Jacob Sickafoose

Lab # 4

Lab Section - 1B

2/14/2020

Description -

For this lab, we were tasked with creating a counter which counts up when you hit the up button, and counts down when you hit the down button. This counter also had to be loadable, using the 16 switches to represent the 16-bits of the counter. There were also small parts like, holding the continuous counter button only counts up until FFFC and there was btnU which only incremented up, once per button press.

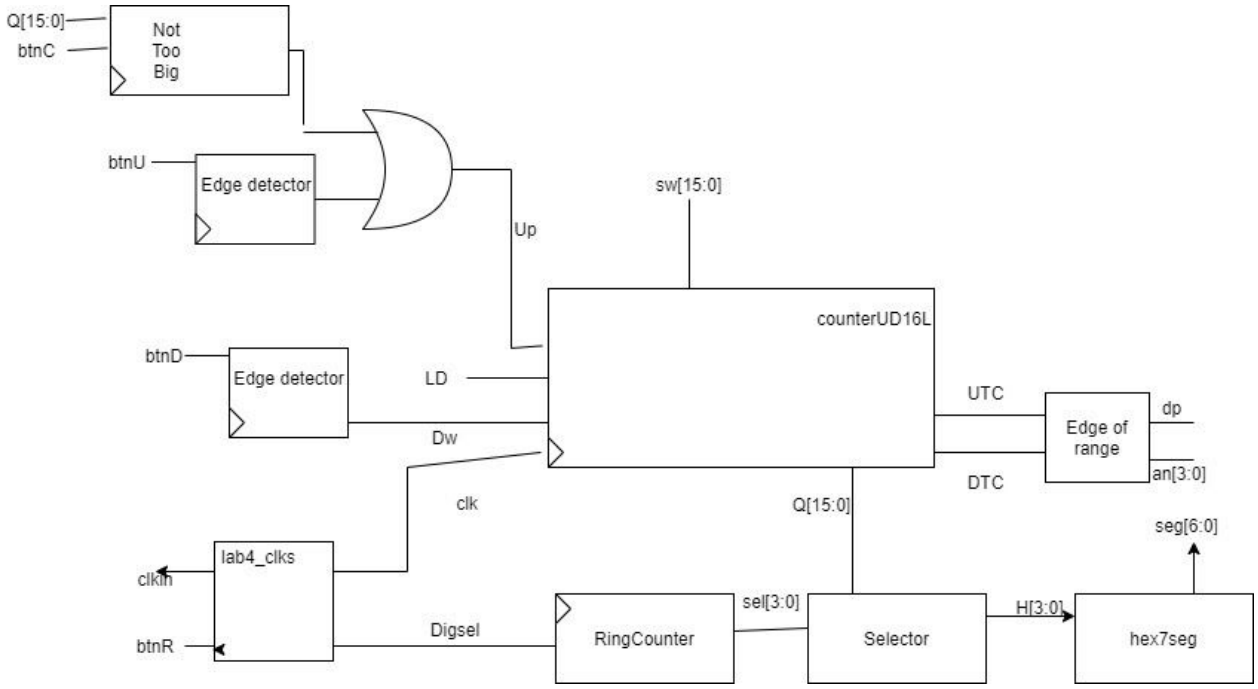
Method -

I started this lab by first implementing the logic for each of the 4-bit counters. I did this by just making a truth table for the output of the 4 flip-flops when either an Up or Dw signal was set to high. These needed to output a UTC and DTC signal when they were at their highest and lowest values. The 4-bit counters finally had to handle replacing the flip-flop values with Din, if LD was high. Next I figured out how to make the 16-bit counter, using 4 of the 4-bit counters. This required utilizing the UTC and DTC signals to tell the next highest bit when to increment, or decrement. I then moved on to the edge detector which used 2 flip-flops to tell when a change from the 0 to 1 state of btnU happened. On the first change of state, it would output high for one clk cycle. Finally, I needed to make the RingCounter, Selector, and hex7seg modules for outputing the 16bit number to all 4 of the 7segment displays. Ring counter just alternated which bit of 4 was high, while selector used that as input to select which 4-bits of the 16-bit number to

upload to the display. This final 4-bit digit of the display was translated to light up the correct segments in hex7seg. The an[3:0] value was also relying on the inverse of the ring counter.

