Lab 4 Writeup
Yaroslav Yanin
11/08/2019
Section: Th 9:50-11:40

Purpose:

The purpose of this lab was to build a sequential circuit using flip flops. That circuit had to use the clock to operate. I had to implement logic that would run in a loop through various flip flops. I was only allowed to use the assign command. The sequential circuit was used to build a 16 bit counter and have the results displayed on all 4 parts of the 7 segment display. After completing each module I had to connect it in the top level.

Methods:

As all other labs, I have started building each module and testing it separately. The first module I built was the 3-bit counter. I wrote out the state table and then wrote out the k-maps for each of the next states. After that I did the same thing for the 5 bit counter. After creating both modules I simply connected them in the 16 bit counter module. I used both modules twice to create the 16 bit counter. I then simply followed the instructions to create the Edge detector, ring counter, and the selector. Those modules were more simple than the counters. I then used the 7-seg module from lab 2 and 3. After finishing all the modules I connected all of them in the top module.

Results:

1. If I hold down btnR then the display shows only one 0. The display is not different, but all but AN0 are turned off. This happens due to the RingCounter. Since only one of the flip flops is initialized to 1, then when I constantly reset it, all other parts of the display show nothing.
2. After changing from regular clock to fast clock, the rightmost digit is bright while the other 3 are dim. Incrementing either up or down is very high, while trying to increment continuously(btnC) increments to FFFC almost instantly.

- All the used truth tables and equations are in the lab notebook.

The counter design is shown in the lab notebook. Each counter has n flip flops for n bits. The flip flops are only connected through UTC and DTC. This way they will not count at the same time and only when the previous flip flop demands it. The design itself is pretty simple. It's a flip flop followed by a cloud of logic describing the next state which then goes back into the flip flop. As stated before, all equations are in the notebook scan.

The Edge Detector design is even more simple. It is two flip flops connected to one another: Q0 is connected to D1 and that produces an out. The logic is: ~Q0&Q1. The design is also included in the lab notebook.

The RIng counter consists of 4 flip flops each connected to one another via output to input. The first input takes in an inverted In[3:0] and then each Q of the flip flop is connected to the input of the next flip flop. When the last flip flop is reached, Q3 is connected to D0. After each flip flop, the output comes out so that there is an output that comes from a closed loop.

The 16-bit counter utilizes two 3-bit counters and two 5 bit counters. Each successive counter is connected to one another in the same way as the flip flops within each counter: through UTC and DTC. The more significant bits take in the previous UTC and DTC values & with one another via wires. After comes the logic for both UTC and DTC where all respective wires and & with one another to go into the output UTC or DTC

For the selector, at first I thought that I should use muxes as I did in lab 3, but that got me more confused so I asked for help. I was told taht it is best to do it with simple logic:

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

To do the top level, I simply followed the given schematic. I connected everything the way it was specified. The logic to deal with buttonC issue: W1 = (~(&W5[15:2]))&btnC. After that, the count should advance continuously even at FFFC to FFFF. The logic for decimal point: dp = ~(((T1)& ~an[3])|((T2)& ~an[0])). The LEDs are simply equal to the switches.

To test my design I simply made TestBenches for each necessary file. The only file I needed to test were the counters. I had no issues with the other modules because they were fairly easy to understand. If something went wrong later, it would be easy to isolate the issue by using the test bench. I therefore tested the 3-bit counter, 5-bit counter, and the 16-bit counter to make sure that it counted up properly. I then tested the top level to make sure it has the right output for each button. I did not need to input too many values, since it would be easy to see if something was not working through a small number of inputs. I then used the Babys board to test whether everything was working properly, since there were still components that needed testing on the physical board, or were just easy to see in on the board.

Simulation discussion:

As we can see from the simulation, when btnC is pressed timer starts to rapidly count up. When btnR is pressed, it resets, when btnU is pressed it increments once, when btnL is pressed it goes to the highest value, when btnD is pressed it decrements by one. I set the switches to a smaller value so that it would be easier to read the simulation. As stated before, I did not need to input many values. I just wanted to make sure that every button did what it was supposed to do. Testing for a small span of values was the most reasonable choice.

Conclusion:

In conclusion, it took me a while to finish the lab. The most difficult part was creating the working counter. The top level was straightforward and the other modules were not too bad. All the code and schematics can be seen in this report. The hardest part was figuring out the logic for everything. The part that was most frustrating was the debugging. I, along with the TA, spent several hours looking for a bug that was ruining my entire code. The issue was that when I was compensating for the continuous incrementation between FFFC to FFFF, I used the output instead of a wire. That was preventing everything from working so it was very difficult to isolate. If I did this lab again, I would start earlier. There are no components I would optimise. I'm pretty happy with everything.

Information from notebook:

Up:

			D2		D1		D0	
Q2	Q1	Q0	~CE	CE	~CE	CE	~CE	CE
0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	1	1	0
0	1	0	0	0	1	1	0	1
0	1	1	0	1	1	0	1	0
1	0	0	1	1	0	0	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	0	1
1	1	1	1	0	1	0	1	0

D0=CE^Q0
D1=Q1^(Q0CE)
D2=Q^(CEQ1Q0)
For 5-bit
D[3]=Q3^(Q2Q1Q0CE)
D[4]=Q4^(Q3Q2Q1Q0CE)

Dw:

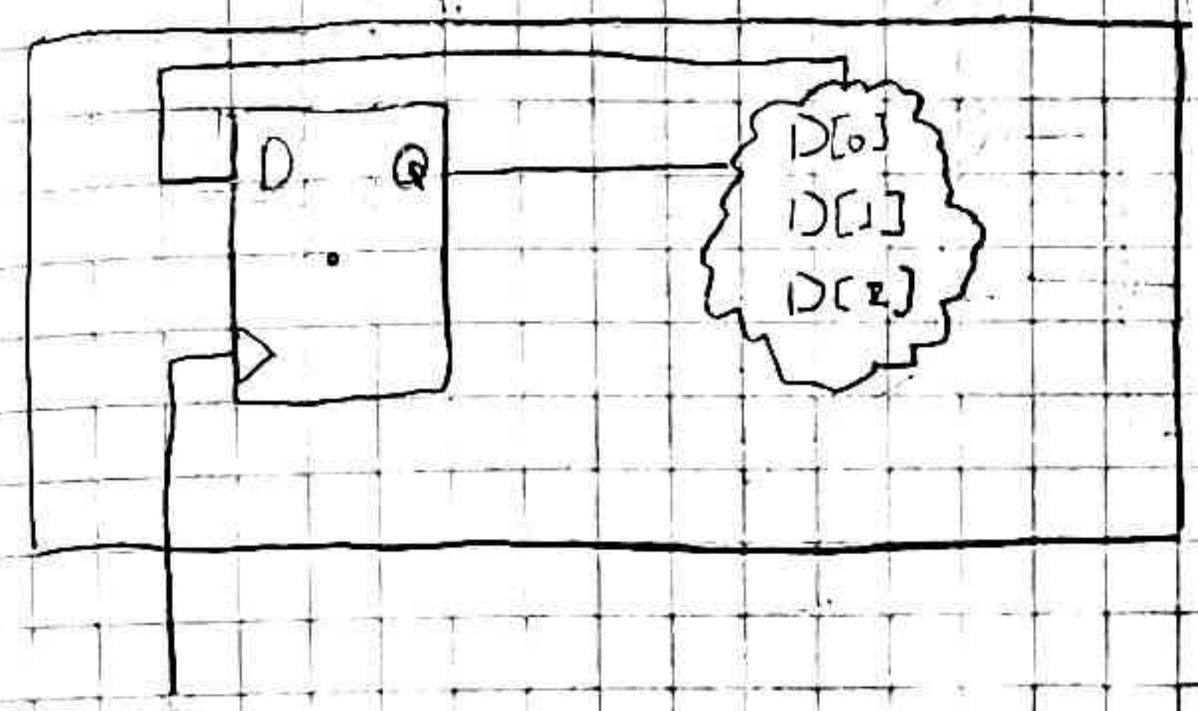
			D2		D1		D0	
Q2	Q1	Q0	~CE	CE	~CE	CE	~CE	CE
0	0	0	0	1	0	1	0	1
0	0	1	0	0	0	0	1	0
0	1	0	0	0	1	0	0	1
0	1	1	0	0	1	1	1	0
1	0	0	1	0	0	1	0	1
1	0	1	1	1	0	0	1	0
1	1	0	1	1	1	0	0	1
1	1	1	1	1	1	1	1	0

D0=CE^Q0
D1=Q1^(~Q0CE)
D2=Q2^(~Q1~Q0CE)

For 5-bit
D[3]=Q3^(~Q2~Q1~Q0CE)
D[4]=Q4^(~Q3~Q2~Q1~Q0CE)

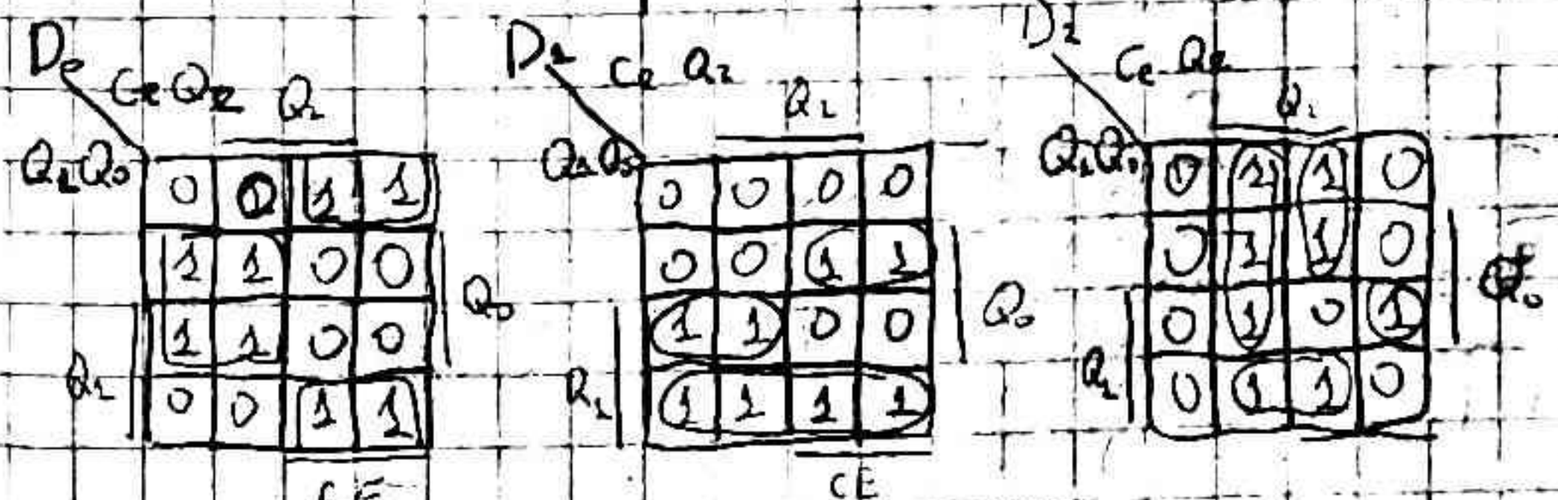
Lab 4

3 bit counter



Up:

Q_2	Q_1	Q_0	D_2 CE	D_2 CE	D_1 CE	D_1 CE	D_0 CE	D_0 CE
0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	1	1	0
0	1	0	0	0	1	1	0	1
0	1	1	0	1	1	0	1	0
1	0	0	1	1	0	0	1	0
1	0	1	1	1	0	1	0	1
1	1	0	1	1	1	0	1	0
1	1	1	1	0	1	0	1	0



$D_0 = Q_0\overline{CE} + CE\,\overline{Q_0} \longrightarrow \boxed{C_E \oplus Q_0}$

$D_1 = Q_1\overline{Q_0} + Q_1\overline{CE} + \overline{Q_1}Q_0\,CE$

$= Q_1(\overline{Q_0} + \overline{CE}) + \overline{Q_1}(Q_0\,CE)$

$= Q_1(\overline{Q_0\,CE}) + \overline{Q_1}(Q_0\,CE) \longrightarrow \boxed{Q_1 \oplus Q_0\,CE}$

$D_2 = \overline{CE}\,Q_2 + \overline{Q_1}Q_2 + \overline{Q_0}Q_2 + CE\,Q_1Q_0 \longrightarrow \boxed{Q_2 \oplus (CE\,Q_1Q_0)}$

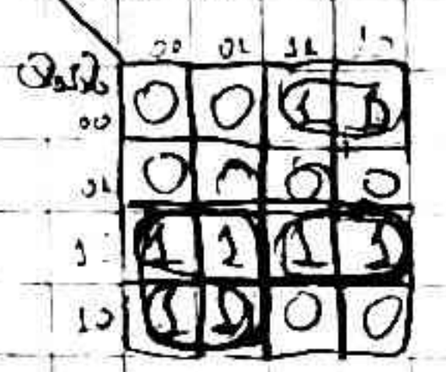
$= Q_2(CE\,Q_1Q_0) + \overline{Q_2}(CE\,Q_1Q_0)$

Down:

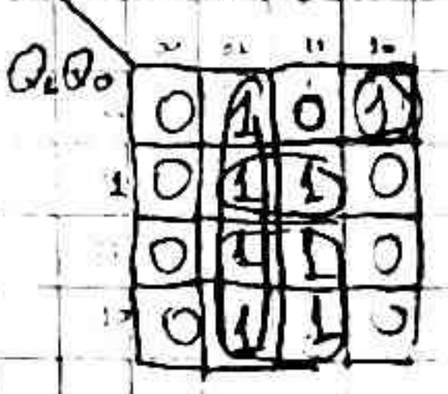
Q_2	Q_1	Q_0	D_2 $\overline{CE}$	D_2 CE	D_1 $\overline{CE}$	D_1 CE	D_0 $\overline{CE}$	D_0 CE
0	0	0	0	1	0	1	0	1
0	0	1	0	0	0	0	1	0
0	1	0	0	0	1	0	0	1
0	1	1	0	0	1	1	1	0
1	0	0	1	0	0	1	0	1
1	0	2	1	1	0	0	1	0
1	1	0	1	1	1	0	0	1
1	1	1	1	1	1	1	1	0

D_0

D_1



D_2



$$D_0 = \overline{CE}\,Q_0 + CE\,\overline{Q_0} = \overline{CE \oplus Q_0}$$

$$D_1 = \overline{CE}\,\overline{Q_1}\,Q_0 + \overline{CE}\,Q_1 + CE\,Q_1\,Q_0$$

$$= \overline{Q_1}(CE\,Q_0) + Q_1(\overline{CE} + \overline{CE\,Q_0})$$

$$= \overline{Q_1}(CE\,Q_0) + Q_1(\overline{Q_0} + \overline{CE}) = \overline{Q_1}(CE\,Q_0) + Q_1(\overline{Q_0}\,\overline{CE}) = \overline{Q_1 \oplus (Q_0\,CE)}$$

$$D_2 = \overline{CE}\,Q_2 + Q_2\,\overline{Q_1}\,Q_0 + Q_2\,Q_1 + CE\,\overline{Q_2}\,\overline{Q_1}\,\overline{Q_0}$$

$$= Q_2(\overline{CE} + \overline{Q_1}\,Q_0 + Q_1) + \overline{Q_2}(CE\,\overline{Q_1}\,\overline{Q_0})$$

$$= Q_2(\overline{CE\,Q_1\,Q_0}) + \overline{Q_2}(CE\,\overline{Q_1}\,\overline{Q_0})$$

$$= Q_2(\overline{CE\,\overline{Q_1}\,\overline{Q_0}}) + \overline{Q_2}(CE\,\overline{Q_1}\,\overline{Q_0})$$

$$= \overline{Q_2 \oplus (CE\,\overline{Q_1}\,\overline{Q_0})}$$

D_0 changes each time the counter goes up or down

D_1 changes every 2 counts up or 2 counts down

D_2 changes every 4 counts up or 4 counts down

↑ Later use for 5 bit counter


D[3:0]
D0
D0 = CE ⊕ Q0
D[2:0]
Din0
D1
Q1
LD
en
clk
D2 = (Up (Q2 ⊕ (Q0 CE)) +
(Dw (Q2 ⊕ (Q̄0 CE))
Up Dw Din2
D2
Q2
D2 = Q2 ⊕ (CE Q2 Q0)
Din2
clk
LD checker:
if (LD == 1) {
load Din
} else {
continue Q
}
UP DW checker:
if (Up == 1) {
CE = 1
} else if (Dw == 1) {
CE = 1
} else {
CE = 0
}
LDchecker = (~{3{LD}} & Q) | ({3{LD}} & Din)
Checking if it should load Din or Q
CE = (1 & Up) | (1 & Dw)
1'b1
Put it in 1 line

Both up & down are low:

```
if (CE=0){
    keep current
}
```

UTC trigger:

```
if (Q2=1 & Q1=1 & Q0=1){
    UTC=1;
}else{
    UTC=0;
}
```

UTC = ({3{1}} & Q) | 0.50

DTC trigger:

```
if (Q2=0 & Q1=0 & Q0=0){
    DTC=1;
}else{
    DTC=0;
}
```

DTC = ({3{0}} & Q) | 05

Clk delay

for D0: Clk works because it changes earlier than
for D1: Needs 2 clk delay
for D2: Needs 4 clk delay

HOW do I delay the Clock?

No need to

Won't let me tie Q and D wires, why?

Because they Q uses to

Stk logic for LD, UP, DW Counter 3 bit

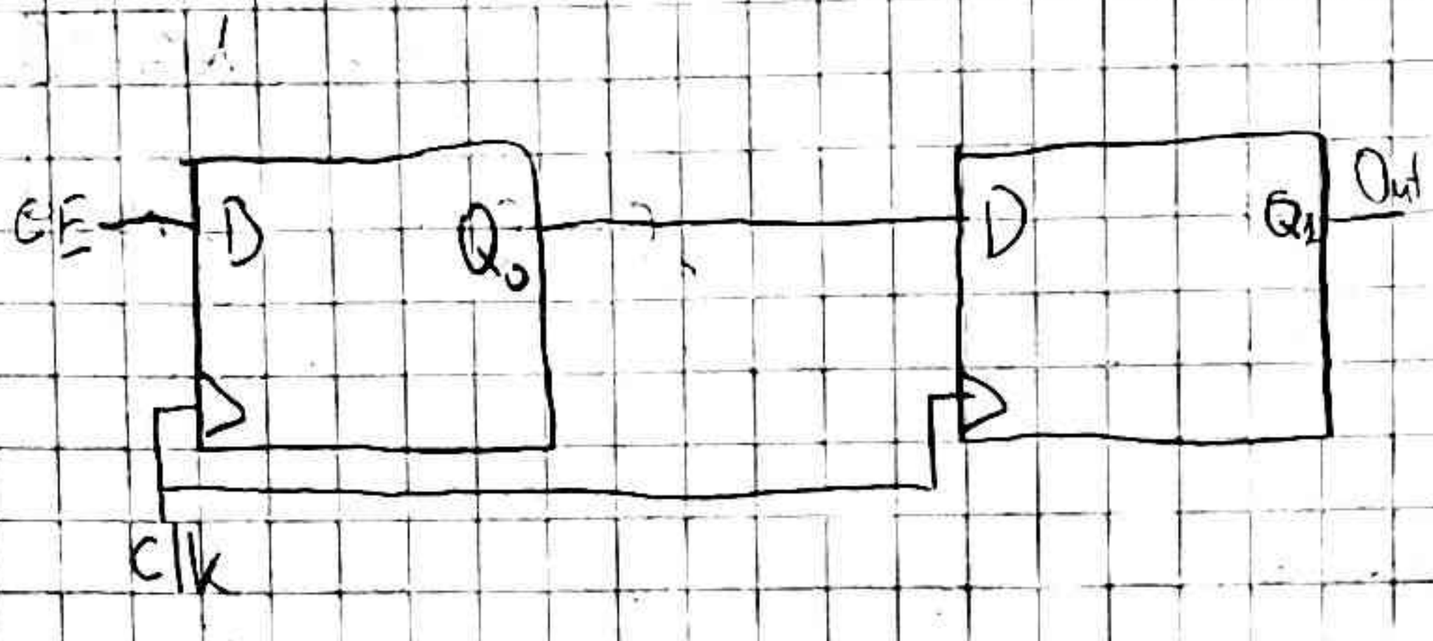
D[0] = (LD & (D_in[0])) | (~LD & UP & (UP ^ Q[0])) |
(~LD & ~UP & DW & (DW ^ Q[0])) |
(~LD & ~UP & ~DW & Q[0])

D[1] = (LD & (D_in[1])) | (~LD & UP & (Q[1] ⊕
⊕ (Q[0] & UP)) | (~LD & ~UP & DW & (Q[1] ⊕ (~Q[0] &
& DW)) | (~LD & ~UP & ~DW & Q[1])

check D_in

D[2] = (LD & (D_in[2])) | (~LD & UP &
& (Q[2] ^ (UP & Q1 & Q0))) | (~LD & ~UP & DW &
& (Q[2] ^ (DW & ~Q[1] & ~Q[0]))) | (~LD & ~UP & ~DW
& (Q[2]))

