Noah Khan

May 6, 2022

16-Bit Hexadecimal Counter

Description:

For this lab, I created a sequential circuit using the library module FDRE for flip-flops. Completing this lab required 9 different modules. These modules are a 4-bit counter, 16-bit counter, selector, ring counter, edge detector, hex7seg, x8 2-to-1 multiplexers, and a top level. In addition, I created a mystery box module which handled logic for the center button. Furthermore, the hex7seg and the x8 2-to-1 multiplexer is reused from lab 3.

Methods:

I began by creating the 4-bit counter module. It had 5 inputs and 3 outputs. The 5 inputs were clk, Up, Dw, LD, and Din, where Din is was a 4-bit vector. The outputs were Q, UTC, and DTC, where Q was a 4-bit vector. This module handles the logic to increment or decrement the number displayed on the seven-segment LED on the Basys 3 board. Additionally, this module instantiates the FDRE flip flop 4 times. The flip flops have the inputs .C, .CE, .D, .Q which represent the clock, count enable, D (flip flop input) value and Q (flip flop output) value, respectively.

UTC is 1 when all the bits in Q are 1 and DTC is 1 when all the bits in Q are 0. To take care of this, I assigned UTC to &Q and DTC to &(~Q). For UTC, the logic ANDs all the bits of UTC together. This way, if all bits are 1, UTC will be 1. For DTC, the logic inverts all the bits in Q then ANDs them together. In this case, if all the bits are 0, they will be inverted, ANDed together, and DTC will equal 1.

I created a wire named clk_enable. This wire will feed into. CE, the count enable. I realized I made a mistake in naming the wire, it should be named count_enable. In any case, it transfers the proper logic, which is Dw XOR'd with Up and all of that is OR'd with LD.

Next, I created a 4-bit bus. This bus handles the addition and subtraction logic. Each wire from the bus feeds into one flip flop. I looked through the textbook and found helpful information for the addition logic. Then, I drew up some expressions that could possibly handle the subtraction logic. I OR'd these logical expressions together. I assigned each wire to their respective expressions.

After creating the 4-bit counter, I created the 16-bit counter. This module was similar to the 4-bit counter and instantiated the 4-bit counter 4 times. UTC and DTC used the same logic as the 4-bit counter.

After the creating the 16-bit counter, I created logic for the selector. From what I understood, H needed to equal the hex number indicated by the select value.

Afterwards, I created the ring counter. The schematic of this logical component could be found in the textbook. It requires 4 FDRE flip flops two inputs, clk and advance, and a 4-bit vector output [3:0] Qout. Where the .C port takes the clk input, .CE takes advance and the first flip flop takes the Qout[3] for .D and Qout[0] for the .Q port. The other flip flops have the same cascading logic. All their .R ports will be 0 and all of them will be initialized to 0 except for one of the flip flops. The one that is 1 is arbitrary.

Next was the edge detector, which had 2 inputs; clk and btn, and 1 output; y. In addition, it instantiates 2 FDRE flip flops Where the first takes btn for the .D port. There needs to be wires connecting .Q ports and .D port for the second flip flop.

