Frank I Gomez
5/3/21
Lab 4, Section D.
ID: 1650550
figomez
CSE 100L

Lab 4 Write Up

1. Description

The purpose of this lab was to create a 16bit counter, using flip flops to store the current value of the counter. We were allowed to use only the system clock obtained from the lab4_clks file, not clkin or digsel, unless otherwise specified.

2. Design

- Top Level

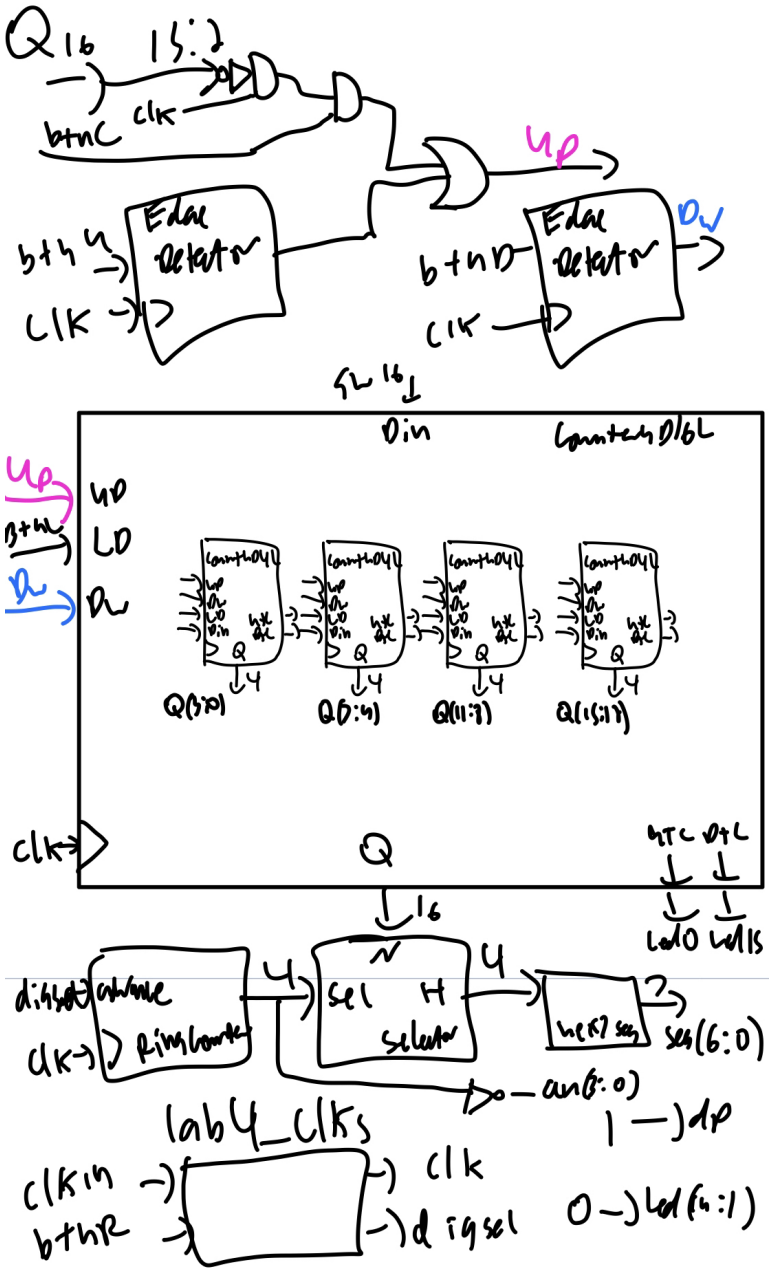


Diagram 1

Even more stuff going on this time around. First thing to note is lab4_clks. That module creates the clk input used pretty much everywhere in this lab. Now, starting from the top down, those double-AND gates at the top handle when btnC is down, and they're OR'd with the output of the btnU edge detector, since they're both used for handling an "Up" input. Likewise, next to them, btnD uses an edge detector for the same reason, and they all input into counterUD16L, alongside

