey counter, a 1 if it goes from

right to left, and a -1 if it goes from left to right. The ring counter is used to cycle through the 4 displays so that they are all displayed seemingly constantly. The clock is used to regulate the state machine, both counters, and the ring counter. Selector takes in the 4 different things to be displayed and then takes in the output from the ring counter, which tells it what to pick. It picks and then sends what to show to the seven hexadecimal display.

I tested my design by using a simulation for the entire design, as a lot of the design was from previous labs. Once I felt good about it, I tested it on the board and fixed a few errors. I chose the inputs given, there weren't any special inputs for this lab. Some corner cases that I had to consider was if a turkey went in but turned around, this was covered by the state machine. I ran into two problems: I assigned an output two separate values, and my score display was inverted upon first testing.



Acronyms to make the equations shorter:

Right = R	Left = L	Both = B	First = F	Last = L

So Right First Left Last = RFLL

Next State:

Idle = ~btnR • ~ btnL • (Idle + RF + LF + RFLL + LFRL)

RF = btnR • ~btnL • (Idle + RF + RFB)

RFB = btnR • btnL • (RF + RFB + RFLL)

RFLL = ~btnR • btnL • (RFB + RFLL)

LF = ~btnR • btnL • (Idle + LF + LFB)

LFB = btnR • btnL • (LF + LFB + LFRL)

LFRL = btnR • ~btnL • (LFB + LFRL)

Output Logic:

pointL = btnR • ~btnL • LFRL

pointR = ~btnR • btnL • RFLL

SRTime = Idle + (btnR ⊕ btnL) • (LFB + RFB) + btnR • btnL • (RF + RFLL + LF + LFRF)

showTime = ~Idle

States:

Idle: This is the initial state. In this state no sensors are blocked, and it will stay in this state until one is blocked. No outputs.

Right First: This state is entered from this idle state if the right sensor is blocked first. The timer starts as soon as it switches to this state, and while it is in the state the timer is shown. If the turkey turns around then it returns to idle, but if it continues then we move to the next state.

Right First Both: This is the next state from Right First. This is entered when both sensors are blocked after the right one being blocked first. The timer is restarted and shown. This state returns to Right First if the left sensor is unblocked, but if it continues then we move on.

Right First Left Last: This state is entered from Right First Both if the turkey continues moving straight, now unblocking the right sensor. The timer is restarted and shown. This state returns to Right First Both if both sensors are blocked again, or to idle if the turkey crosses all the way. A point is added to the turkey count, the total is always shown.

Left First: This state is entered from this idle state if the left sensor is blocked first. The timer starts as soon as it switches to this state, and while it is in the state the timer is shown. If the turkey turns around then it returns to idle, but if it continues then we move to the next state.

Left First Both: This is the next state from Left First. This is entered when both sensors are blocked after the left one being blocked first. The timer is restarted and shown. This state returns to Left First if the right sensor is unblocked, but if it continues then we move on.

Left First Right Last: This state is entered from Left First Both if the turkey continues moving straight, now unblocking the left sensor. The timer is restarted and shown. This state returns to Left First Both if both sensors are blocked again, or to idle if the turkey crosses all the way. A point is subtracted from the turkey count, the total is always shown.

Components:

State Machine: New, tracks the state that the turkey is in to properly count how many turkeys pass and in what direction.

16 Bit Counter: Unchanged. Used to both count the number of turkeys passed and the time that the current turkey is taking.

Sign Changer: Unchanged. Used to determine if the number of turkeys passed is positive or negative, in order to display the correct sign.

Ring: Unchanged.

Selector: Unchanged.

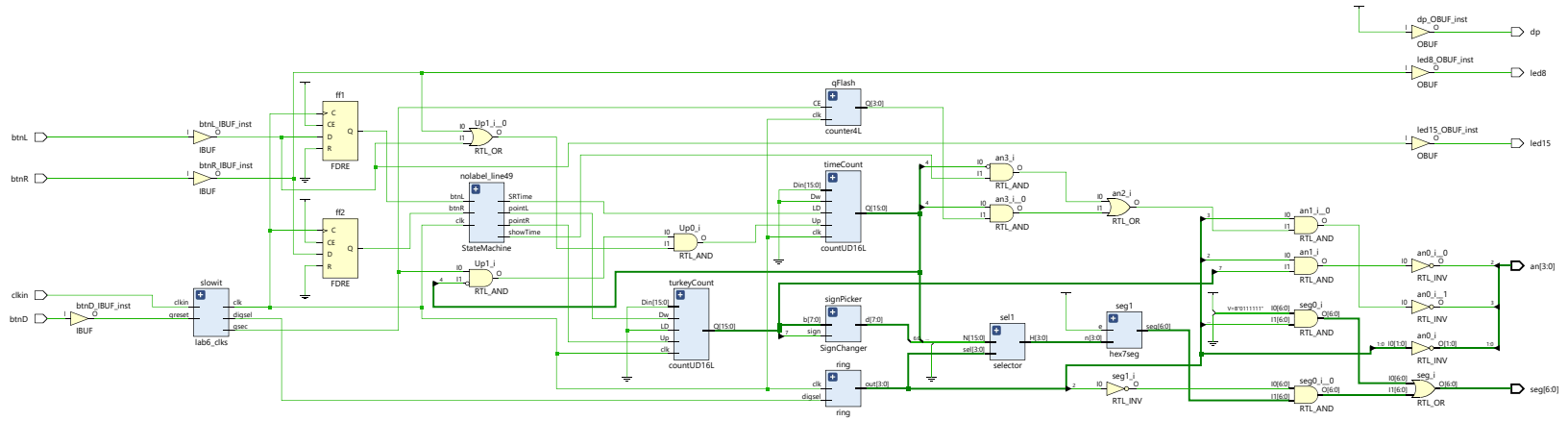
qFlash: Uses 4-bit counter, unique because it counts qsec. Used to count up to 4 seconds and then cause the display to flash.