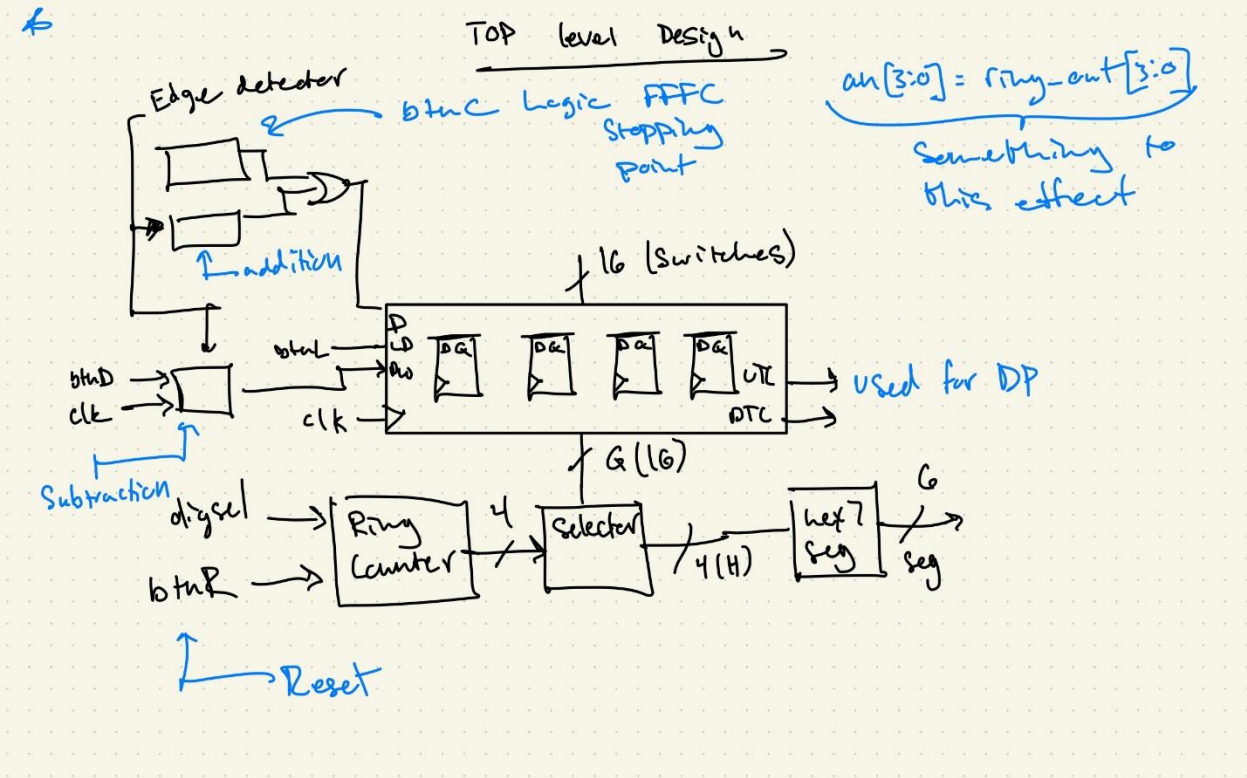
Next, I created a mystery box module. This module was not necessary because the logic could be implemented into the top-level module. However, I did this for cleanliness. This module had a 16-bit vector as the input and btnC. The output can have any name. I named it “out.” I inverted all the 14 most left bits for the 16-bit vector and OR’d the bits all together, then, I AND’d it with the btnC.

Finally, I created the top-level module. This module had 7 inputs, clkin, btnR, btnU, btnD, btnC, btnL, and a 16-bit vector sw. It had 4 outputs a 7-bit vector seg, dp, 4-bit vector an, and 16-bit vector led. I created an instance for mystery box, 2 for edge detector, ring counter, 16-bit counter, selector, hex7seg and lab4_clks, which was given in the lab manual. I connected all instances together while using the diagram in the lab manual as reference. I connected each index of “an” to each index of the output from the ring counter and inverted it. The logic I sued for dp utilized index 1 and 2 for the ring counter output. Index 2 was AND’d with the UTC output and that was OR’d with index 1 and AND’d with the DTC output and the entire expression was negated.

I created a testbench for the lower-level modules to make sure they functioned properly.

Results:

Design:



Testing & Simulation:

I created a testbench for most of the lower-level modules in lab 4. All simulations can be found in the lab submission file. The first module I tested was the 4-bit counter. I tested btnU, btnD, and btnL. I believe the test cases were making sure btnL worked while either btnU or btnD was on. In addition, I made sure UTC was 1 when all the bits were 1 and DTC was 1 when all the bits were 0.

The 16-bit counter was basically the same thing. However, while running simulations, I did notice there was an error with my load (btnL) logic. The right most hex number form Din would be loaded into all the Q values. This was fixed by changing some logic in the 16-bit counter module.

The ring counter was fairly simple, I gave a value to the “advance” input and saw the bit shift.