btnL. BtnL corresponds to LD, which, when active, will load the 16bit binary value stored in the 16 switches into the counter. Inside the 16bit counter are 4 4bit counters, which store the current counter value, and output UTC and DTC at F and 0 respectively. When all 4 counters return UTC or DTC, the corresponding leds (0 and 15) will be lit on the display. The 16 bit value then leaves the counter, being forwarded into the Selector. The selector iterates between 4 bits of the value, thanks to the input from RingCounter, and then passes said value to hex7seg, which does the same thing as the last lab, displaying the hex value. For brevity, two ring counters were technically used, as a second ring counter iterates the 4 AN values as well, changing which digit of the seven segment display is lit, but it's included in the diagram as it is to save space. Finally, leds 1-14 are set to 0, and dp is set to 1, powering them off.

- counterUD16L

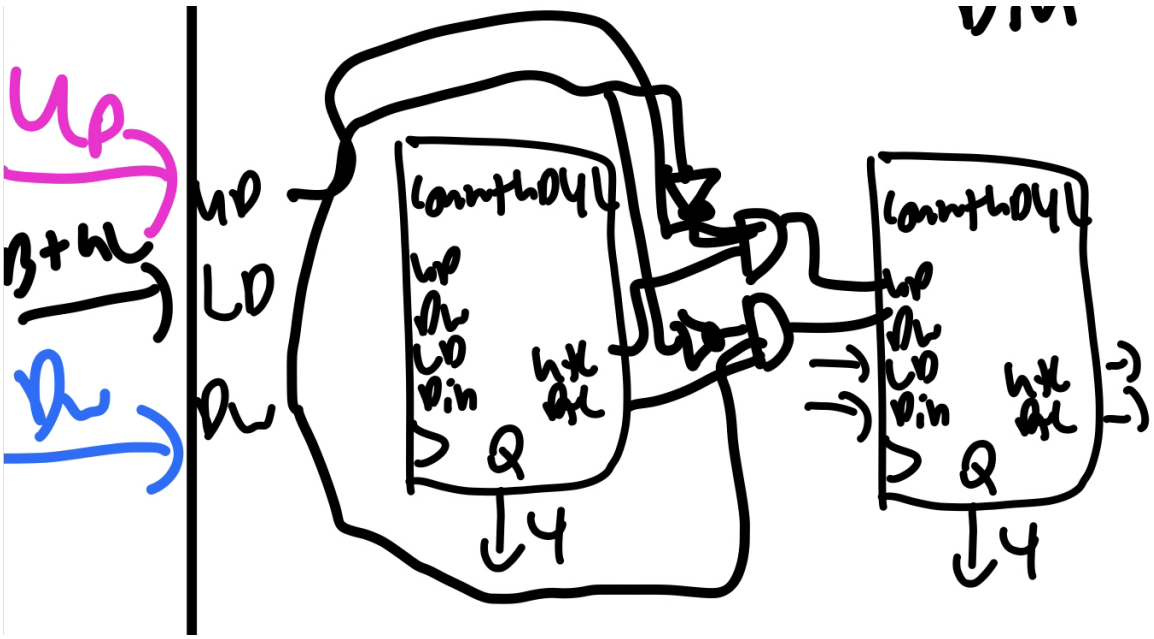


Diagram 2

The part I want to explain here is the difficulty behind getting counters beyond the first to increment/decrement. They only increase when an Up value is received alongside the ALL the previous UTC values, and only if Down is not being pressed at the same time. The same follows for Down, DTC, and not Up. While it makes sense, it leads to the diagram looking like this mess, three times over by the end.