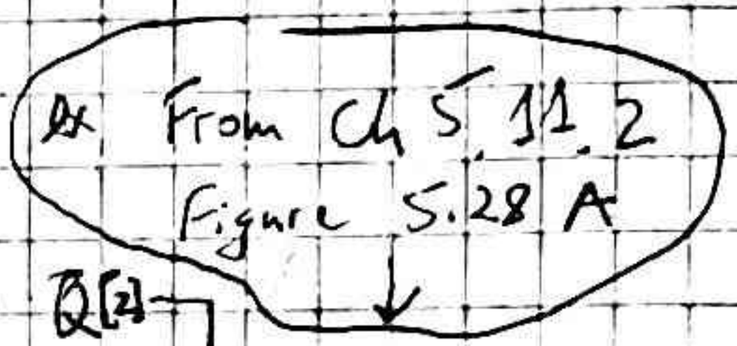
Edge Detector

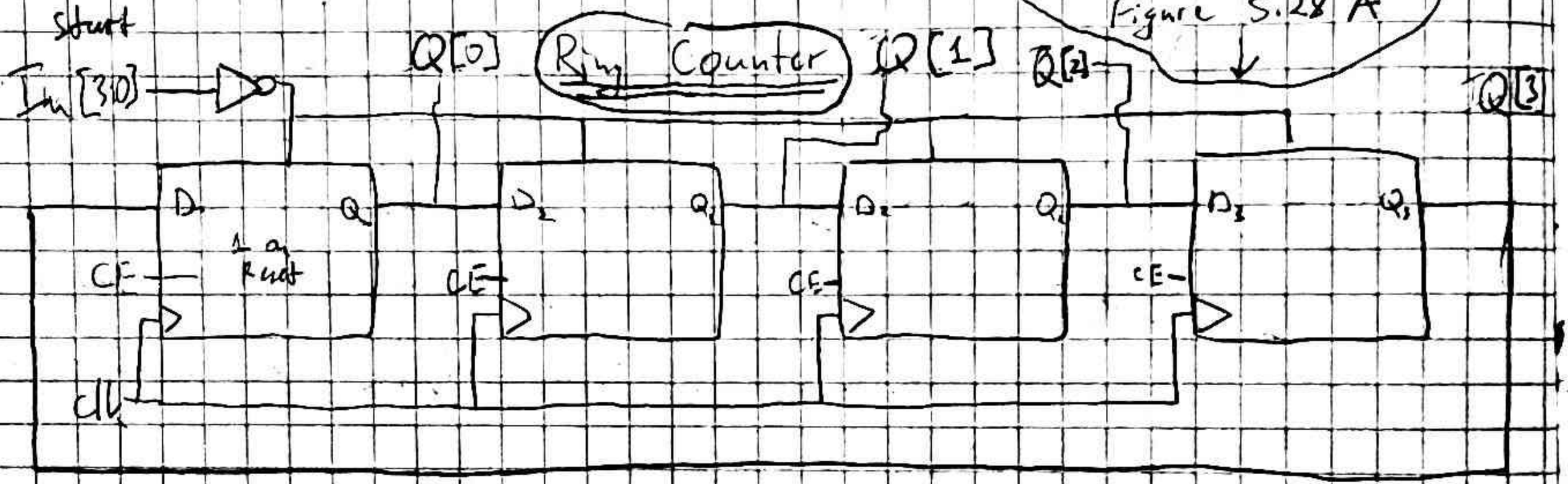


Logic

$\overline{Q_0}$ & Q_1

0 Follow by 1





Is there a LD ? (N_0)

How to put I_n into

$D_0 = Q_3$ $D_1 = Q_0$ $D_2 = Q_2$ $D_3 = Q_2$

5 bit adder

Up:

$D_0 = CE \oplus Q_0$

$D_1 = Q_1 \oplus (CE Q_0)$

$D_2 = Q_2 \oplus (CE Q_1 Q_0)$

$D_3 = Q_3 \oplus (CE Q_2 Q_1 Q_0)$

$D_4 = Q_4 \oplus (CE Q_3 Q_2 Q_1 Q_0)$

Dn:

$D_0 = CE \oplus Q_0$

$D_1 = Q_1 \oplus (CE \overline{Q_0})$

$D_2 = Q_2 \oplus (CE \overline{Q_0}\,\overline{Q_1})$

$D_3 = Q_3 \oplus (CE \overline{Q_0}\,\overline{Q_1}\,\overline{Q_2})$

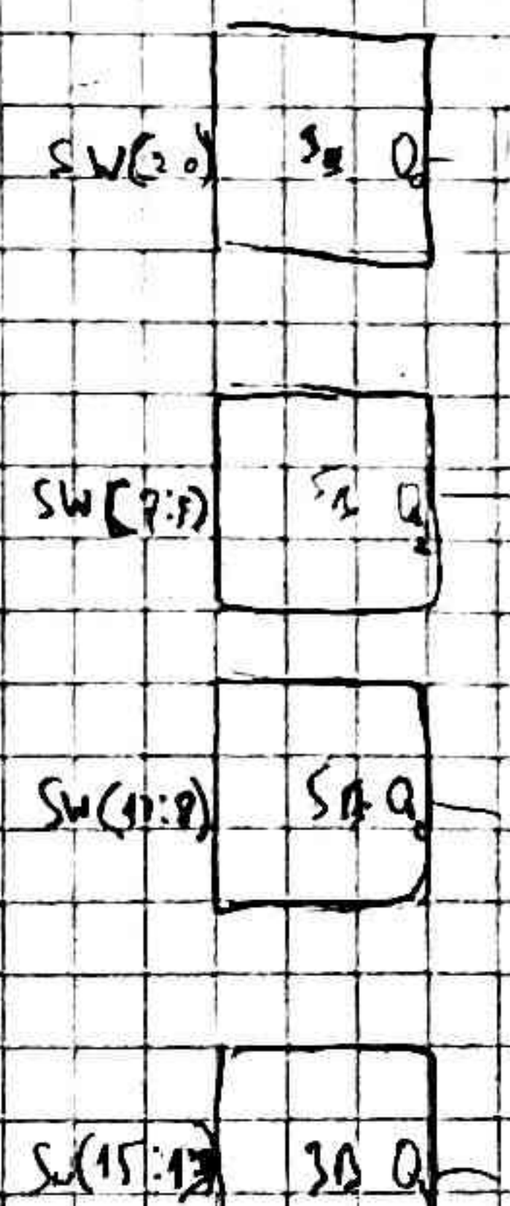
$D_4 = Q_4 \oplus (CE \overline{Q_0}\,\overline{Q_1}\,\overline{Q_2}\,\overline{Q_3})$

D[0], D[1], D[2] are written out on 3-bit counter

D[3] = (LD & D_in[3]) | (~LD & Up & (Q3 ^ (CE & Q2 & Q1 & Q0)) | (~LD & ~Up & Dw & (Q3 ^ (CE & ~Q2 & ~Q1 & ~Q0))) | (~LD & ~Up & ~Dw & Q[3])

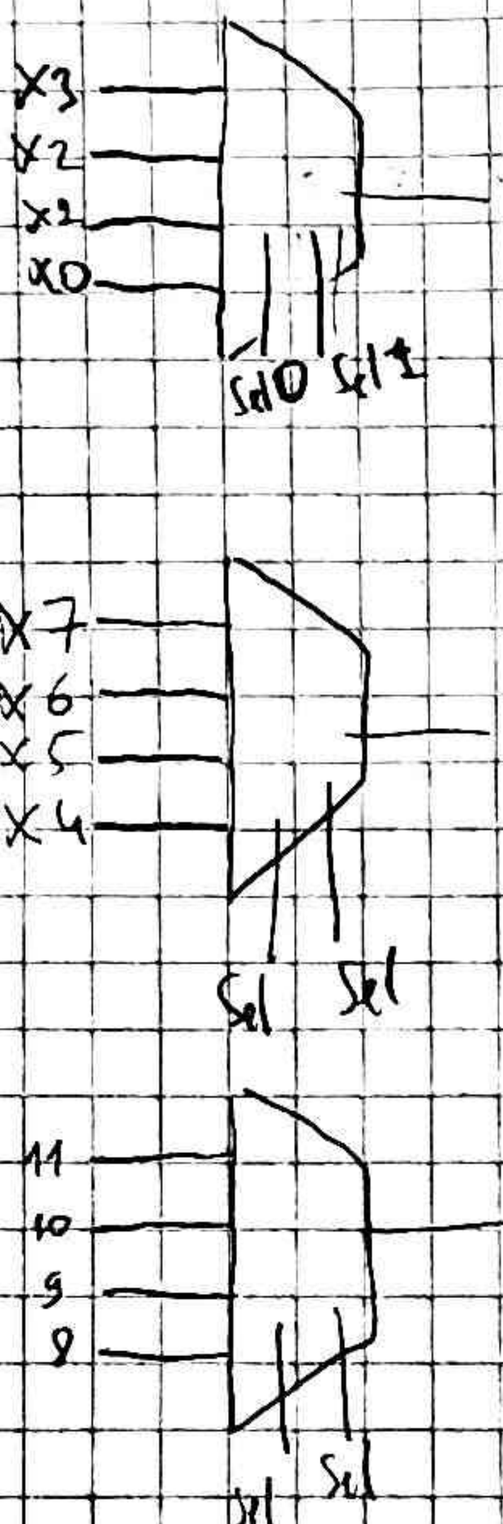
D[4] = (LD & D_in[4]) | (~LD & Up & (Q4 ^ (CE & Q3 & Q2 & Q1 & Q0)) | (~LD & ~Up & Dw & (Q4 ^ (CE & ~Q3 & ~Q2 & ~Q1 & ~Q0))) | (~LD & ~Up & ~Dw & Q4)

16 bit counter



UPs and DWs connected via and gates

Selector



H => 4 bits

sel => 4 bits

$$H = N[15:12] \;\&\; \{4\{(S1:0)^{100}\}\} = 100^{0}$$

$$H = N[11:8] \;\&\; sel[2:0]^{0,00}$$

$$H = N[7:4] \;\&\; sel[1:0]$$

$$H = N[3:0] \;\&\; sel[0]$$

$$H = N[15:12] \;\&\; (sel[1] \;\&\; \sim sel[2] \;\&\; \sim sel[1] \;\&\; \sim sel[0])$$