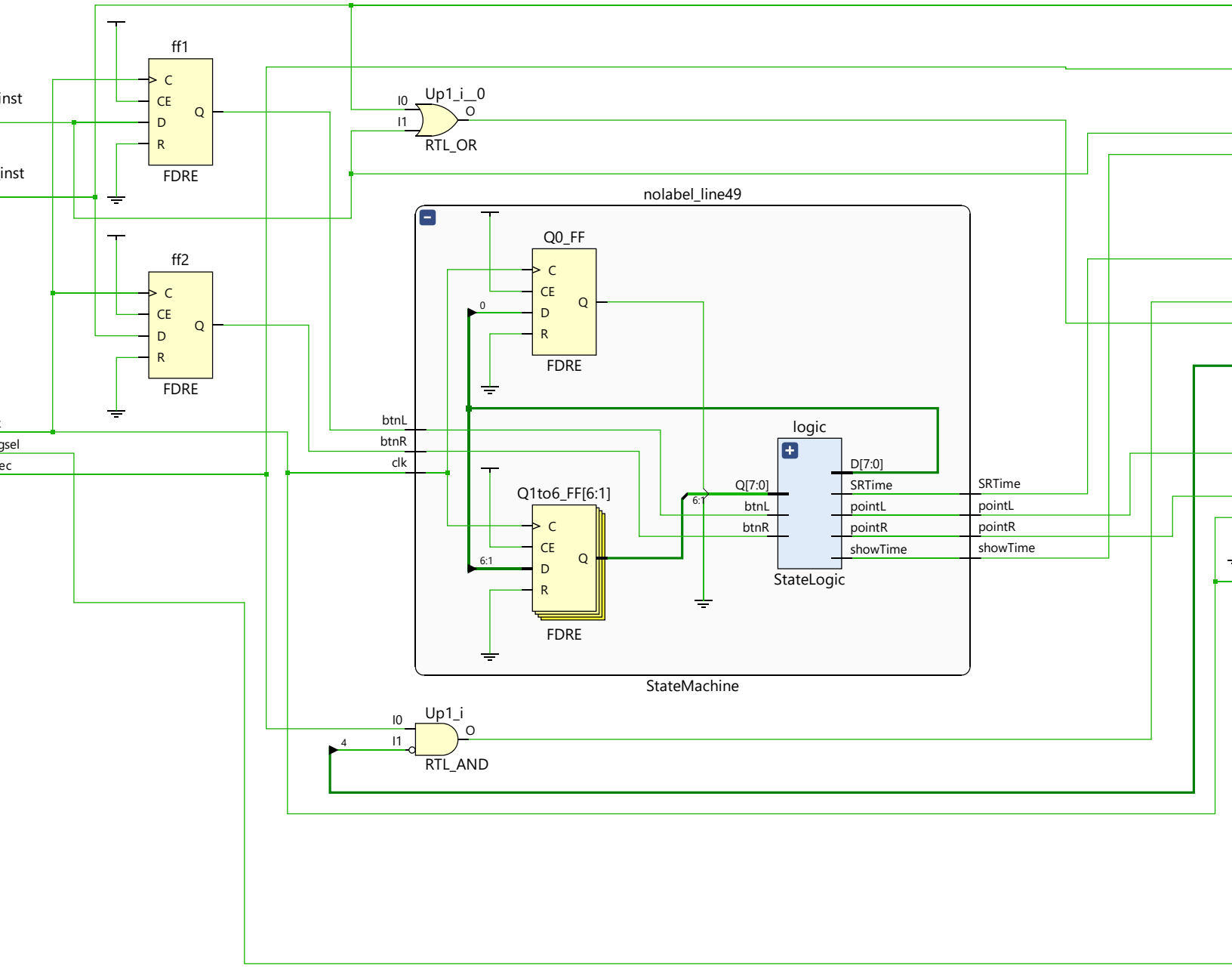
7 Segment Display: Unchanged.

All counters, **muxes**, and the **Adder** are unchanged and work to aid other functions.