The edge detector was simple too, I checked to see when the clock edge would rise.

For the selector, I tested random inputs to see what the outcome would be. It worked properly, by giving me the hex number depending on the switch (0001, 0010, 0100, 1000) that was active.

I used the given testbench to test the top-level module.

I did not test the modules obtained from lab 3 and I did not test the mystery box module.

Lab Questions:

When I hold down btnR, all the LED segments on the seven-segment display turn off. I think this is because the lab 4 clks module handles the global reset which is mapped to btnR.

When fastclk is connected to the clock inputs instead of clk, I see an[2:0] are faint. I am assuming the segments are displaying for a short amount of time making the display seem faint.

Conclusion:

From this lab, I learned how to create a sequential circuit using FDRE flip flops. The hardest part for me was figuring out the logic for the 4 bit-counter and implementing it. A reoccurring problem for me was figuring out what the count enable input was for the other flip flops in the other modules. If I could do this lab again, would have gotten started on it earlier. I would not know how to optimize it.

Appendix A:

4-bit counter (1)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/21/2022 03:42:26 PM
// Design Name:
// Module Name: countUD4L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD4L(
    input clk,
    input Up,           // increment
    input Dw,           // decrement
    input LD,           // load control
    input [3:0] Din,    // the 4-bit vector Din that will be loaded on the clock
edge if LD is high.
    output [3:0] Q,     // current value held by the counter
    output UTC,         //1 only when the counter is at 1111 (Up Terminal Count)
    output DTC          //1 only when the counter is at 0000 (down Terminal Count)
    );

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

    wire [3:0]FF_wire;
    wire clk_enable;

    assign clk_enable = (Dw ^ Up) | LD;

    assign FF_wire[0] = (Q[0] ^ clk_enable);
    assign FF_wire[1] = ( Up & (Q[1] ^ (Q[0] & clk_enable))) | (Dw & (Q[1] ^ (~Q[0]
& clk_enable)));
```

4-bit counter (2)

```
    assign FF_wire[2] = ( Up & (Q[2] ^ (Q[1] & Q[0] & clk_enable ))) | (Dw & (Q[2] ^
(~Q[1] & ~Q[0] & clk_enable )));
    assign FF_wire[3] = ( Up & (Q[3] ^ (Q[2] & Q[1] & Q[0] & clk_enable ))) | (Dw
&(Q[3] ^ (~Q[2] & ~Q[1] & ~Q[0] & clk_enable )));

    FDRE #(.INIT(1'b0) ) ff_Q0 (.C(clk), .R(1'b0), .CE(clk_enable), .D( (LD &
Din[0]) | (~LD & FF_wire[0]) ), .Q(Q[0]));
    FDRE #(.INIT(1'b0) ) ff_Q1 (.C(clk), .R(1'b0), .CE(clk_enable), .D( (LD &
Din[1]) | (~LD & FF_wire[1]) ), .Q(Q[1]));
    FDRE #(.INIT(1'b0) ) ff_Q2 (.C(clk), .R(1'b0), .CE(clk_enable), .D( (LD &
Din[2]) | (~LD & FF_wire[2]) ), .Q(Q[2]));
    FDRE #(.INIT(1'b0) ) ff_Q3 (.C(clk), .R(1'b0), .CE(clk_enable), .D( (LD &
Din[3]) | (~LD & FF_wire[3]) ), .Q(Q[3]));