Top Level

~ (FFFC | FFFD | FFFE | FFFF) & btnC

~(&Q[15:2]) & Btnl - advance up

F

1111 1111 1111 1100

15 ... 2 1 0

Decimal w5

if(Q == FFFF)

leftmost decimal = 0

else if(Q == 0)

right most decimal = 0

Lft D = (~(w5 & (FFFF)) || 0'b1

Rgt D = (w5 & ~(0000)) || 0'b1

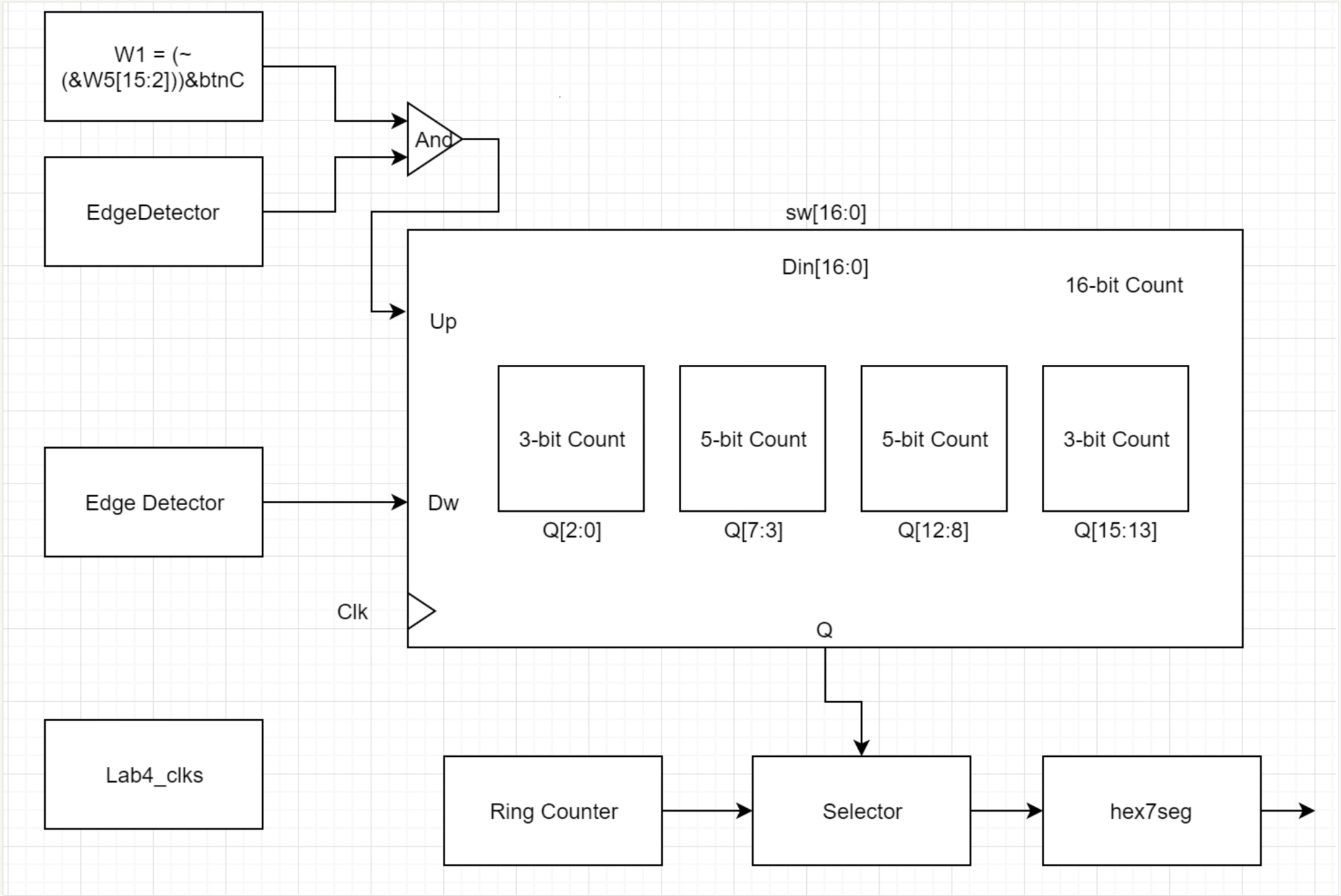
AS

2379

2:04pm

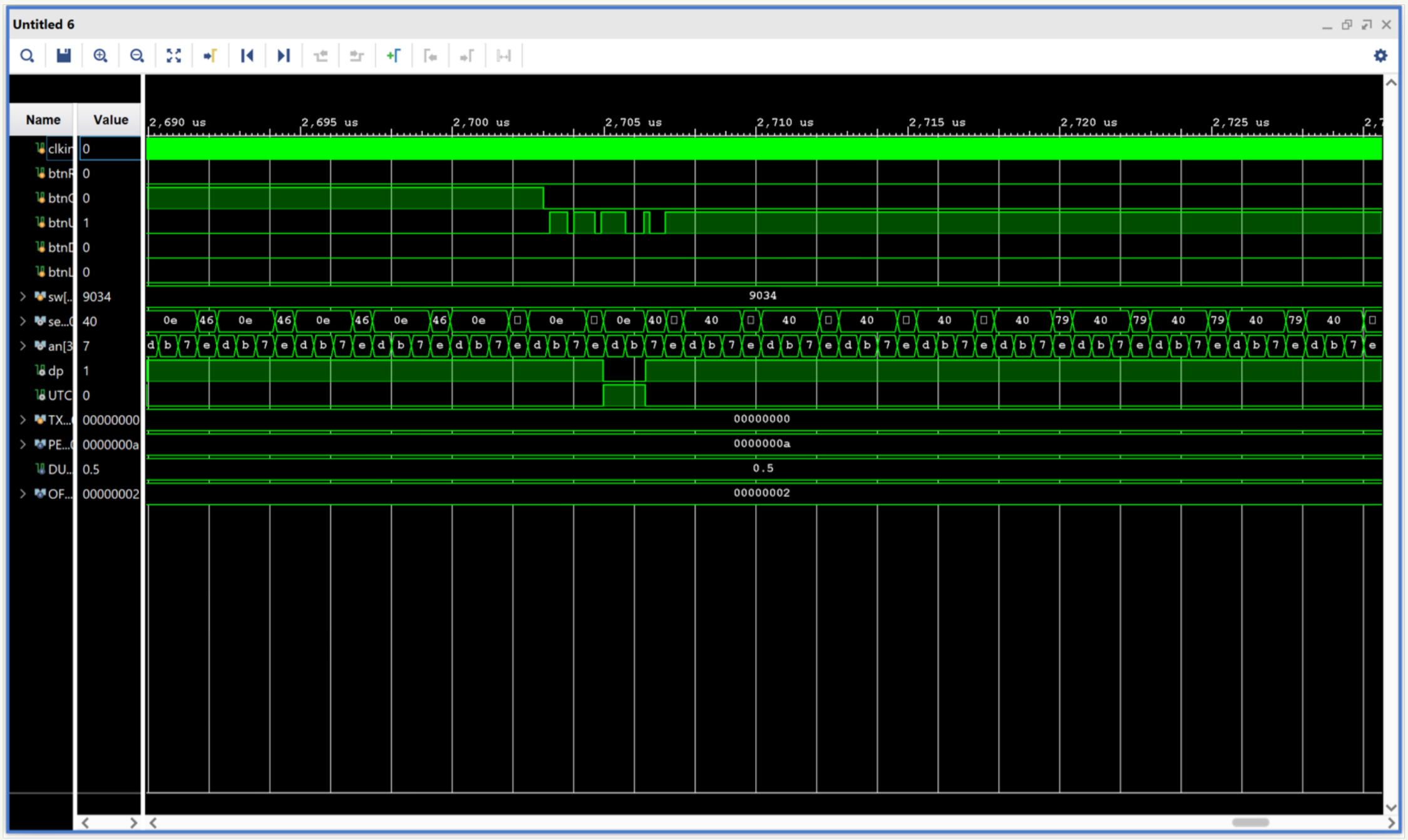
2/1/19

Diagram Of the top

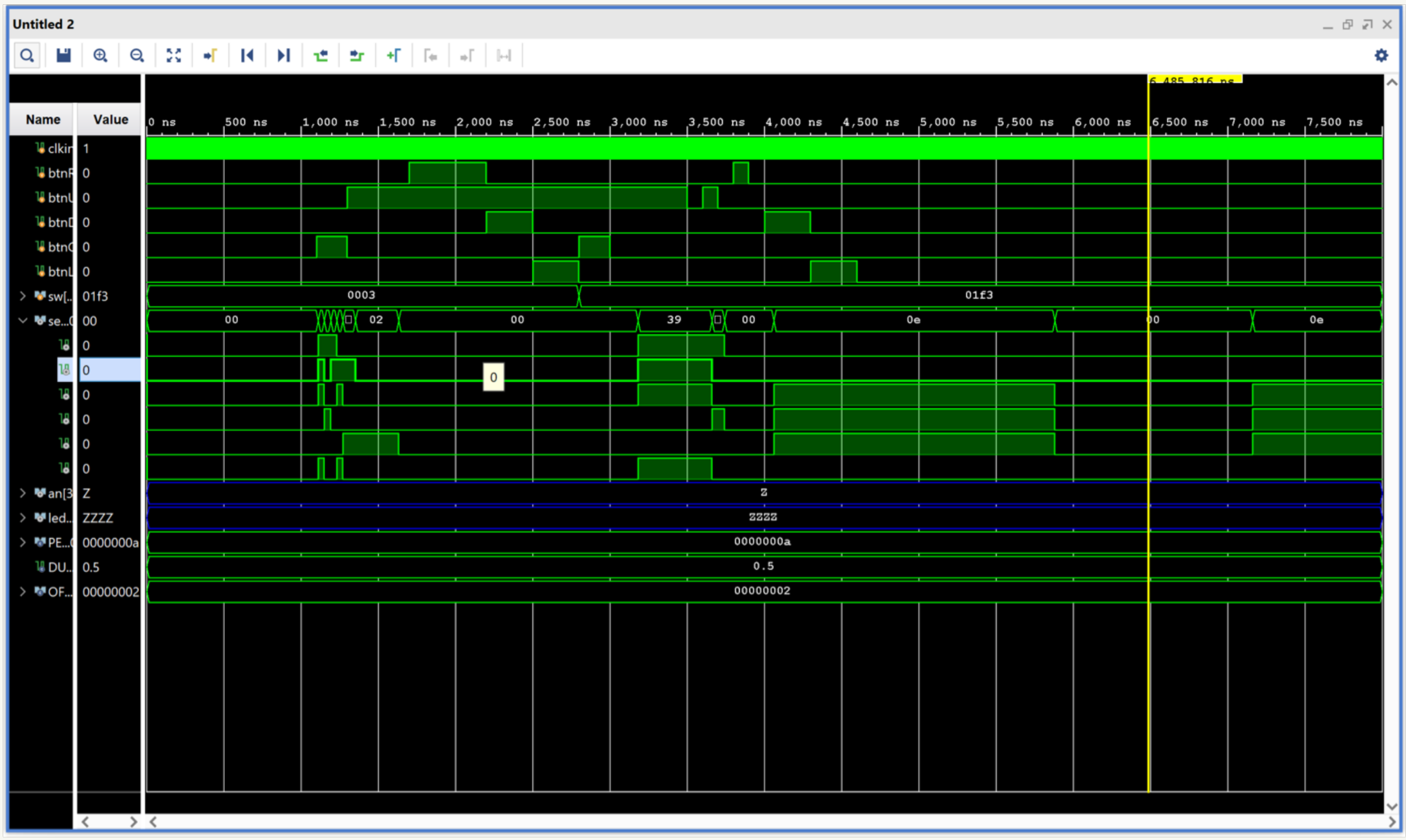


Simmulation

Requested verifier:



My Simmulation



Untitled 4

Name | Value

an[3 | Z
led... | ZZZZ
PE...(0 | 0000000a
DU... | 0.5
OF... | 00000002
Q[1... | 00ff

0 us 1 us 2 us 3 us 4 us 5 us 6 us 7 us 8 us 9 us 10 us 11 us 12 us 13 us 14 us 15 us 16 us 17 us 18 us 19 us 20 us

Z
ZZZZ
0000000a
0.5
00000002
00ff

Untitled 1

Name | Value

clkin | 1
btnR | 0
btnU | 0
btnD | 0
btnC | 1
btnL | 0
sw[... | 0003
se...(0 | 39
an[3 | Z
led... | ZZZZ
PE...(0 | 0000000a
DU... | 0.5
OF... | 00000002

1,050 ns 1,100 ns 1,150 ns 1,200 ns 1,250 ns 1,300 ns 1,350 ns 1,400 ns 1,450 ns 1,500 ns 1,550 ns 1,600 ns 1,650 ns 1,700 ns 1,750 ns 1,800 ns

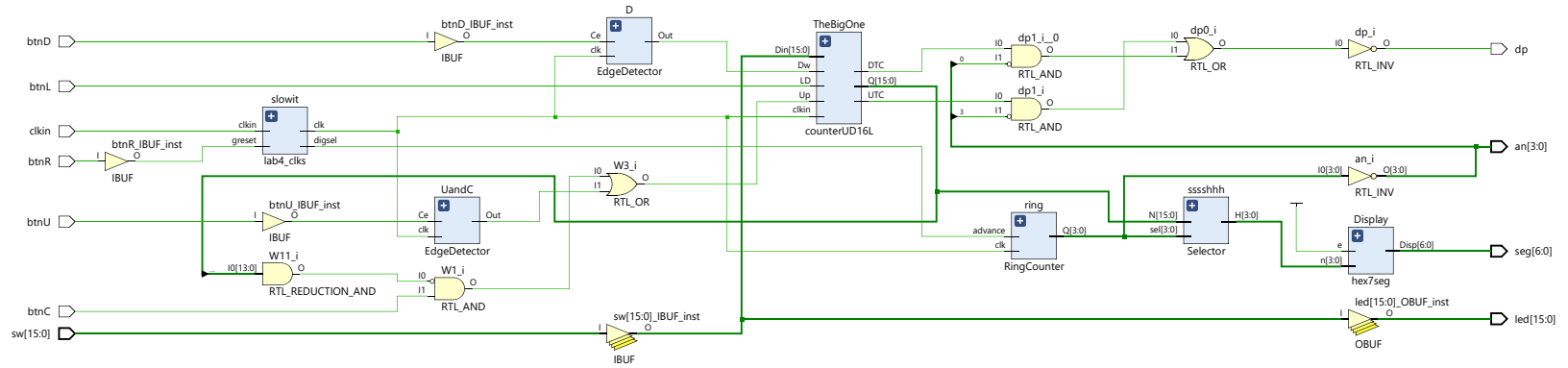
0003
00 39 24 30 19 12 02 00
Z
ZZZZ
0000000a
0.5
00000002

```
`timescale 1ns / 1ps


module TopLevel(
    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


    );
```

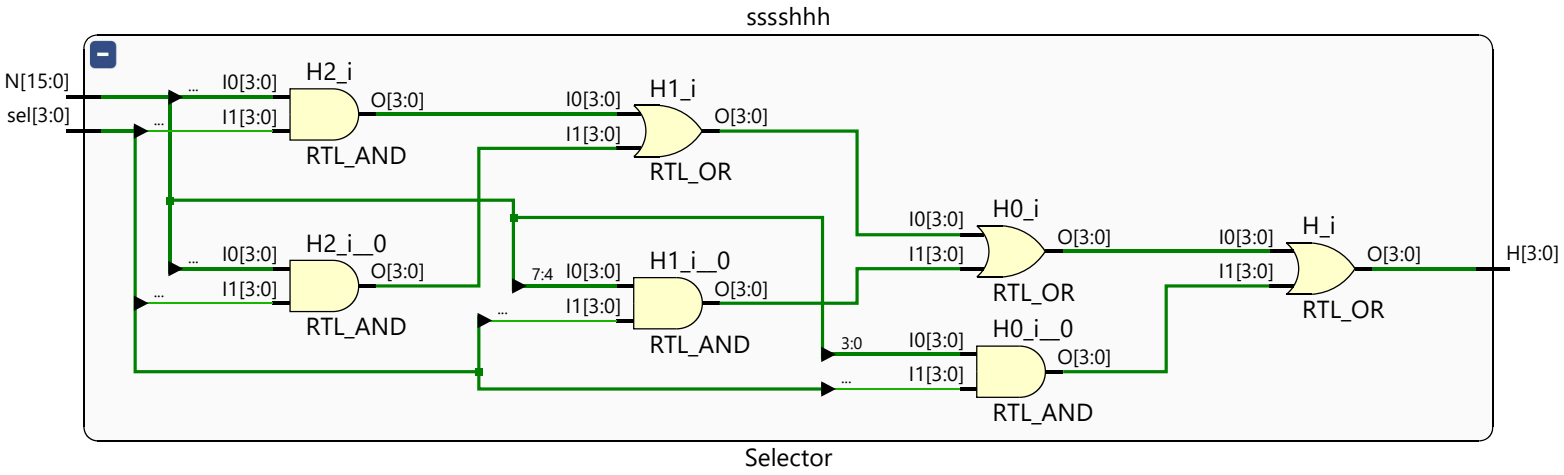
```
    wire W1, W2, W3;

    wire clk;
    wire [15:0] Q;
    wire [15:0]W5;    //For N in Selector
    //ButtonC and edgeDetector
    assign W1 = (~(&W5[15:2]))&btnC;
    EdgeDetector UandC (.clk(clk),
.Ce(btnU), .Out(W2));

    assign W3 = W1|W2;  //Up val in 16 bit

    wire W4;
    EdgeDetector D (.clk(clk), .Ce(btnD),
.Out(W4));  //Dw val in 16 bit


    wire T1, T2;
    counterUD16L TheBigOne (.clkin(clk),
.Dw(W4),.LD(btnL), .Up(W3), .Din(sw),
    .UTC(T1), .DTC(T2), .Q(W5));

    wire [3:0] W6;       //For sel in
```

```
selector


    wire [3:0] W7; //Hex input
    Selector sssshhh (.sel(W6), .N(W5),
.H(W7));
    wire extra;
    hex7seg Display (.n(W7), .e(1'b1),
.Disp(seg));
    wire digsel;
    //Lab4_clks
        assign an = ~W6;

    lab4_clks slowit (.clkin(clkin),
.greset(btnR), .clk(clk), .digsel(digsel));
    RingCounter ring (.clk(clk),
.advance(digsel), .Q(W6));


    //LED
    assign led = sw;

    //Decimal
    assign dp = ~(((T1)& ~an[3])|((T2)&
```

```
~an[0]));



endmodule
```