Design -

This is the schematic for my top level:



We had to figure out how to make btnC only increment until the top 14-bits of the Q[15:0] output were 1. This kept it in the range of FFFC - FFFF. We also had to turn on the left, center decimal point when the number was FFFF, and the right center point when the number was 0000. For the 4-bit counters, I used the following equation to give me the input for the flip-flops when Up or Dw were on:

$D_0=Q_0$^Up^Dw

$D_1=Q_1$^(Q_0&Up)^(~Q_0&Dw)

$D_2=Q_2$^(Q_0&Q_1&Up)^(~Q_0&~Q_1&Dw)

$D_3=Q_3$^(Q_0&Q_1&Q_2&Up)^(~Q_0&~Q_1&~Q_2&Dw)

I did a similar thing with the edge detector. I made the following truth table for the two bit flip-flops, when btn is high:

		~Btn		Btn	
Q_1	Q_0	D_1	D_0	D_1	D_0
0	0	0	0	0	1
0	1	1	0	1	1
1	0	0	0	0	1
1	1	1	0	1	1

Where I got the following equations for the D's:

D_0=btn
D_1=Q_0

Conclusion -

In order to finish this lab, I had to learn how to use flip-flops, as well as how to string them together for different effects. We had to use them to count, use them to detect edges in alternating signals, and how to make a ring counter with them. The longest part was figuring out how to make the counters and implement their truth table. It took a long time also, to string all the counters together and figure out how to use their UTC and DTC values. The rest was pretty easy to implement. If I had to do this lab again, it would be much better now that I understand how flip-flops work.

Appendix -

```
module lab4_top(
    input clkin,
    input btnR,
    input btnU,
    input btnD,
    input btnC,
    input btnL,
    input [15:0] sw,
    output [6:0] seg,
    output dp,
    output [3:0] an,
    output [15:0] led
    );

    wire dig_sel;
    wire Dw, OR1, OR2, OR3, UTC, DTC;
    wire [15:0] Q;
    wire [3:0] sel, H;
    wire clk;
    assign led = sw;

    questionMark question (.Q(Q), .btn(btnC), .out(OR1));
    lab4_clks clks (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(dig_sel));
    edgeDetector edgeDetectorUp (.btn(btnU), .clk(clk), .btnOut(OR2));
    edgeDetector edgeDetectorDw (.btn(btnD), .clk(clk), .btnOut(Dw));
    assign OR3 = OR1 | OR2;
    counterUD16L Counter (.clk(clk), .Din(sw), .Dw(Dw), .Up(OR3), .LD(btnL),
    .UTC(UTC), .DTC(DTC), .Q(Q));
    ringCounter RingCounter (.adv(dig_sel), .clk(clk), .sel(sel));
    selector Selector (.sel(sel), .N(Q), .H(H));
    hex7seg hex7seg (.n(H), .seg(seg));

    assign an = ~sel;
    assign dp = ~(DTC & sel[1] | UTC & sel[2]);
endmodule
```

```
module questionMark(
    input [15:0] Q,
    input btn,
    output out
    );

    assign out = btn&~(&Q[15:2]);
endmodule
```

```
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 counterUD16L(
    input clk,
    input [15:0] Din,
    input Dw,
    input Up,
    input LD,
    output UTC,
    output DTC,
    output [15:0] Q
    );
    wire [3:0] utc;
    wire [3:0] 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]));
    countUD4L count2 (.Up(utc[1]&Up&utc[0] | Up&UTC), .Dw(dtc[0]&dtc[1]&Dw | Dw&DTC), .LD(LD), .Din(Din[11:8]), .clk(clk), .UTC(utc[2]), .DTC(dtc[2]), .Q(Q[11:8]));
    countUD4L count3 (.Up(utc[2]&Up&utc[0]&utc[1] | Up&UTC), .Dw(dtc[0]&dtc[1]&dtc[2]&Dw | Dw&DTC), .LD(LD), .Din(Din[15:12]), .clk(clk), .UTC(utc[3]), .DTC(dtc[3]), .Q(Q[15:12]));

    assign UTC = utc[0]&utc[1]&utc[2]&utc[3];
    assign DTC = dtc[0]&dtc[1]&dtc[2]&dtc[3];

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
```