- countUD4L

As for counting Up and Down at the local level, there's actually a really nice equation provided in class for this exact thing. $D_n = Q_n \oplus (Q_{n-1} * Q_{n-2} * \ldots * Up)$ and $D_n = Q_n \oplus (!Q_{n-1} * !Q_{n-2} * \ldots * Dw)$. Using this, each bit will only flip when all the values before it are 1, or when all the values before it are 0. There's a little more logic, checking if LD is pressed, which overrides all iteration, in addition to doing nothing if Up and Down are held at the same time. These values are then loaded into the flip flops. UTC and DTC are high at all 1s and all 0s respectively.

- edge_detector

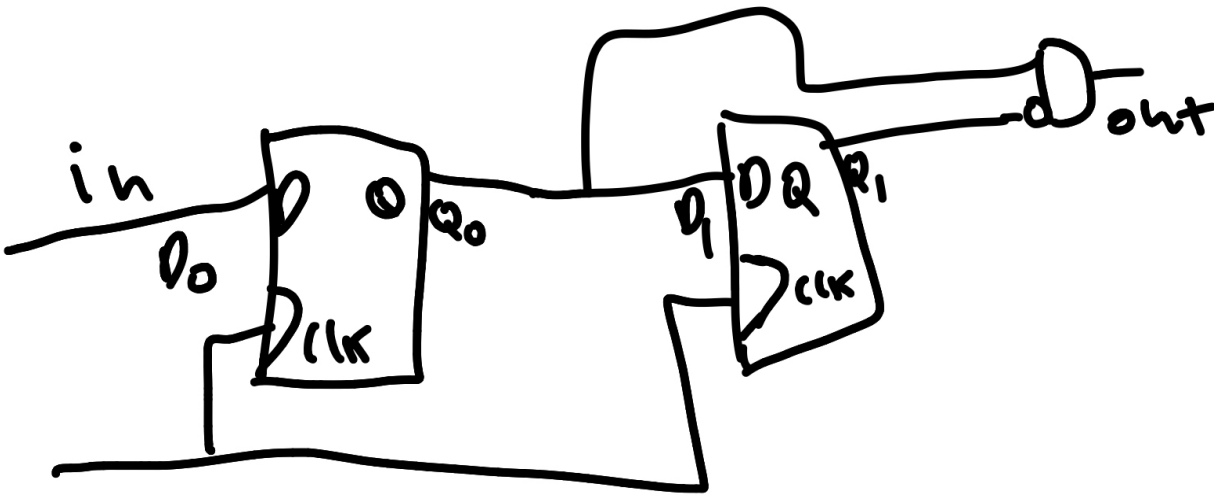


Diagram 3

Now for something a little simpler, and feasible to display, the edge detector. Simply put, when the edge detector receives a 0, followed by a 1, it will return a 1. Two 1s, two 0s, and a 1 then a 0 will return 0.