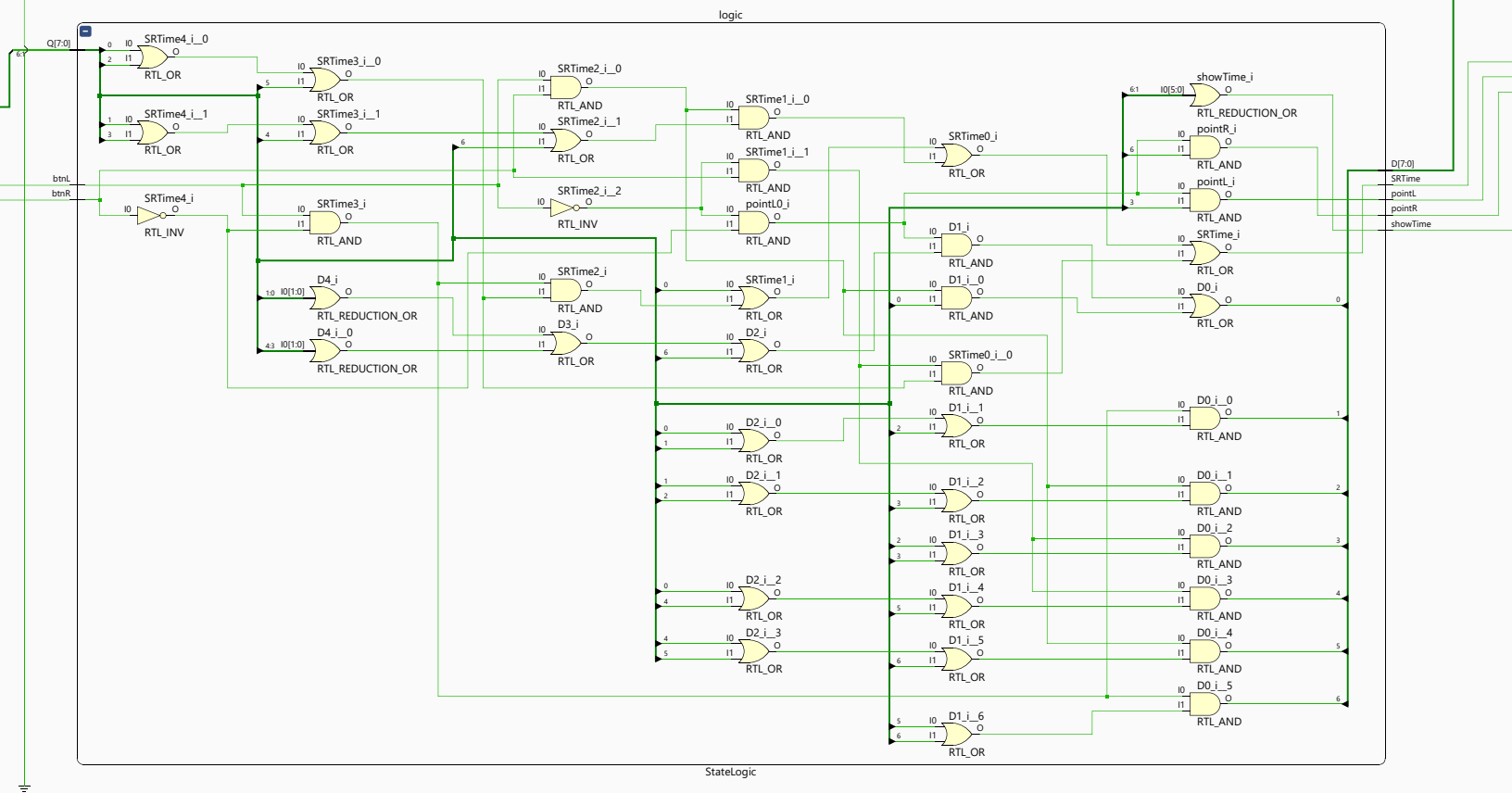
Conclusion:

I learned how to choose an output for the seven hex display that was not defined in selector. To display the negative sign we needed to create extra logic outside of selector because selector could only output numbers. I had trouble getting the negative sign to show, but after adding a check if I'm showing anode 2, which is the sign anode, then it should show a negative if it is a negative number. If I did this lab again, I would make sure that I didn't accidentally give an output two assigns, this hindered my testing for a couple of days. I would optimize my counters, they're both 16 bit but my time counter only needs to be 4 bits and my score counter only needs to be 8 bits.


btnL
btnR
clkin
btnD
btnL_IBUF_inst
btnR_IBUF_inst
btnD_IBUF_inst
IBUF
ff1
ff2
FDRE
slowit
clkin
greset
clk
digsel
qsec
lab6_clks
Up1_i_0
RTL_OR
Up1_i
RTL_AND
Up0_i
nolabel_line49
btnL
btnR
clk
SRTime
pointL
pointR
showTime
StateMachine
qFlash
CE
clk
Q[3:0]
counter4L
timeCount
Din[15:0]
Dw
LD
Up
Q[15:0]
countUD16L
turkeyCount
signPicker
b[7:0]
sign
d[7:0]
SignChanger
ring
out[3:0]
an3_i
an3_i_0
an2_i
sel1
N[15:0]
sel[3:0]
H[3:0]
selector
seg1
n[3:0]
seg[6:0]
hex7seg
seg1_i
an1_i_0
an1_i
an0_i_0
an0_i_1
an0_i
seg0_i
seg0_i_0
RTL_INV
RTL_AND
dp_OBUF_inst
led8_OBUF_inst
led15_OBUF_inst
OBUF
dp
led8
led15
an[3:0]
seg[6:0]


ff1
C
CE
D
R
Q
FDRE
ff2
C
CE
D
R
Q
FDRE
Up1_i__0
I0
I1
O
RTL_OR
nolabel_line49
Q0_FF
C
CE
D
R
Q
0
FDRE
btnL
btnR
clk
Q1to6_FF[6:1]
C
CE
D
R
Q
6:1
FDRE
logic
Q[7:0]
6:1
btnL
btnR
D[7:0]
SRTime
pointL
pointR
showTime
StateLogic
SRTime
pointL
pointR
showTime
StateMachine
Up1_i
I0
I1
O
4
RTL_AND

nolabel_line49



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/12/2019 01:48:39 PM
// Design Name:
// Module Name: Lab6
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Lab6(
    input btnL,
    input btnR,
    input btnD, // (greset)
    input clkin,

    output led8,
    output led15,
    output [3:0] an,
    output dp,
    output [6:0] seg
    );
```

```
    assign dp = 1'b1; // never want dp on :(

    wire clk, digsel, qsec;

    lab6_clks slowit (.clkin(clkin), .greset(btnD), .clk(clk),
.digsel(digsel), .qsec(qsec));

    wire left, right;