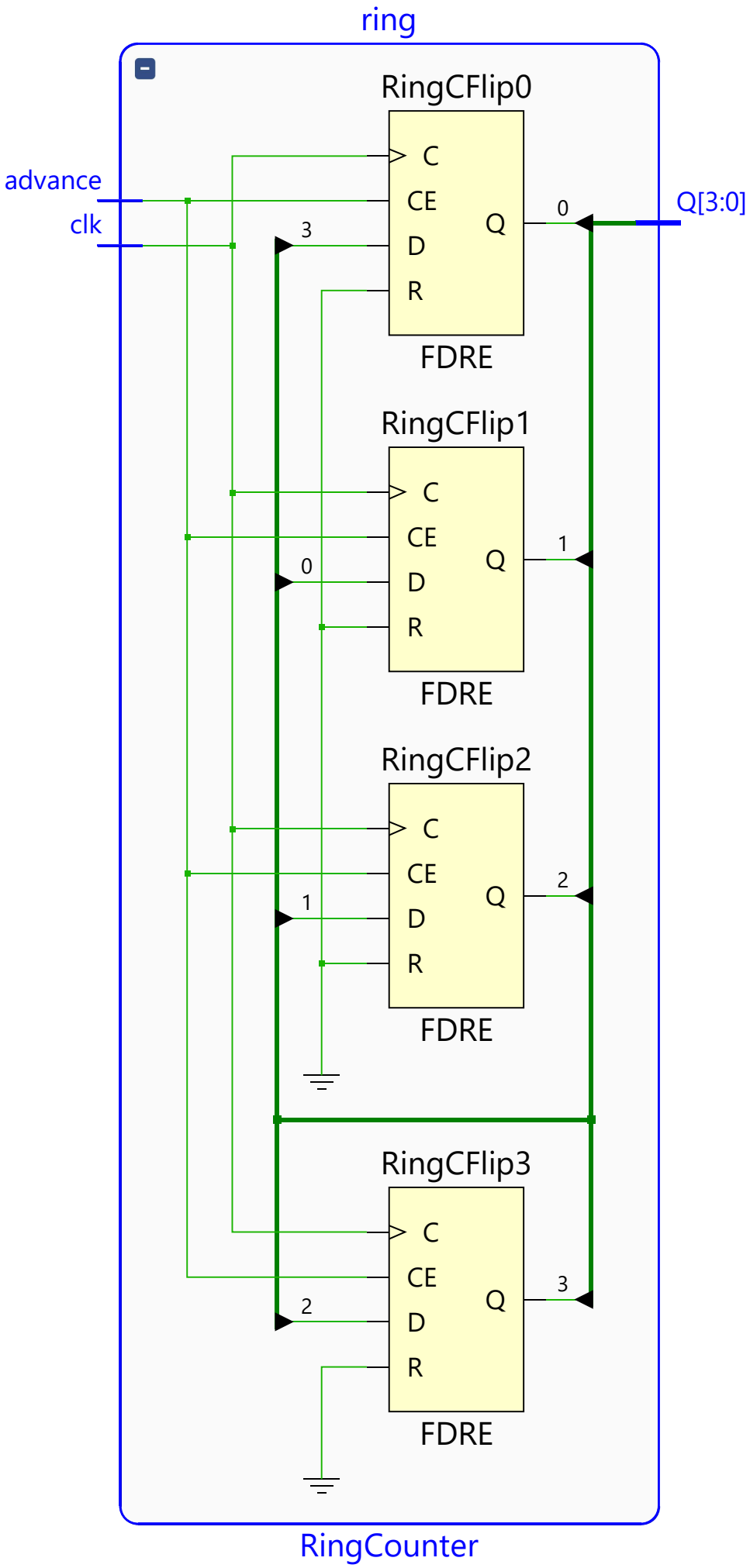

btnD
btnL
clkin
btnR
btnU
btnC
sw[15:0]
btnD_IBUF_inst
IBUF
D
EdgeDetector
Ce
clk
Out
TheBigOne
Din[15:0]
Dw
LD
Up
clkin
DTC
Q[15:0]
UTC
counterUD16L
dp1_i_0
RTL_AND
dp0_i
RTL_OR
dp_i
RTL_INV
dp
slowit
clkin
greset
clk
digsel
lab4_clks
btnR_IBUF_inst
W3_i
RTL_OR
an[3:0]
an_i
I0[3:0]
O[3:0]
RTL_INV
UandC
EdgeDetector
btnU_IBUF_inst
W11_i
I0[13:0]
RTL_REDUCTION_AND
W1_i
RTL_AND
ring
advance
RingCounter
Q[3:0]
sssshhh
N[15:0]
sel[3:0]
H[3:0]
Selector
Display
e
n[3:0]
Disp[6:0]
hex7seg
seg[6:0]
sw[15:0]_IBUF_inst
led[15:0]_OBUF_inst
OBUF
led[15:0]

```
`timescale 1ns / 1ps


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

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

endmodule
```


sssshhh
N[15:0]
sel[3:0]
H2_i
I0[3:0]
I1[3:0]
O[3:0]
RTL_AND
H1_i
I0[3:0]
I1[3:0]
O[3:0]
RTL_OR
H2_i__0
I0[3:0]
I1[3:0]
O[3:0]
RTL_AND
7:4
H1_i__0
I0[3:0]
I1[3:0]
O[3:0]
RTL_AND
H0_i
I0[3:0]
I1[3:0]
O[3:0]
RTL_OR
3:0
H0_i__0
I0[3:0]
I1[3:0]
O[3:0]
RTL_AND
H_i
I0[3:0]
I1[3:0]
O[3:0]
RTL_OR
H[3:0]
Selector

```
`timescale 1ns / 1ps


module RingCounter(
    input clk,
    input advance,
    output [3:0] Q
    );


    FDRE #(.INIT(1'b1)) RingCFlip0
(.C(clk), .CE(advance), .D(Q[3]),
.Q(Q[0]));
    FDRE #(.INIT(1'b0)) RingCFlip1
(.C(clk), .CE(advance), .D(Q[0]),
.Q(Q[1]));
    FDRE #(.INIT(1'b0)) RingCFlip2
(.C(clk), .CE(advance), .D(Q[1]),
.Q(Q[2]));
    FDRE #(.INIT(1'b0)) RingCFlip3
(.C(clk), .CE(advance), .D(Q[2]),
.Q(Q[3]));
```

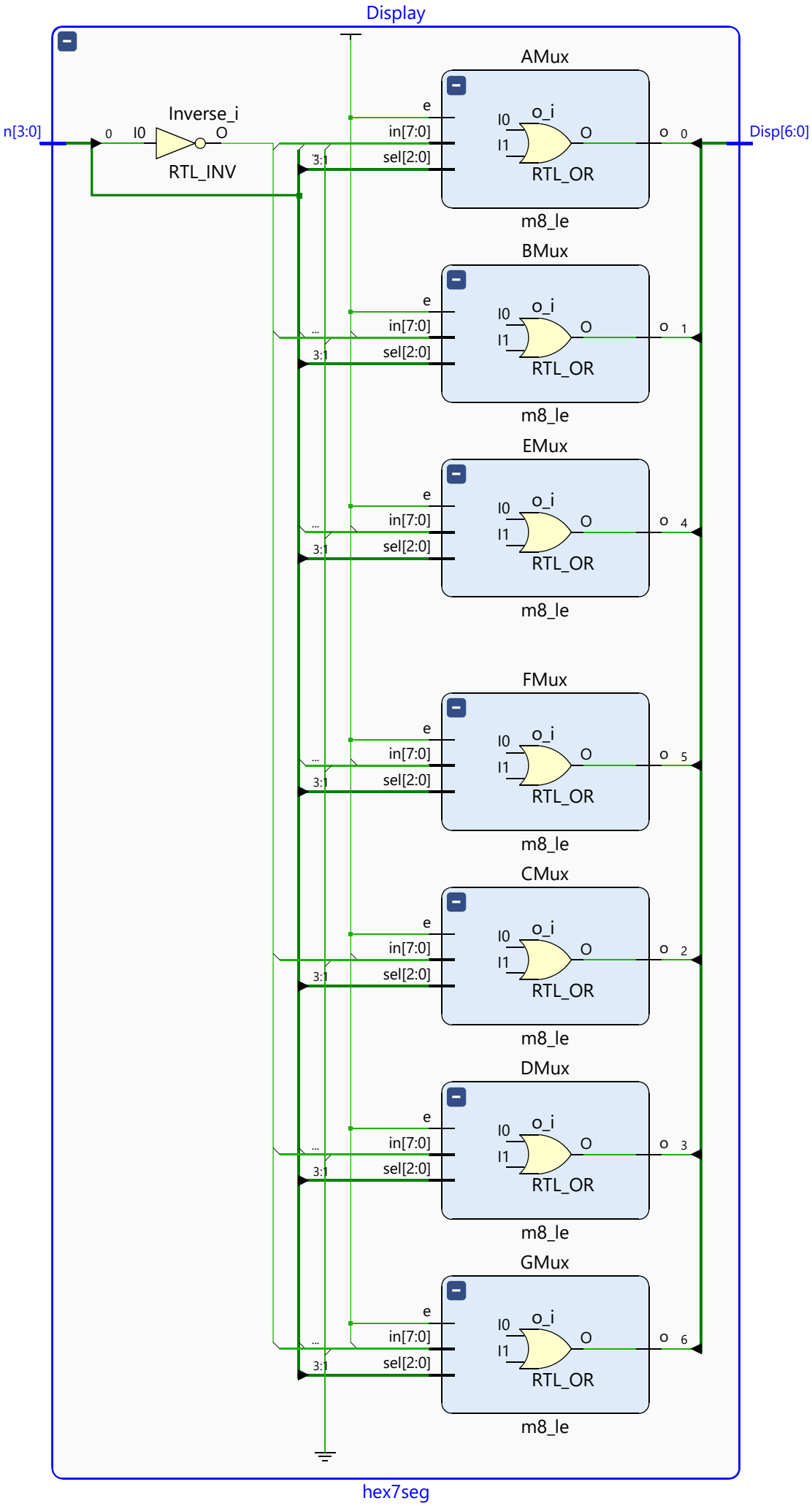
```
endmodule
```


ring
RingCFlip0
C
CE
D
R
Q
FDRE
advance
clk
3
0
Q[3:0]
RingCFlip1
C
CE
D
R
Q
FDRE
0
1
RingCFlip2
C
CE
D
R
Q
FDRE
1
2
RingCFlip3
C
CE
D
R
Q
FDRE
2
3
RingCounter

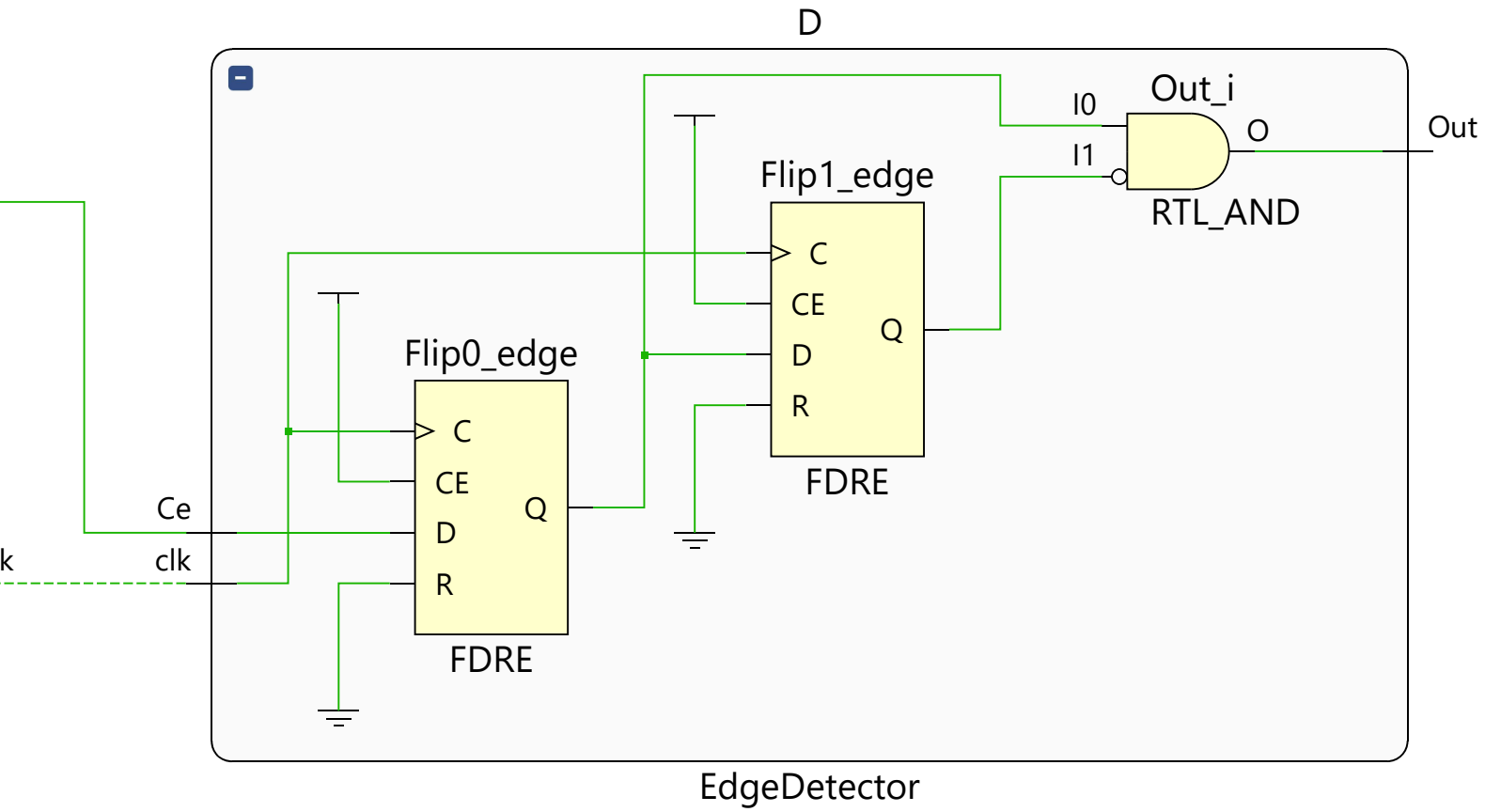
```
`timescale 1ns / 1ps

module hex7seg(
    input [3:0] n,

    input e,

    output [6:0] Disp
    );
    wire Inverse;

    assign Inverse = ~n[0];


    m8_le AMux(.in({1'b0, n[0], n[0],
1'b0, 1'b0, Inverse, 1'b0, n[0]}),
.sel({n[3], n[2], n[1]}), .e(1'b1),
.o(Disp[0]));
    m8_le BMux(.in({1'b1, Inverse, n[0],
1'b0,Inverse, n[0], 1'b0,1'b0}),
.sel({n[3], n[2], n[1]}), .e(1'b1),
.o(Disp[1]));
    m8_le CMux(.in({1'b1, Inverse,
1'b0,1'b0,1'b0,1'b0,Inverse,1'b0}),
```

```
.sel({n[3], n[2], n[1]}),
.e(1'b1),.o(Disp[2]));
    m8_le
DMux(.in({n[0],1'b0,Inverse,1'b0,n[0],Inve
rse,1'b0,n[0]}), .sel({n[3], n[2], n[1]}),
.e(1'b1), .o(Disp[3]));
    m8_le
EMux(.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(1'b1), .o(Disp[4]));
    m8_le FMux(.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(1'b1),
.o(Disp[5]));
    m8_le
GMux(.in({1'b0,Inverse,1'b0,1'b0,n[0],1'b0
,1'b0,1'b1}), .sel({n[3], n[2], n[1]}),
.e(1'b1), .o(Disp[6]));



endmodule
```

```
module m8_le(

input [7:0] in,
input [2:0] sel,
input e,
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
```