- ring_counter

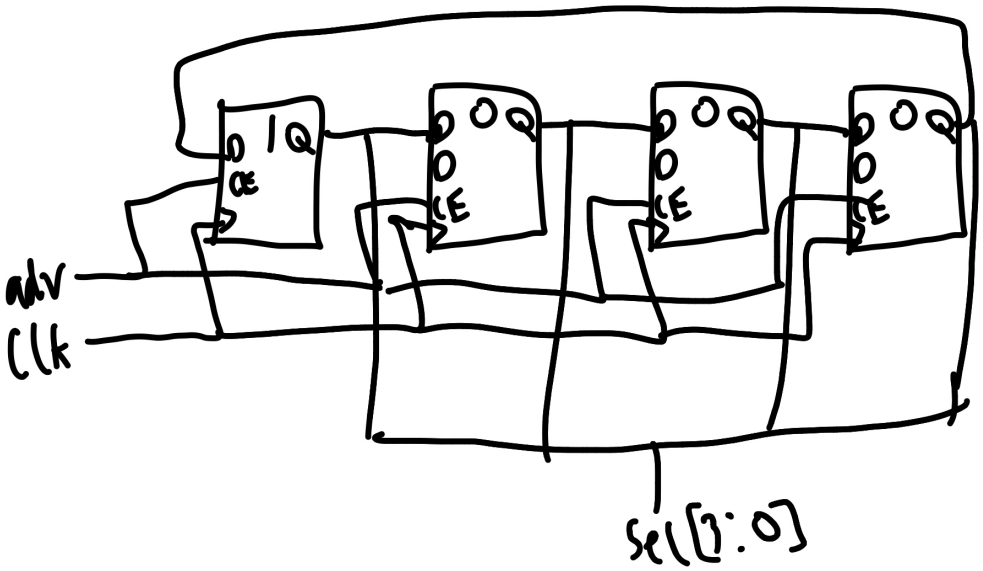


Diagram 4

Also an easy explanation. There's 4 flip flops, with one of them initialized with a value of 1. Every clock cycle, the output of each flip flop will be passed to the next, with the last flip flop passing back to the first flip flop, shifting the single bit around between them. This is the only module to make use of digsel instead of clk.

- Selector

Here's how the selector works; it takes a 16bit bus from the 16bit counter, and a 4bit bus from the ring counter. The ring counter will always be 0001, 0010, 0100, or 1000. Using these 4bits, the selector divides the 16bit input into 4 groups of 4bits, and passes those 4bits to the hex7seg. It does this fast enough so that on the hardware, all 4 values are displayed at once.

- lab4_clks