    FDRE #(.INIT(1'b0) ) ff1 ( .C(clk), .R(1'b0), .CE(1'b1),
.D(btnL), .Q(left) ); // synchronize the buttons
    FDRE #(.INIT(1'b0) ) ff2 ( .C(clk), .R(1'b0), .CE(1'b1),
.D(btnR), .Q(right) );

    wire showTime, SRTime, pointL, pointR;

    StateMachine( .clk(clk), .btnL(left), .btnR(right),
.showTime(showTime), .SRTime(SRTime), .pointL(pointL),
.pointR(pointR) );

    wire [7:0] useless, timer; // timer for lower 8 bits, useless
to satisfy machine output (top 8 bits)

    countUD16L timeCount( .clk(clk), .Up(qsec & ~timer[4] & (btnR |
btnL)), .Dw(1'b0), .LD(SRTime), .Din(16'b0000000000000000), .Q(
{useless, timer} ) );

    wire [7:0] temp, value; // temp is meaningless, value has the
MSB = sign and other bits are the value
    countUD16L turkeyCount( .clk(clk), .Up(pointR), .Dw(pointL),
.LD(1'b0), .Din(16'b0000000000000000), .Q( {temp, value} ) );

    // logic using sign changer to make an[2]: G = 1. MSB of
    wire [6:0] final; // holds the 7 bit final value
    wire meaningless; // holds the MSB (meaningless)

    wire sign;
```

```
    assign sign = value[7];

    SignChanger signPicker( .sign(sign), .b(value), .d(
{meaningless, final} ) );

    assign led8 = btnR;
    assign led15 = btnL;

    wire [15:0] Q; // Q is the bits that are going to be used to
select and then show
    wire [3:0] sel, n; // sel is selector 4 bits, n is chosen 4
bits of Q

    assign Q = { timer[5:2], 4'b0000, {1'b0, final} };

    ring ring( .digsel(digsel), .clk(clk), .out(sel) );

    selector sel1( .sel(sel), .N(Q), .H(n) );

    wire [3:0] qF; // holds flashing signal
    wire flash = timer[4]; // allows flash signal

    counter4L qFlash( .clk(clk), .CE(qsec), .Q(qF) ); // qF[1]
causes a flash every half second

    assign an[3] = ~( sel[3] & ( (~flash & showTime) | (flash &
qF[1]) ) ); // timer
    assign an[2] = ~(sel[2] & sign ); // sign
    assign an[1:0] = ~sel[1:0]; // score

    wire [6:0] segTemp, neg;
    assign neg = 7'b0111111; // 1 000 000 (makes G = 1)
//    assign neg = 7'b1000000; // 1 000 000 (makes G = 1)

    hex7seg seg1( .n(n), .e(1'b1), .seg(segTemp) );

//    assign seg = ~( ( ~neg & {7{sel[2]}} ) | ~( {7{~sel[2]}} &
```

```
segTemp ) );
    assign seg = ( ( neg & {7{sel[2]}} ) | ( {7{~sel[2]}} & segTemp
) );

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/05/2019 04:33:07 PM
// Design Name:
// Module Name: StateMachine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module StateMachine(
    input clk,
    input btnL,
    input btnR,

    output showTime,
    output SRTime,
    output pointL,
    output pointR
    );

    wire [6:0] Q;
    wire [6:0] D;
```

```
    StateLogic  logic( .btnL(btnL), .btnR(btnR), .Q(Q),
.showTime(showTime), .SRTime(SRTime), .pointL(pointL),
.pointR(pointR), .D(D) );

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(1'b1), .D(D[0]),
.Q(Q[0])); // greset gives 000 0001 (start in idle state)
    FDRE #(.INIT(1'b0)) Q1to6_FF[6:1] (.C({6{clk}}),
.CE({6{1'b1}}), .D(D[6:1]), .Q(Q[6:1]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/05/2019 04:57:33 PM
// Design Name:
// Module Name: StateLogic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module StateLogic(
    input btnL,
    input btnR,
    input [7:0] Q,

    output showTime,
    output SRTime,
    output pointL,
    output pointR,
    output [7:0] D
    );

    // NEXT STATE
```

```
    assign D[6] = btnL & ~btnR & (Q[5] | Q[6]); // Right First Left
Last (RFLL)
    assign D[5] = btnL & btnR & (Q[4] | Q[5] | Q[6]); // Right
First Both (RFB)
    assign D[4] = ~btnL & btnR & (Q[0] | Q[4] | Q[5]); // Right
First (RF)
    assign D[3] = ~btnL & btnR & (Q[2] | Q[3]); // Left First Right
Last (LFRL)
    assign D[2] = btnL & btnR & (Q[1] | Q[2] | Q[3]); // Left First
Both (LFB)
    assign D[1] = btnL & ~btnR & (Q[0] | Q[1] | Q[2]); // Left
First (LF)
    assign D[0] = ( ~btnL & ~btnR & ( ( |Q[1:0] ) | ( |Q[4:3] ) |
Q[6] ) ) | ( btnL & btnR & Q[0] ); // idle