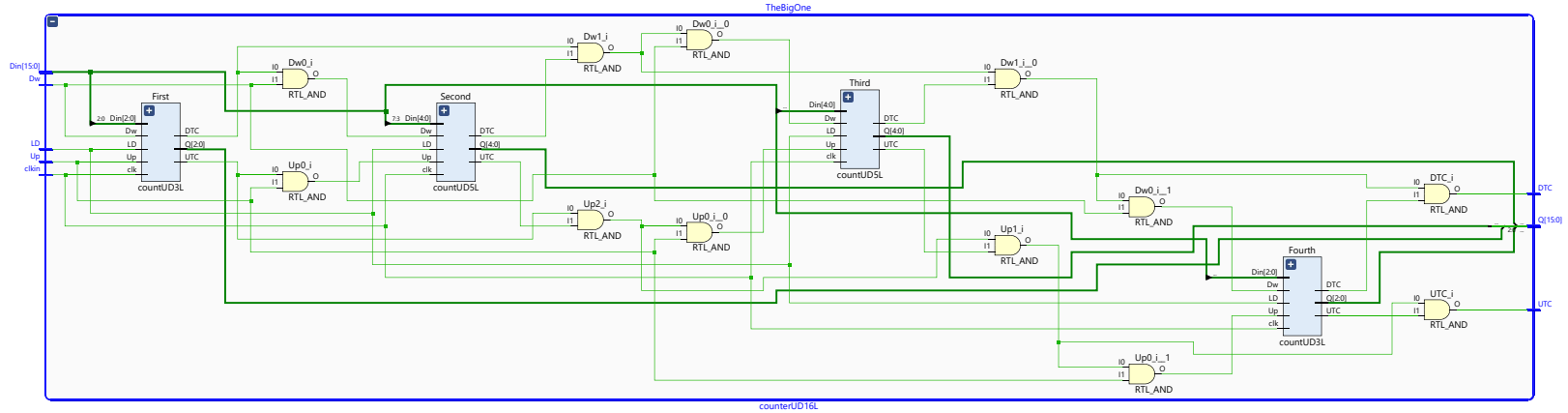

Display
n[3:0]
Disp[6:0]
Inverse_i
RTL_INV
AMux
BMux
EMux
FMux
CMux
DMux
GMux
m8_le
RTL_OR
e
in[7:0]
sel[2:0]
o_i
hex7seg

```
`timescale 1ns / 1ps


module EdgeDetector(
    input clk,
    input Ce,
    output Out
    );

    wire [1:0] Q;

    FDRE #(.INIT(1'b0)) Flip0_edge
(.C(clk), .R(1'b0), .CE(1'b1), .D(Ce),
.Q(Q[0]));
    FDRE #(.INIT(1'b0)) Flip1_edge
(.C(clk), .R(1'b0), .CE(1'b1), .D(Q[0]),
.Q(Q[1]));

    assign Out = Q[0]&~Q[1];

endmodule
```


D
Out_i
I0
I1
O
Out
RTL_AND
Flip1_edge
C
CE
D
R
Q
FDRE
Flip0_edge
C
CE
D
R
Q
FDRE
Ce
clk
EdgeDetector

```
`timescale 1ns / 1ps

module counterUD16L(
    input clkin,
    input [15:0] Din,
    input Up,
    input Dw,
    input LD,



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

     //wire clk;
     wire [3:0] D;
     wire [3:0] T;

     countUD3L First(.clk(clkin), .Up(Up),
.Dw(Dw), .LD(LD), .Din(Din[2:0]),
.Q(Q[2:0]), .UTC(D[0]), .DTC(T[0]));
```

```
    countUD5L Second(.clk(clkin),
.Up(D[0]&Up), .Dw(T[0]&Dw), .LD(LD),
.Din(Din[7:3]), .Q(Q[7:3]), .UTC(D[1]),
.DTC(T[1]));
    countUD5L Third(.clk(clkin),
.Up(D[0]&D[1]&Up), .Dw(T[0]&T[1]&Dw),
.LD(LD), .Din(Din[12:8]), .Q(Q[12:8]),
.UTC(D[2]), .DTC(T[2]));
    countUD3L Fourth(.clk(clkin),
.Up(D[0]&D[1]&D[2]&Up),
.Dw(T[0]&T[1]&T[2]&Dw), .LD(LD),
.Din(Din[15:13]), .Q(Q[15:13]),
.UTC(D[3]), .DTC(T[3]));
    assign DTC = T[0]&T[1]&T[2]&T[3];
    assign UTC = D[0]&D[1]&D[2]&D[3];
    //assign Q =
D[2:0]&D[7:3]&D[12:8]&D[15:13];


endmodule
```