Takes in verilog's clock and breaks it into a cycle of rising and falling edges, to allow the entire file to function on a cycle. Also provides digsel for the ring counter.

3. Testing and Simulation

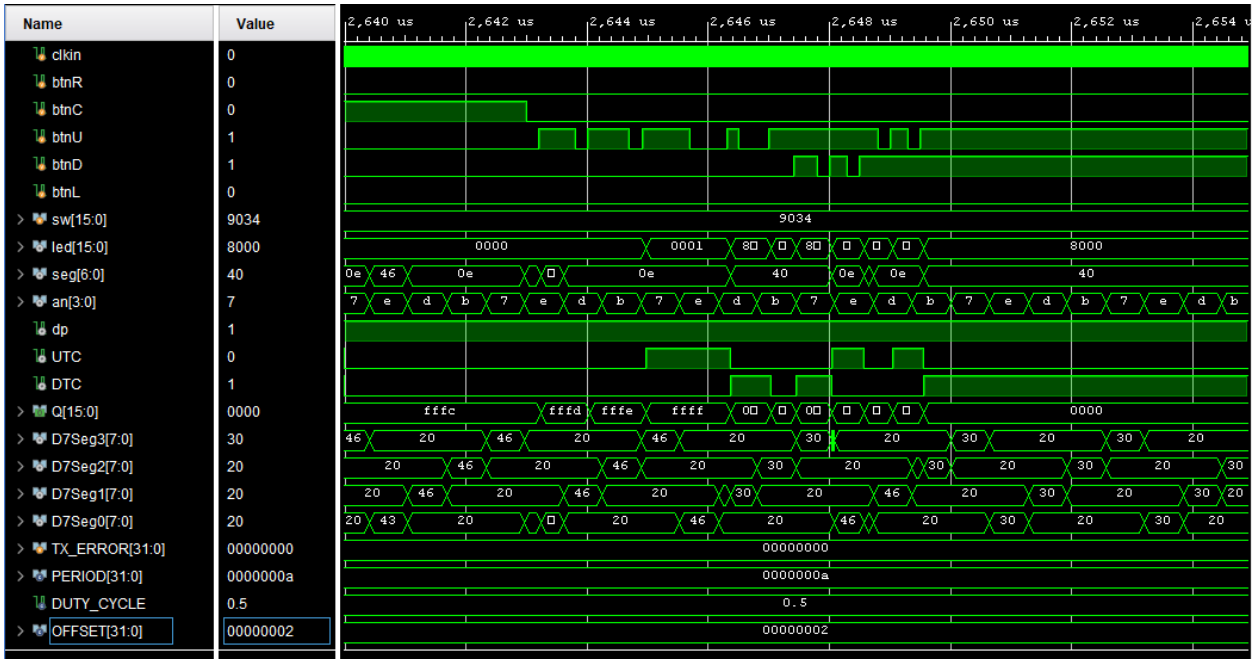
This lab contained some especially frustrating bugs including, but not limited to: btnC adding an additional count beyond FFFC, btnC working on the simulator but not on the board, misunderstanding of how flip flops worked, creating race conditions, and more of the sort. In the end, nearly all of them were sorted thanks to the assistance of the lab TAs or the professor's lecture slides, as many of my issues (such as my logic for counting), were actually given by the professor in lecture, and if I had been paying closer attention, I could have saved myself a lot of time. Otherwise, the simulation went alright once I rewrote all the parts I didn't understand the first time around.