    // OUTPUTS
    assign showTime = ( |Q[6:1] );
//    assign SRTime = ( btnL & ~btnR & ( Q[0] | Q[2] | Q[5] ) ) | (
btnL & btnR & ( Q[1] | Q[3] | Q[4] | Q[6] ) ) | ( ~btnL & btnR & (
Q[0] | Q[2] | Q[5] ) );
    assign SRTime = Q[0] | ( btnL & ~btnR & ( Q[0] | Q[2] | Q[5] )
) | ( btnL & btnR & ( Q[1] | Q[3] | Q[4] | Q[6] ) ) | ( ~btnL &
btnR & ( Q[0] | Q[2] | Q[5] ) );

    assign pointL = ~btnL & ~btnR & Q[3];
    assign pointR = ~btnL & ~btnR & Q[6];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/24/2019 03:05:17 PM
// Design Name:
// Module Name: countUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD16L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [15:0] Din, // only for load enabled

    output [15:0] Q,
    output UTC,
    output DTC
    );

    wire [3:0] outUTC, outDTC;
```

```
    wire [15:0] tempQ;

    countUD3L count3_0(.clk(clk), .Up(Up), .Dw(Dw), .LD(LD),
.Din(Din[2:0]), .Q(tempQ[2:0]), .UTC(outUTC[0]), .DTC(outDTC[0]));
    countUD5L count5_1(.clk(clk), .Up(outUTC[0]&Up&~LD),
.Dw(outDTC[0]&Dw), .LD(LD), .Din(Din[7:3]), .Q(tempQ[7:3]),
.UTC(outUTC[1]), .DTC(outDTC[1]));
    countUD5L count5_2(.clk(clk), .Up((&outUTC[1:0])&Up&~LD),
.Dw((&outDTC[1:0])&Dw&~LD), .LD(LD), .Din(Din[12:8]),
.Q(tempQ[12:8]), .UTC(outUTC[2]), .DTC(outDTC[2]));
    countUD3L count3_3(.clk(clk), .Up((&outUTC[2:0])&Up&~LD),
.Dw((&outDTC[2:0])&Dw&~LD), .LD(LD), .Din(Din[15:13]),
.Q(tempQ[15:13]), .UTC(outUTC[3]), .DTC(outDTC[3]));

    assign UTC = &outUTC;
    assign DTC = &outDTC;
    assign Q = tempQ;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2019 03:45:06 PM
// Design Name:
// Module Name: countUD5L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD5L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [4:0] Din, // only for load enabled

    output [4:0] Q,
    output UTC,
    output DTC
    );

    wire enable;
```

```
    assign enable = Up | Dw | LD; // if all on then load overrides.
up/dw cancel out
    // ~enable = ~Up & ~Dw & ~LD;

    wire [4:0] D;

    FDRE #(.INIT(1'b0) ) ff0 (.C(clk), .R(1'b0), .CE(enable),
.D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .R(1'b0), .CE(enable),
.D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .R(1'b0), .CE(enable),
.D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0) ) ff3 (.C(clk), .R(1'b0), .CE(enable),
.D(D[3]), .Q(Q[3]));
    FDRE #(.INIT(1'b0) ) ff4 (.C(clk), .R(1'b0), .CE(enable),
.D(D[4]), .Q(Q[4]));

    assign D[0] = (~Q[0] & (Up ^ Dw) & ~LD) | (Din[0] & LD) | (Q[0]
& ~enable);
    assign D[1] = ((Q[1] ^ (Up & Q[0])) & ~Dw & ~LD) | ((Q[1] ^ (Dw
& ~Q[0])) & ~Up & ~LD) | (Din[1] & LD) | (Q[1] & ~enable);
    assign D[2] = ((Q[2] ^ (Up & Q[1] & Q[0])) & ~Dw & ~LD) |
((Q[2] ^ (Dw & ~Q[1] & ~Q[0])) & ~Up & ~LD) | (Din[2] & LD) | (Q[2]
& ~enable);
    assign D[3] = ((Q[3] ^ (Up & Q[2] & Q[1] & Q[0])) & ~Dw & ~LD)
| ((Q[3] ^ (Dw & ~Q[2] & ~Q[1] & ~Q[0])) & ~Up & ~LD) | (Din[3] &
LD) | (Q[3] & ~enable);
    assign D[4] = ((Q[4] ^ (Up & Q[3] & Q[2] & Q[1] & Q[0])) & ~Dw
& ~LD) | ((Q[4] ^ (Dw & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0])) & ~Up &
~LD) | (Din[4] & LD) | (Q[4] & ~enable);

    assign UTC = &Q;
    assign DTC = &(~Q);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2019 03:45:06 PM
// Design Name:
// Module Name: countUD5L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module counter4L(
    input clk,
    input CE,

    output [3:0] Q
    );

    wire [3:0] D;

    FDRE #(.INIT(1'b0) ) ff0 (.C(clk), .R(1'b0), .CE(CE), .D(D[0]),
.Q(Q[0]));
    FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .R(1'b0), .CE(CE), .D(D[1]),
.Q(Q[1]));
```

```
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .R(1'b0), .CE(CE), .D(D[2]),
.Q(Q[2]));
    FDRE #(.INIT(1'b0) ) ff3 (.C(clk), .R(1'b0), .CE(CE), .D(D[3]),
.Q(Q[3]));

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

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2019 03:44:43 PM
// Design Name:
// Module Name: countUD3L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD3L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [2:0] Din, // only for load enabled

    output [2:0] Q,
    output UTC,
    output DTC
    );

    wire enable;
```

```
    assign enable = Up | Dw | LD; // if all on then load overrides.
up/dw cancel out
    // ~enable = ~Up & ~Dw & ~LD;

    wire [2:0] D;

    FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .R(1'b0), .CE(enable),
.D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .R(1'b0), .CE(enable),
.D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0) ) ff3 (.C(clk), .R(1'b0), .CE(enable),
.D(D[2]), .Q(Q[2]));