endmodule
```

16-bit counter (1)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2022 02:04:11 AM
// 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, // increment
    input Dw, // decrement
    input LD, // load control
    input [15:0] Din, // the 4-bit vector Din that will be loaded on the clock edge
if LD is high.
    output [15:0] Q, // current value held by the counter
    output UTC,  //1 only when the counter is at 1111 (Up Terminal Count)
    output DTC  //1 only when the counter is at 0000 (down Terminal Count)

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

     wire [3:0] utc_connect;
     wire [3:0] dtc_connect;

    countUD4L c0  ( .clk(clk), .Up(Up), .Dw(Dw), .LD(LD), .Din(Din[3:0]),
.Q(Q[3:0]),  .UTC(utc_connect[0]), .DTC(dtc_connect[0]) );
    countUD4L c1  ( .clk(clk), .Up(Up & ~Dw & utc_connect[0]), .Dw(Dw & ~Up &
dtc_connect[0]), .LD(LD), .Din(Din[7:4]), .Q(Q[7:4]),  .UTC(utc_connect[1]),
```

```
.DTC(dtc_connect[1]) );
    countUD4L c2  ( .clk(clk), .Up(Up & ~Dw & (&utc_connect[1:0])), .Dw(Dw & ~Up &
(&dtc_connect[1:0])), .LD(LD), .Din(Din[11:8]), .Q(Q[11:8]), .UTC(utc_connect[2]),
.DTC(dtc_connect[2]) );
    countUD4L c3  ( .clk(clk), .Up(Up & ~Dw & (&utc_connect[2:0])), .Dw(Dw & ~Up &
(&dtc_connect[2:0])), .LD(LD), .Din(Din[15:12]), .Q(Q[15:12]),.UTC(utc_connect[3]),
.DTC(dtc_connect[3]) );

endmodule
```

Edge Detector

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2022 05:16:50 PM
// Design Name:
// Module Name: EdgeDetector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module EdgeDetector(

    input clk,
    input btn,
    output y

    );

    // TESTED i think it works, may have possible errors?

    wire [1:0] wires;

    FDRE #(.INIT(1'b0) ) ffUp_0 (.C(clk), .CE(1'b1), .D(btn), .Q(wires[0]));
    FDRE #(.INIT(1'b0) ) ffUp_1 (.C(clk), .CE(1'b1), .D(wires[0]), .Q(wires[1]));

    assign y = wires[0] & ~wires[1];

endmodule
```

Hex7seg

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/12/2022 01:47:15 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,
    output [6:0] seg_out
    );
    wire not_n0;
    assign not_n0 = ~n[0];

    m8_1 sevensegment0(.sel(n[3:1]), .in({1'b0, n[0], n[0], 1'b0, 1'b0, not_n0,
1'b0, n[0]}), .o(seg_out[0]));
    m8_1 sevensegment1(.sel(n[3:1]), .in({1'b1, not_n0, n[0], 1'b0, not_n0, n[0],
1'b0, 1'b0}), .o(seg_out[1]));
    m8_1 sevensegment2(.sel(n[3:1]), .in({1'b1, not_n0, 1'b0, 1'b0, 1'b0, 1'b0,
not_n0, 1'b0}), .o(seg_out[2]));
    m8_1 sevensegment3(.sel(n[3:1]), .in({n[0], 1'b0, not_n0, n[0], n[0], not_n0,
1'b0, n[0]}), .o(seg_out[3]));
    m8_1 sevensegment4(.sel(n[3:1]), .in({1'b0, 1'b0, 1'b0, n[0], n[0], 1'b1, n[0],
n[0]}), .o(seg_out[4]));
    m8_1 sevensegment5(.sel(n[3:1]), .in({1'b0, n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b1,
n[0]}), .o(seg_out[5]));
    m8_1 sevensegment6(.sel(n[3:1]), .in({1'b0, not_n0, 1'b0, 1'b0, n[0], 1'b0,
1'b0, 1'b1}), .o(seg_out[6]));

endmodule
```

x8 2-to-1 multiplexers

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/11/2022 07:10:50 PM
// Design Name:
// Module Name: m8_1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m8_1(

    input [7:0] in,
    input [2:0]sel,
    output o

    );

    assign o = (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] & sel[2] & sel[1] & sel[0]);

endmodule
```

Ring Counter

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2022 04:21:52 PM
// Design Name:
// Module Name: RingCounter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module RingCounter(
    input clk,
    input advance,
    output [3:0] Qout
    );
    // TESTED
    FDRE #(.INIT(1'b0) ) ff_0 (.C(clk), .R(1'b0), .CE(advance), .D(Qout[3]),
.Q(Qout[0]));
    FDRE #(.INIT(1'b0) ) ff_1 (.C(clk), .R(1'b0), .CE(advance), .D(Qout[0]),
.Q(Qout[1]));
    FDRE #(.INIT(1'b0) ) ff_2 (.C(clk), .R(1'b0), .CE(advance), .D(Qout[1]),
.Q(Qout[2]));
    FDRE #(.INIT(1'b1) ) ff_3 (.C(clk), .R(1'b0), .CE(advance), .D(Qout[2]),
.Q(Qout[3]));

endmodule
```