4. Conclusion

In this lab we learned how to make use of flip flops to store values, allowing us to create a counter that doesn't need to have itself as an input, accidentally causing a race condition. Honestly, all my gripes with this lab would have been solved if I had just paid more attention in lecture, as my 4bit counter took up more time than any other module, and it was literally given to us in lecture. No comments on what should be changed, if a student doesn't listen in lecture then it's their loss. Makes one wonder if my problems with labs 2 and 3 could have been for the same reason.

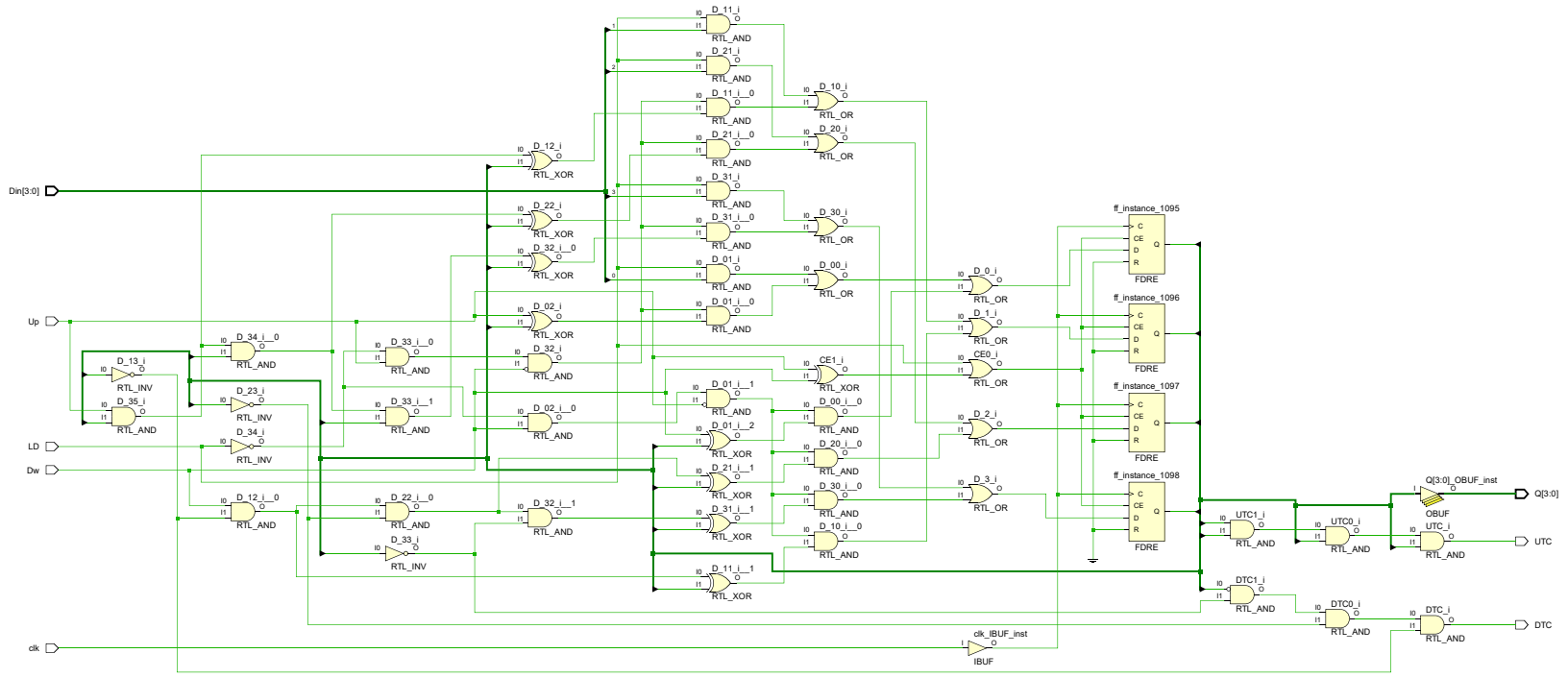
5. Appendix

(Note: Waveform Viewer cannot be printed to PDF, and as such, it is added here as a screenshot.)
(Note: hex7seg and m8_1e are from lab 3, and as such, will not be included.)



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/25/2021 07:25:15 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,
    input Dw,
    input LD,
    input [3:0] Din,
    output [3:0] Q,
    output UTC,
    output DTC
    );
    //wire [3:0] G;
    wire [3:0] D;
    wire [3:0] up;
    wire [3:0] down;

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

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

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

    FDRE #(.INIT(1'b0) ) ff_instance_1095 (.C(clk), .CE(LD | (Up ^ Dw)), .D(D[0]),
.Q(Q[0]));
    FDRE #(.INIT(1'b0) ) ff_instance_1096 (.C(clk), .CE(LD | (Up ^ Dw)), .D(D[1]),
.Q(Q[1]));
    FDRE #(.INIT(1'b0) ) ff_instance_1097 (.C(clk), .CE(LD | (Up ^ Dw)), .D(D[2]),
.Q(Q[2]));
    FDRE #(.INIT(1'b0) ) ff_instance_1098 (.C(clk), .CE(LD | (Up ^ Dw)), .D(D[3]),
.Q(Q[3]));