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

    assign UTC = &Q;
    assign DTC = &(~Q);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/15/2019 03:33:43 PM
// Design Name:
// Module Name: m8_1e
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m8_1e(
    input [7:0] in,
    input [2:0] sel,
    input e, // enable

    output o
    );

    wire [7:0] temp;

    assign temp[0] = in[0] & ~sel[2] & ~sel[1] & ~sel[0]; // 000
    assign temp[1] = in[1] & ~sel[2] & ~sel[1] & sel[0];  // 001
    assign temp[2] = in[2] & ~sel[2] & sel[1] & ~sel[0];  // 010
```

```
    assign temp[3] = in[3] & ~sel[2] & sel[1] & sel[0];   // 011
    assign temp[4] = in[4] & sel[2] & ~sel[1] & ~sel[0];  // 100
    assign temp[5] = in[5] & sel[2] & ~sel[1] & sel[0];   // 101
    assign temp[6] = in[6] & sel[2] & sel[1] & ~sel[0];   // 110
    assign temp[7] = in[7] & sel[2] & sel[1] & sel[0];    // 111

    assign o = (|temp) & e;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/15/2019 03:33:19 PM
// Design Name:
// Module Name: m4_1e
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m4_1e(
    input [3:0] in,
    input [1:0] sel,
    input e,

    output o
    );

    wire [3:0] temp;

    assign temp[0] = in[0] & ~sel[1] & ~sel[0];
    assign temp[1] = in[1] & ~sel[1] & sel[0];
    assign temp[2] = in[2] & sel[1] & ~sel[0];
```

```
    assign temp[3] = in[3] & sel[1] & sel[0];

    assign o = (|temp) & e;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/15/2019 03:33:55 PM
// Design Name:
// Module Name: m2_1x8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:

//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m2_1x8(
    input [7:0] in0, // normal, sel = 0
    input [7:0] in1, // complement, sel = 1
    input sel,

    output [7:0] o
    );

    wire [7:0] a;

    assign a = {8{sel}};
```

```
    assign o = (in0 & ~a) | (in1 & a);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/15/2019 03:38:50 PM
// Design Name:
// Module Name: SignChanger
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module SignChanger(
    input sign,
    input [7:0] b,

    output [7:0] d
    );

    wire [7:0] c, comp; // c = carry in/outs, comp = 2's comp

    Adder add0(.a(~b[0]), .cin(sign), .b(1'b0), .o(c[0]),
.s(comp[0]));
    Adder add1(.a(~b[1]), .cin(c[0]), .b(1'b0), .o(c[1]),
.s(comp[1]));
```

```
    Adder add2(.a(~b[2]), .cin(c[1]), .b(1'b0), .o(c[2]),
.s(comp[2]));
    Adder add3(.a(~b[3]), .cin(c[2]), .b(1'b0), .o(c[3]),
.s(comp[3]));
    Adder add4(.a(~b[4]), .cin(c[3]), .b(1'b0), .o(c[4]),
.s(comp[4]));
    Adder add5(.a(~b[5]), .cin(c[4]), .b(1'b0), .o(c[5]),
.s(comp[5]));
    Adder add6(.a(~b[6]), .cin(c[5]), .b(1'b0), .o(c[6]),
.s(comp[6]));
    Adder add7(.a(~b[7]), .cin(c[6]), .b(1'b0), .o(c[7]),
.s(comp[7])); // last carry out doesnt matter

    m2_1x8 mux2pick( .in0(b), .in1(comp), .sel(sign), .o(d) ); //
controls whether 2's comp or normal is returned

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/15/2019 05:02:49 PM
// Design Name:
// Module Name: Adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Adder(
    input a,
    input cin,
    input b,

    output o, // carryout
    output s // sum
    );

    m4_1e mux4sum( .in({b,~b,~b,b}), .sel({a,cin}), .e(1), .o(s) );
    m4_1e mux4carry( .in({1'b1,b,b,1'b0}), .sel({a,cin}), .e(1),
.o(o) );
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/24/2019 04:06:57 PM
// Design Name:
// Module Name: ring
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ring(
    input digsel,
    input clk,

    output [3:0] out
    );

    wire [3:0] rcount;