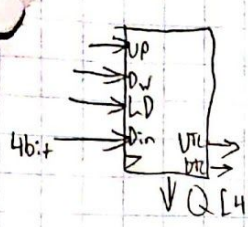
```
module m8_1(
    input [7:0] in,
    input [2:0] sel,
    output o
    );

    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 4



UP - increment
Dw - decrement
LD - load from switches (takes precedence)
Din - 4 bit vector loaded to FF if Load is high

UTC - (UP Terminal Count) is 1 when 1111 stored
DTC - (Down Terminal Count) is 1 when the counter is 0000

Counter4UDL

Q_3	Q_2	Q_1	Q_0	UP	down
0	0	0	0	0 0 0 1	1 1 1 1
0	0	0	1	0 0 1 0	0 0 0 0
0	0	1	0	0 0 1 1	0 0 0 1
0	0	1	1	0 1 0 0	0 0 1 0
0	1	0	0	0 1 0 1	0 0 1 1
0	1	0	1	0 1 1 0	0 1 0 0
0	1	1	0	0 1 1 1	0 1 0 1
0	1	1	1	1 0 0 0	0 1 1 0
1	0	0	0	1 0 0 1	0 1 1 1
1	0	0	1	1 0 1 0	1 0 0 0
1	0	1	0	1 0 1 1	1 0 0 1
1	0	1	1	1 1 0 0	1 0 1 0
1	1	0	0	1 1 0 1	1 0 1 1
1	1	0	1	1 1 1 0	1 1 0 0
1	1	1	0	1 1 1 1	1 1 0 1
1	1	1	1	0 0 0 0	1 1 1 0

$D_0 = Q_0 \oplus UP \oplus Dw$

$D_1 = Q_1 \oplus (Q_0 UP) \oplus (\overline{Q}_0 Dw)$

$D_2 = Q_2 \oplus (UP\, Q_1 Q_0) \oplus (Dw \overline{Q}_1 \overline{Q}_0)$

$D_3 = Q_3 \oplus (UP\, Q_2 Q_1 Q_0) \oplus (Dw \overline{Q}_2 \overline{Q}_1 \overline{Q}_0)$

EdgeDetector

Q_1	Q_0	btn d1 d0	btn d1 d0
0	0	0 0	0 1
0	1	1 0	1 1
1	0	0 0	0 1
1	1	1 0	1 1

$D_0 = btn$

$D_1 = Q_0$

Ring Counter

Q_3 Q_2 Q_1 Q_0	adv	$\overline{adv}$
0 0 0 1	0 0 1 0	0 0 0 1
0 0 1 0	0 1 0 0	0 0 1 0
0 1 0 0	1 0 0 0	0 1 0 0
1 0 0 0	0 0 0 1	1 0 0 0

$D[0] = Q_0 \sim adv + Q_3\, adv$

$D[1] = Q_1 \sim adv + Q_0 adv$

$D[2] = Q_2 \sim adv + Q_1\, adv$

$D[3] = Q_3 \sim adv + Q_2\, adv$

16 bit Counter

when dw and up, ~~f~~

a100 → a0fe

00a6 dw ⇒ f0a5

≡

Question Mark

1111	1111	1111	1100
1111	1111	1111	1101
1111	1111	1111	1110
1111	1111	1111	1111

Q[15:2] = 1

~~Q[15:2]~~

TN 6676
2/6/20 14:48
1 day late