    assign UTC = Q[3] & Q[2] & Q[1] & Q[0];
    assign DTC = ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0];
endmodule
```


Din[3:0]
Up
LD
Dw
clk
D_13_i
RTL_INV
D_35_i
RTL_AND
D_34_i__0
D_23_i
D_34_i
D_12_i__0
D_33_i__0
D_33_i__1
D_22_i__0
D_33_i
D_12_i
D_22_i
D_32_i__0
D_02_i
D_32_i
D_32_i__1
D_02_i__0
D_11_i
D_21_i
D_11_i__0
D_21_i__0
D_31_i
D_31_i__0
D_01_i
D_01_i__0
D_01_i__1
D_01_i__2
D_21_i__1
D_31_i__1
D_11_i__1
D_10_i
D_20_i
D_30_i
D_00_i
CE1_i
D_00_i__0
D_20_i__0
D_30_i__0
D_10_i__0
D_0_i
D_1_i
CE0_i
D_2_i
D_3_i
ff_instance_1095
ff_instance_1096
ff_instance_1097
ff_instance_1098
FDRE
UTC1_i
UTC0_i
UTC_i
DTC1_i
DTC0_i
DTC_i
Q[3:0]_OBUF_inst
OBUF
Q[3:0]
UTC
DTC
clk_IBUF_inst
IBUF

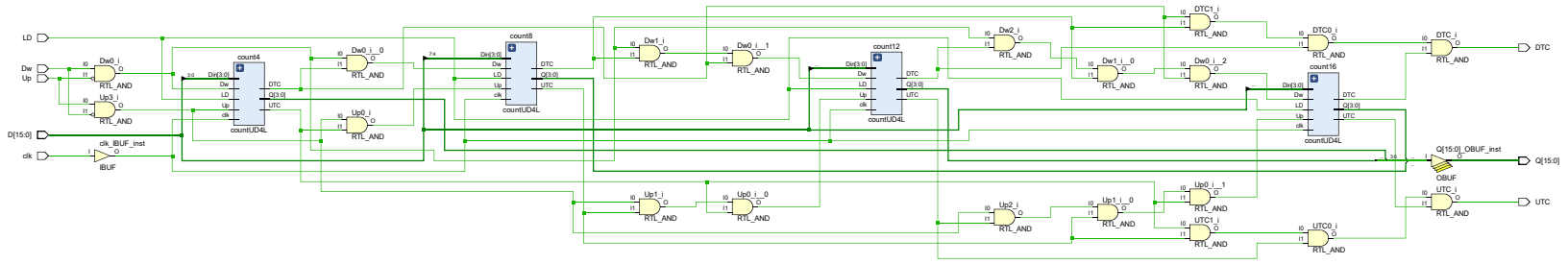
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2021 11:54:00 AM
// Design Name:
// Module Name: countUD4L_testbench
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD4L_testbench();
    reg Up = 1'b0;
    reg Dw = 1'b0;
    reg LD = 1'b0;
    wire [3:0] D = 4'b0000;
    //wire [3:0] C = 4'b0101;
    wire [3:0] Q;
    wire UTC;
    wire DTC;
    reg clkin;
    countUD4L UUT (.clk(clkin), .Up(Up), .Dw(Dw), .LD(LD), .Din(D), .Q(Q),
.UTC(UTC), .DTC(DTC));

    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;

    initial    // Clock process for clkin
    begin
        #OFFSET
          clkin = 1'b1;
        forever
        begin
            #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
```

```
    end
end
initial
begin
 // add your stimuli here
 // to set signal foo to value 0 use
 // foo = 1'b0;
 // to set signal foo to value 1 use
 // foo = 1'b1;
 //always advance time my multiples of 100ns
 // to advance time by 100ns use the following line
  #1000;
  LD = 1'b1;
  #100;
  LD = 1'b0;
  #100;

  Up = 1'b1; //1
  #100;
  Up = 1'b1;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  #100;
  Up = 1'd0;
  #100;
  LD = 1'b1;
  #100; //0
  LD = 1'b0;
  Up = 1'b0;
  #100; //0
  Dw = 1'b1;
  #100;
  #100;
```