    FDRE #(.INIT(1'b1) ) ff1 (.C(clk), .R(1'b0), .CE(digsel),
.D(rcount[3]), .Q(rcount[0])); // initalizied to 1
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .R(1'b0), .CE(digsel),
.D(rcount[0]), .Q(rcount[1]));
```

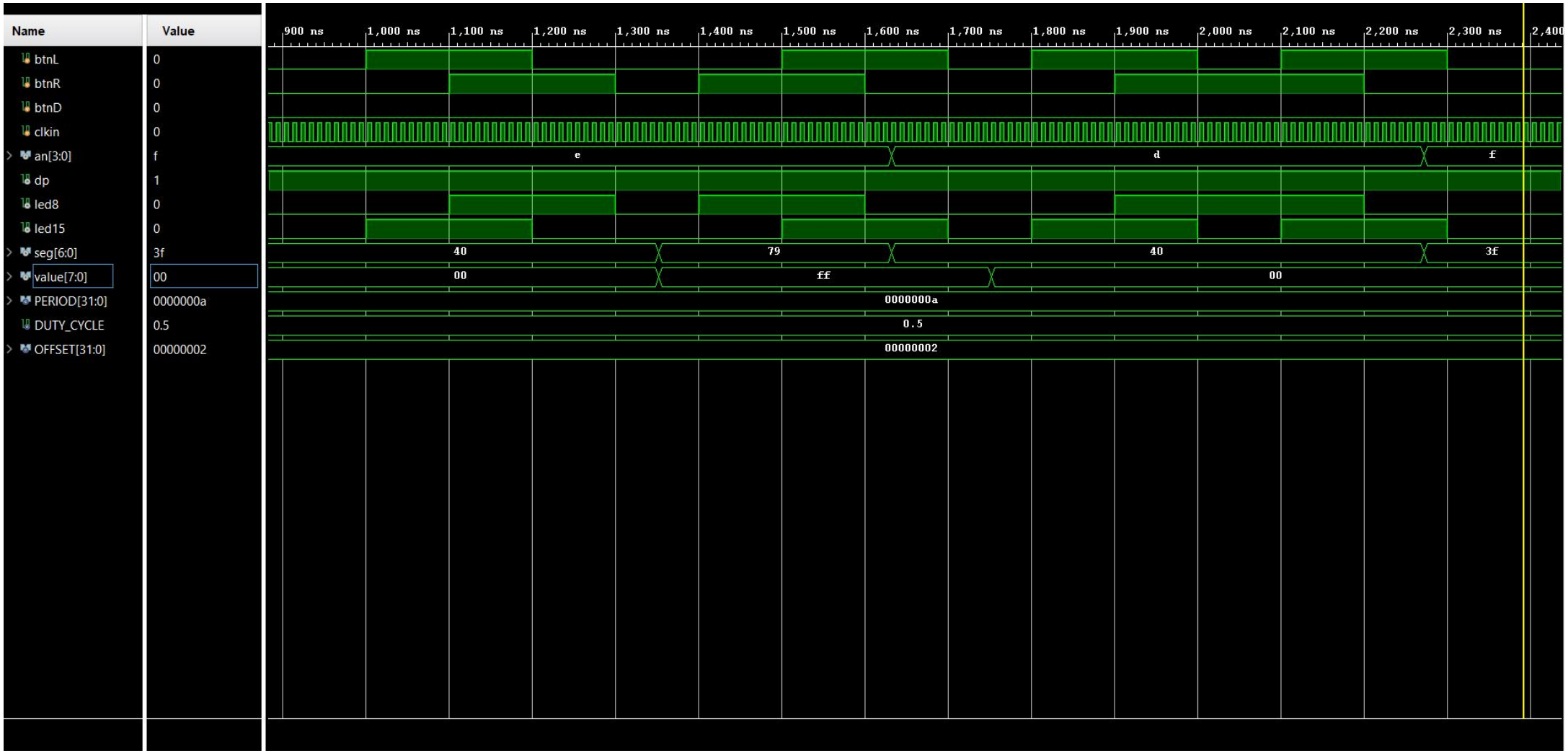
```
    FDRE #(.INIT(1'b0) ) ff3 (.C(clk), .R(1'b0), .CE(digsel),
.D(rcount[1]), .Q(rcount[2]));
    FDRE #(.INIT(1'b0) ) ff4 (.C(clk), .R(1'b0), .CE(digsel),
.D(rcount[2]), .Q(rcount[3]));

    assign out = rcount;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/24/2019 03:40:36 PM
// Design Name:
// Module Name: selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module selector(
    input [3:0] sel,
    input [15:0] N,

    output [3:0] H
    );
    //H is x[15:12] when sel=(1000)
    //H is x[11:8] when sel=(0100)
    //H is x[7:4] when sel=(0010)
    //H is x[3:0] when sel=(0001)

    wire fail;
    assign fail = (sel[0] & (|sel[3:1])) | (sel[1] & (|sel[3:2])) |
```

```
(sel[2] & sel[3]); // more than one is on

    assign H = (N[15:12] & {4{sel[3] & ~fail}}) | (N[11:8] &
{4{sel[2] & ~fail}}) | (N[7:4] & {4{sel[1] & ~fail}}) | (N[3:0] &
{4{sel[0] & ~fail}});

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/16/2019 06:41:25 PM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module hex7seg(
    input [3:0] n,
    input e, // enable

    output [6:0] seg
    );

    m8_1e mux8A( .in({1'b0, n[0], n[0], 1'b0, 1'b0, ~n[0], 1'b0,
n[0]}), .sel({n[3], n[2], n[1]}), .e(e), .o(seg[0])); // A
    m8_1e mux8B( .in({1'b1, ~n[0], n[0], 1'b0, ~n[0], n[0], 1'b0,
1'b0}), .sel({n[3], n[2], n[1]}), .e(e), .o(seg[1])); // B
    m8_1e mux8C( .in({1'b1, ~n[0], 1'b0, 1'b0, 1'b0, 1'b0, ~n[0],
1'b0}), .sel({n[3], n[2], n[1]}), .e(e), .o(seg[2])); // C
```

```
    m8_1e mux8D( .in({n[0], 1'b0, ~n[0], n[0], n[0], ~n[0], 1'b0,
n[0]}), .sel({n[3], n[2], n[1]}), .e(e), .o(seg[3])); // D
    m8_1e mux8E( .in({1'b0, 1'b0, 1'b0, n[0], n[0], 1'b1, n[0],
n[0]}), .sel({n[3], n[2], n[1]}), .e(e), .o(seg[4])); // E
    m8_1e mux8F( .in({1'b0, n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b1,
n[0]}), .sel({n[3], n[2], n[1]}), .e(e), .o(seg[5])); // F
    m8_1e mux8G( .in({1'b0, ~n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b0,
1'b1}), .sel({n[3], n[2], n[1]}), .e(e), .o(seg[6])); // G