Selector

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2022 02:40:13 AM
// 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
    );
    // TESTED
    assign H[0] = (sel[0] & N[0] ) | (sel[1] & N[4] ) | (sel[2]& N[8] )  | (sel[3]&
N[12]);
    assign H[1] = (sel[0] & N[1] ) | (sel[1] & N[5] ) | (sel[2]& N[9] )  | (sel[3]&
N[13]);
    assign H[2] = (sel[0] & N[2] ) | (sel[1] & N[6] ) | (sel[2]& N[10] ) | (sel[3]&
N[14]);
    assign H[3] = (sel[0] & N[3] ) | (sel[1] & N[7] ) | (sel[2]& N[11] ) | (sel[3]&
N[15]);


endmodule
```

Mystery Box

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/27/2022 03:38:01 PM
// Design Name:
// Module Name: MysteryBox
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module MysteryBox(
    input [15:0] Q,
    input btnC,
    //input clk,
    output out
    );

    assign out = |(~Q[15:2])&btnC;

endmodule
```

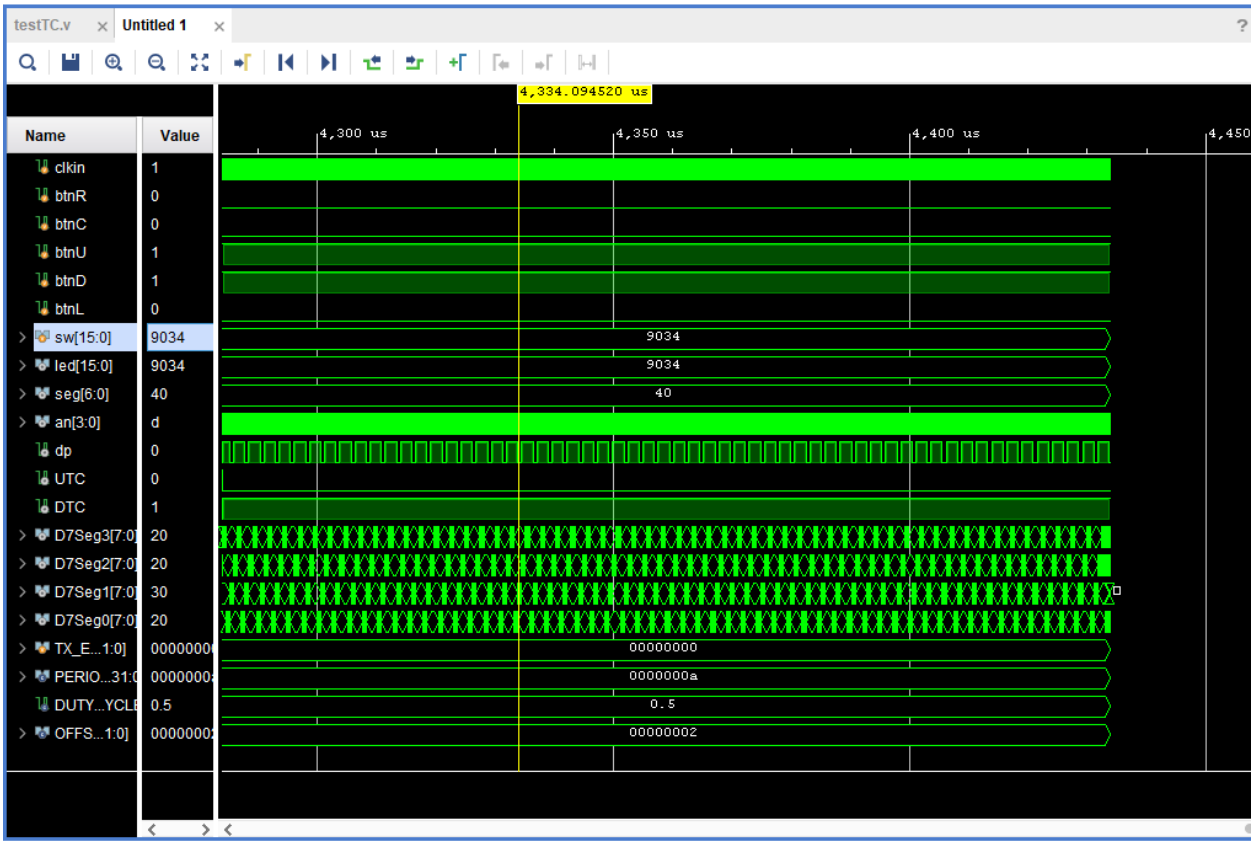
Top-Level (1)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/25/2022 08:55:52 PM
// Design Name:
// Module Name: Top_Level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Top_Level(