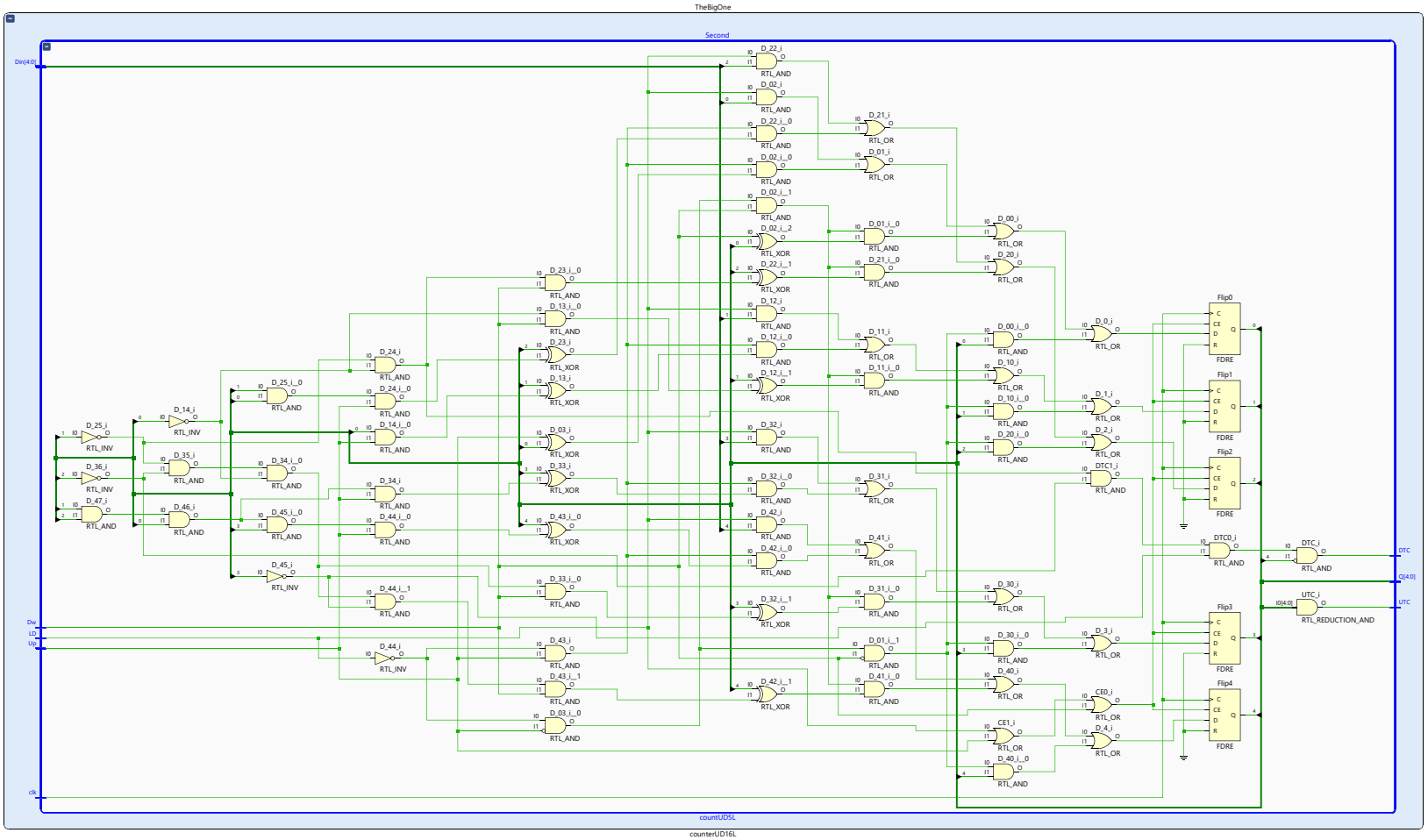

TheBigOne
Din[15:0]
Dw
LD
Up
clkin
First
Din[2:0]
Dw
LD
Up
clk
DTC
Q[2:0]
UTC
countUD3L
Second
Din[4:0]
Q[4:0]
countUD5L
Third
countUD5L
Fourth
countUD3L
Dw0_i
Dw1_i
Up0_i
Up2_i
Dw0_i_0
Dw1_i_0
Up0_i_0
Up1_i
Dw0_i_1
Up0_i_1
DTC_i
UTC_i
RTL_AND
I0
I1
O
DTC
Q[15:0]
UTC
counterUD16L

```
`timescale 1ns / 1ps


module countUD5L(

    input clk,
    input [4:0] Din,
    input Up,
    input Dw,
    input LD,

    //For the logic
    //input Ce,


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

    );


        //wire [2:0] Q;
    wire [4:0] D;
```

```
    //assign Ce = 1'b1;

    //UTC trigger
    assign UTC = &Q;

    //DTC trigger
    assign DTC = ~Q[0] & ~Q[1] & ~Q[2]&
~Q[3]& ~Q[4]& ~Q[5];



    //For counting up
    FDRE #(.INIT(1'b0)) Flip0
(.C(clk),.R(1'b0), .CE(LD|Up|Dw),
.D(D[0]), .Q(Q[0]));
    assign D[0] =
(LD&Din[0])|(~LD&Up&(Up^Q[0]))|(~LD&~Up&Dw
&(Dw^Q[0]))|(~LD&~Up&~Dw&Q[0]);

    FDRE #(.INIT(1'b0)) Flip1
(.C(clk),.R(1'b0), .CE(LD|Up|Dw),
.D(D[1]), .Q(Q[1]));
    assign D[1] =
```

```
(LD&Din[1])|(~LD&Up&(Q[1]^(Q[0]&
Up)))|(~LD&~Up&Dw&(Q[1]^(~Q[0]&Dw)))|(~LD&
~Up&~Dw&Q[1]);

    FDRE #(.INIT(1'b0)) Flip2
(.C(clk),.R(1'b0), .CE(LD|Up|Dw),
.D(D[2]), .Q(Q[2]));
    assign D[2] =
(LD&Din[2])|(~LD&Up&(Q[2]^(Q[1]&Q[0]&Up)))
|(~LD&~Up&Dw&(Q[2]^(~Q[1]&~Q[0]&Dw)))|(~LD
&~Up&~Dw&Q[2]);

    FDRE #(.INIT(1'b0)) Flip3
(.C(clk),.R(1'b0), .CE(LD|Up|Dw),
.D(D[3]), .Q(Q[3]));
    assign D[3] =
(LD&Din[3])|(~LD&Up&(Q[3]^(Q[1]&Q[2]&Q[0]&
Up)))|(~LD&~Up&Dw&(Q[3]^(~Q[1]&~Q[2]&~Q[0]
&Dw)))|(~LD&~Up&~Dw&Q[3]);

    FDRE #(.INIT(1'b0)) Flip4
(.C(clk),.R(1'b0), .CE(LD|Up|Dw),
.D(D[4]), .Q(Q[4]));
    assign D[4] =
```

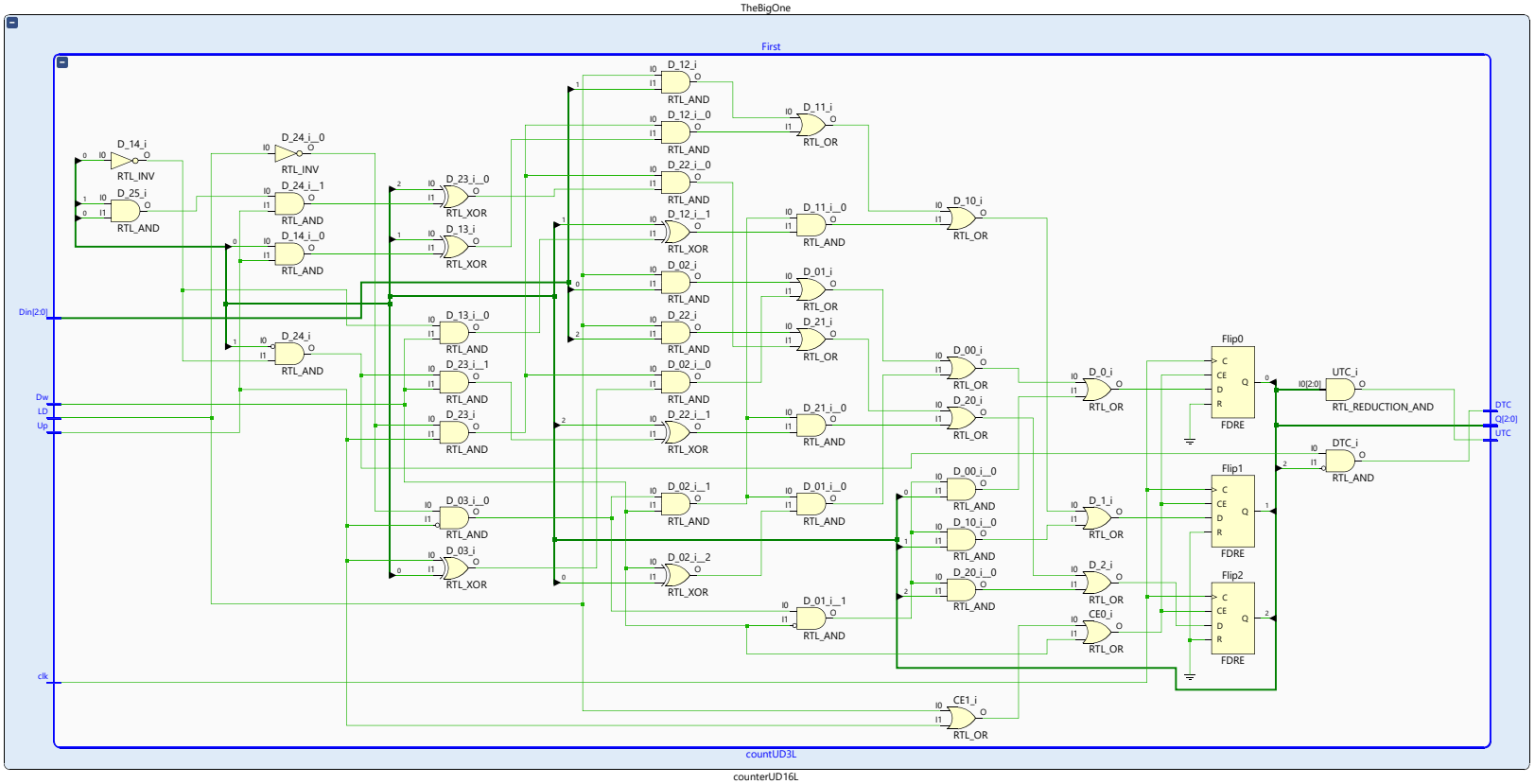
```
(LD&Din[4])|(~LD&Up&(Q[4]^(Q[1]&Q[2]&Q[0]&
Q[3]&Up)))|(~LD&~Up&Dw&(Q[4]^(~Q[1]&~Q[2]&
~Q[0]&~Q[3]&Dw)))|(~LD&~Up&~Dw&Q[4]);



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


TheBigOne
Second
Din[4:0]
Dw
LD
Up
clk
DTC
Q[4:0]
UTC
countUD5L
counterUD16L

```
`timescale 1ns / 1ps


module countUD3L(
    input clk,
    input [2:0] Din,
    input Up,
    input Dw,
    input LD,

    //For the logic
    //input Ce,


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

    );

    //wire [2:0] Q;
    wire [2:0] D;

    //assign Ce = 1'b1;
```

```
    //UTC trigger
    assign UTC = &Q;

    //DTC trigger
    assign DTC = ~Q[0] & ~Q[1] & ~Q[2];



    //For counting up
    FDRE #(.INIT(1'b0)) Flip0 (.C(clk),
.R(1'b0), .CE(LD|Up|Dw), .D(D[0]),
.Q(Q[0]));
    assign D[0] =
(LD&Din[0])|(~LD&Up&(Up^Q[0]))|(~LD&~Up&Dw
&(Dw^Q[0]))|(~LD&~Up&~Dw&Q[0]);

    FDRE #(.INIT(1'b0)) Flip1 (.C(clk),
.R(1'b0), .CE(LD|Up|Dw), .D(D[1]),
.Q(Q[1]));
    assign D[1] =
(LD&Din[1])|(~LD&Up&(Q[1]^(Q[0]&
Up)))|(~LD&~Up&Dw&(Q[1]^(~Q[0]&Dw)))|(~LD&
~Up&~Dw&Q[1]);
```

```
    FDRE #(.INIT(1'b0)) Flip2
(.C(clk),.R(1'b0), .CE(LD|Up|Dw),
.D(D[2]), .Q(Q[2]));
    assign D[2] =
(LD&Din[2])|(~LD&Up&(Q[2]^(Q[1]&Q[0]&Up)))
|(~LD&~Up&Dw&(Q[2]^(~Q[1]&~Q[0]&Dw)))|(~LD
&~Up&~Dw&Q[2]);

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

endmodule
```


TheBigOne
First
countUD3L
counterUD16L
Din[2:0]
Dw
LD
Up
clk
DTC
Q[2:0]
UTC
D_14_i
RTL_INV
D_25_i
RTL_AND
D_24_i_0
RTL_INV
D_24_i_1
RTL_AND
D_14_i_0
RTL_AND
D_24_i
RTL_AND
D_23_i_0
RTL_XOR
D_13_i
RTL_XOR
D_13_i_0
RTL_AND
D_23_i_1
RTL_AND
D_23_i
RTL_AND
D_03_i_0
RTL_AND
D_03_i
RTL_XOR
D_12_i
RTL_AND
D_12_i_0
RTL_AND
D_22_i_0
RTL_AND
D_12_i_1
RTL_XOR
D_02_i
RTL_AND
D_22_i
RTL_AND
D_02_i_0
RTL_AND
D_22_i_1
RTL_XOR
D_02_i_1
RTL_AND
D_02_i_2
RTL_XOR
D_11_i
RTL_OR
D_11_i_0
RTL_AND
D_01_i
RTL_OR
D_21_i
RTL_OR
D_21_i_0
RTL_AND
D_01_i_0
RTL_AND
D_01_i_1
RTL_AND
D_10_i
RTL_OR
D_00_i
RTL_OR
D_20_i
RTL_OR
D_00_i_0
RTL_AND
D_10_i_0
RTL_AND
D_20_i_0
RTL_AND
CE1_i
RTL_OR
D_0_i
RTL_OR
D_1_i
RTL_OR
D_2_i
RTL_OR
CE0_i
RTL_OR
Flip0
FDRE
Flip1
FDRE
Flip2
FDRE
UTC_i
RTL_REDUCTION_AND
DTC_i
RTL_AND