endmodule
```

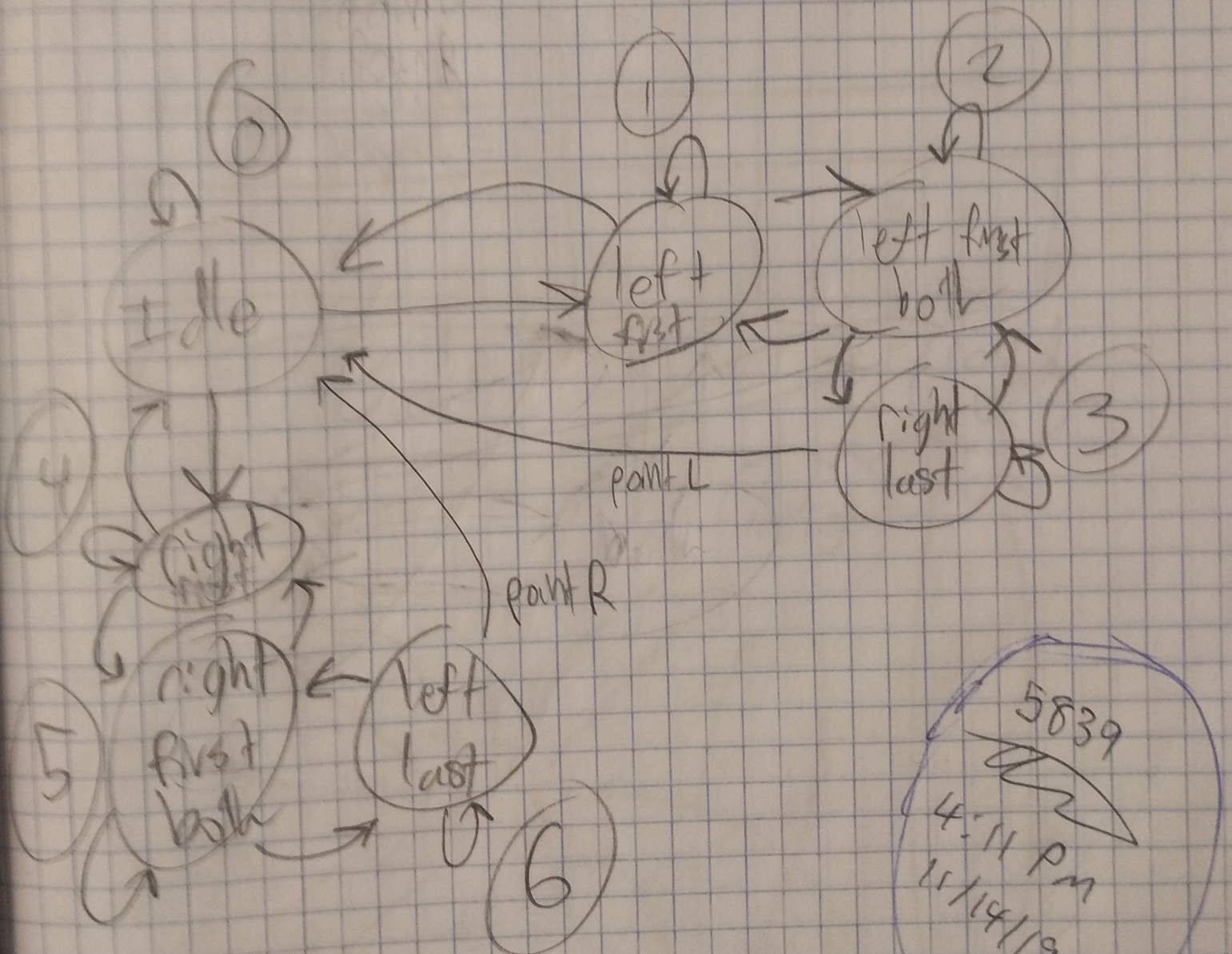

Name
Value
btnL
0
btnR
0
btnD
0
clkin
0
an[3:0]
f
dp
1
led8
0
led15
0
seg[6:0]
3f
value[7:0]
00
PERIOD[31:0]
0000000a
DUTY_CYCLE
0.5
OFFSET[31:0]
00000002
900 ns
1,000 ns
1,100 ns
1,200 ns
1,300 ns
1,400 ns
1,500 ns
1,600 ns
1,700 ns
1,800 ns
1,900 ns
2,000 ns
2,100 ns
2,200 ns
2,300 ns
e
d
f
40
79
40
3f
00
ff
00
0000000a
0.5
00000002

Lab 6

11/12/19
3:10 PM

Inputs
startL
startR

Outputs
paintL
paintR
SR Time start/restart timer
stop time



5839
4:11 PM
11/14/19

Lab 6 – States

0 Idle = $\overline{btnL} \cdot \overline{btnR} \cdot (idle + LFRL + RFLL)$

1 left first = $btnL \cdot \overline{btnR} \cdot idle + LF \cdot btnL \cdot \overline{}$

2 left first both = $LF \cdot btnL \cdot btnR + LFB \cdot btnL \cdot btnR$

3 Left first Right last = $LFB \cdot \overline{btnL} \cdot \overline{btnR} + LFRL \cdot \overline{btnL} \cdot btn$

4 right first = $\overline{btnL} \cdot \overline{btnR} \cdot idle + RF \cdot \overline{btnL} \cdot$

5 right first both = $RF \cdot btnL \cdot btnR + RFB \cdot btnL \cdot btnR$

6 right first left last = $RFB \cdot btnL \cdot \overline{btnR} + RFLL \cdot \overline{btnL} \cdot$

shared inverted on display

Also accidently gave an output

2 designs.