    input clkin,        //- this is the 100MHz clock on the Basys 3 Board
    input btnR,         //- this button will be connected the built-in global reset
of the Artix 7 FPGA
    input btnU,         //- when pressed this button causes the counter to increment
by just one
    input btnD,         //- when pressed this button causes the counter to decrement
by just one
    input btnC,         //- when pressed this button causes the counter to advance
continuously except in the FFFC to FFFF range
    input btnL,         //- when pressed this button causes the counter to load the
value determined by the switches on the clock edge
    input [15:0] sw,    //- this 16 bit vector determines the value loaded into the
counter when btnL is pressed
    output [6:0] seg,       //- this 7 bit vector controls the segments in the
7-segment display
    output dp,              //- this controls the dp segments in the 7-segment
display. Remember: when value is UTC left center is lit and when value is DTC right
center is lit.
    output [3:0] an,        //- this 4 bit vector controls the 4 digits of
7-segment display
    output [15:0] led       //- this 16 bit output corresponds to the switches.
```

Top-Level (2)

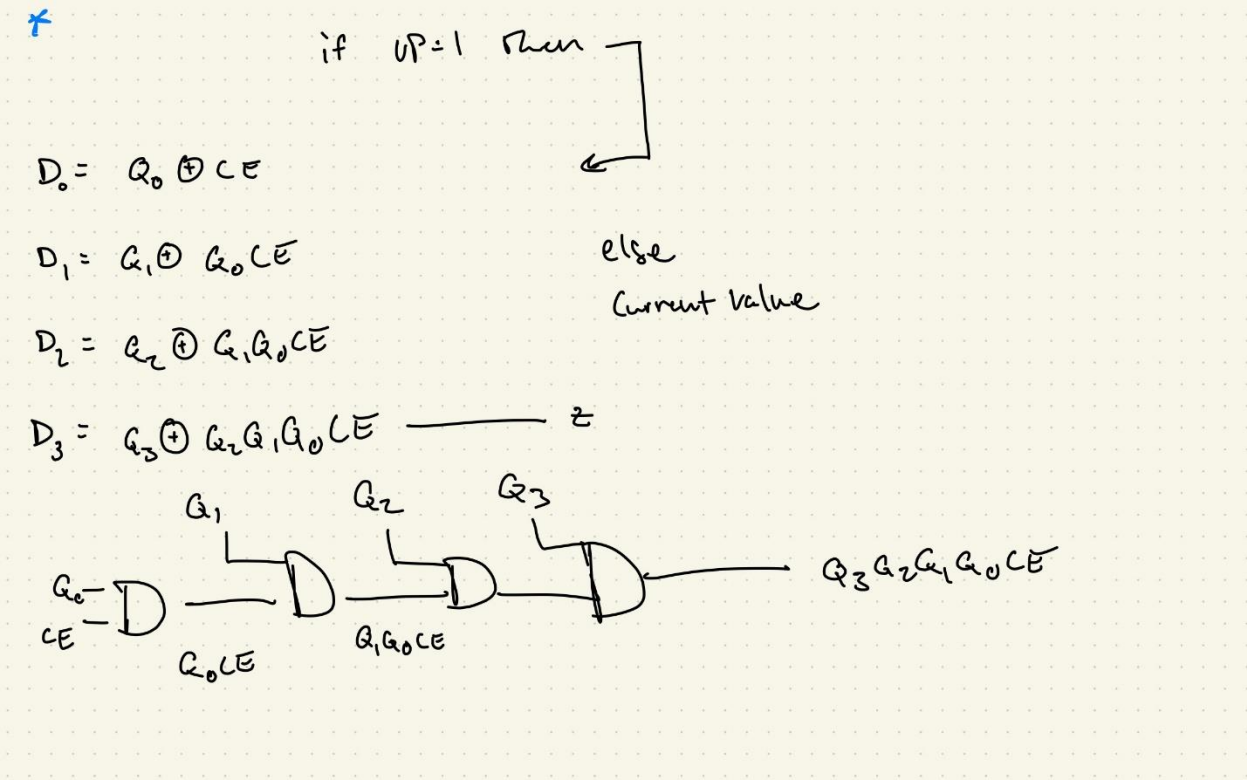
```
    );
    // if issues occur, troubleshoot: ringcounter, selector, edge detector
    wire [3:0] select_to_hex7;
    wire [3:0] ring_to_selctor;
    wire [15:0] count16_out;
    wire digsel, clk, edge_up, edge_dw, mystery_to_count16, utc_wire, dtc_wire;

    MysteryBox mb ( .Q(count16_out), .btnC(btnC), .out(mystery_to_count16) );
    EdgeDetector egdeUp ( .clk(clk), .btn(btnU), .y(edge_up) );
    EdgeDetector egdeDw ( .clk(clk), .btn(btnD), .y(edge_dw) );

    RingCounter ring ( .clk(clk), .advance(digsel), .Qout(ring_to_selctor) );

    countUD16L count16 ( .clk(clk), .Up(mystery_to_count16 | edge_up), .Dw(edge_dw),
.LD(btnL), .Din(sw), .Q(count16_out), .UTC(utc_wire), .DTC(dtc_wire));
    selector sel ( .sel(ring_to_selctor), .N(count16_out), .H(select_to_hex7));
    hex7seg seven_seg ( .n(select_to_hex7), .seg_out(seg) );

    lab4_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel));

    assign dp = ~((ring_to_selctor[2]&utc_wire)|(ring_to_selctor[1]&dtc_wire));
//an[2] is utc, an[1] is dtc
    assign an[0] = ~ring_to_selctor[0];
    assign an[1] = ~ring_to_selctor[1];
    assign an[2] = ~ring_to_selctor[2];
    assign an[3] = ~ring_to_selctor[3];
    assign led[15:0] = sw[15:0];

endmodule
```

Waveform

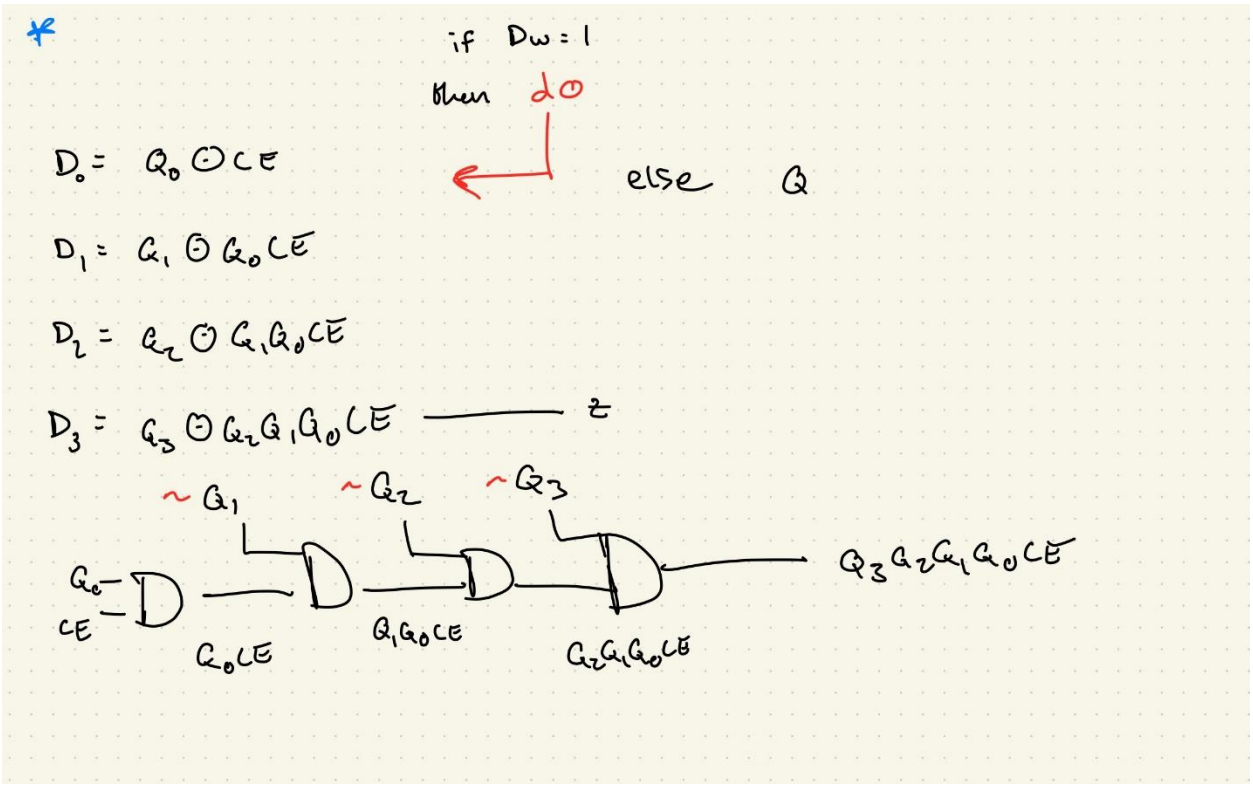

testTC.v
Untitled 1
4,334.094520 us
4,300 us
4,350 us
4,400 us
Name
Value
clkin
1
btnR
0
btnC
0
btnU
1
btnD
1
btnL
0
sw[15:0]
9034
led[15:0]
9034
seg[6:0]
40
an[3:0]
d
dp
0
UTC
0
DTC
1
D7Seg3[7:0]
20
D7Seg2[7:0]
20
D7Seg1[7:0]
30
D7Seg0[7:0]
20
TX_E...1:0]
PERIO...31:0
DUTY...YCLE
0.5
OFFS...1:0]
9034
9034
40
00000000
0000000a
0.5
00000002

Appendix B:

Addition



Subtraction



Selector logic

Selector

S_3	S_2	S_1	S_0	H
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1

Boolean Expression

$$\bar{S}_3\bar{S}_2\bar{S}_1S_0N[3:0] + \bar{S}_3\bar{S}_2S_1\bar{S}_0N[7:4]$$

$$\bar{S}_3S_2\bar{S}_1\bar{S}_0N[11:8] + S_3\bar{S}_2\bar{S}_1\bar{S}_0N[15:12]$$

$$S_0N[3:0] + S_1N[7:4] +$$

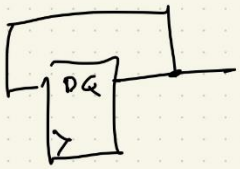
$$S_2N[11:8] + S_3N[15:12]$$

Mystery box

Stop at FFFC Do not go further

Focus on the First 14 bits
↳ last 2 bits?

if G[15:2] = 1
then ~btnC
else




```
!(~G[15:2]) & btnC
```