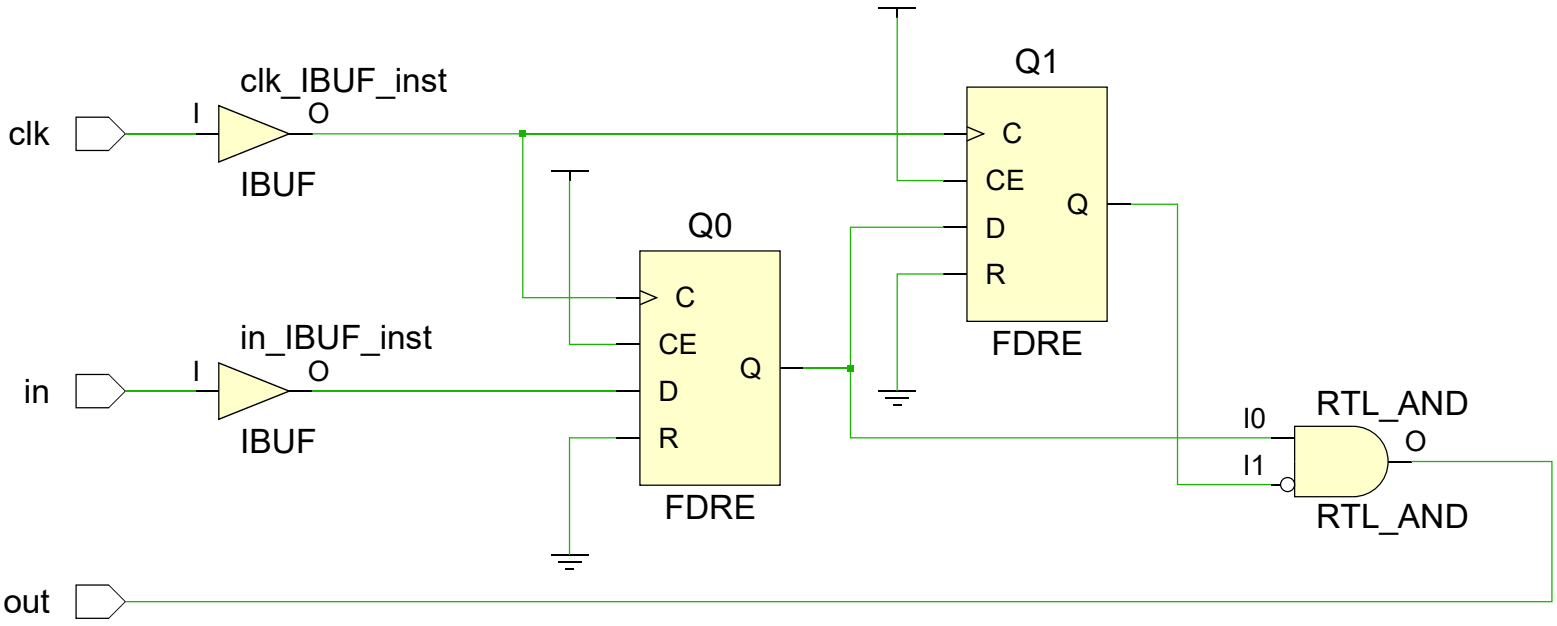
```
      #100;
      #100;
      #100;
      #100;
      #100;
      #100;
      Up = 1'b1;
      #100;
      #100;
      LD = 1'b1;
      #100;
      #100;
      Up = 1'b0;
      Dw = 1'b0;
    end
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/25/2021 09:01:51 PM
// Design Name:
// Module Name: countUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module counterUD16L(
    input clk,
    input Up,
    input Dw,
    input [15:0] D,
    output [15:0] Q,
    output UTC,
    output DTC,
    input LD
    );
    wire UTC0, UTC1, UTC2, UTC3;
    wire DTC0, DTC1, DTC2, DTC3;
    //wire [15:0] R;

    //m2_1x4 sel4(.in0(Q[3:0]), .in1(D[3:0]), .sel(LD), .o(R[3:0]));
    //m2_1x4 sel8(.in0(Q[7:4]), .in1(D[7:4]), .sel(LD), .o(R[7:4]));
    //m2_1x4 sel12(.in0(Q[11:8]), .in1(D[11:8]), .sel(LD), .o(R[11:8]));
    //m2_1x4 sel16(.in0(Q[15:12]), .in1(D[15:12]), .sel(LD), .o(R[15:12]));

    countUD4L count4(.clk(clk), .Up(Up & ~Dw), .Dw(Dw & ~Up), .LD(LD), .Din(D[3:0]),
.Q(Q[3:0]), .UTC(UTC0), .DTC(DTC0));
    countUD4L count8(.clk(clk), .Up(Up & ~Dw & UTC0), .Dw(Dw & ~Up & DTC0), .LD(LD),
.Din(D[7:4]), .Q(Q[7:4]), .UTC(UTC1), .DTC(DTC1));
    countUD4L count12(.clk(clk), .Up(Up & ~Dw & UTC1 & UTC0), .Dw(Dw & ~Up & DTC1 &
```

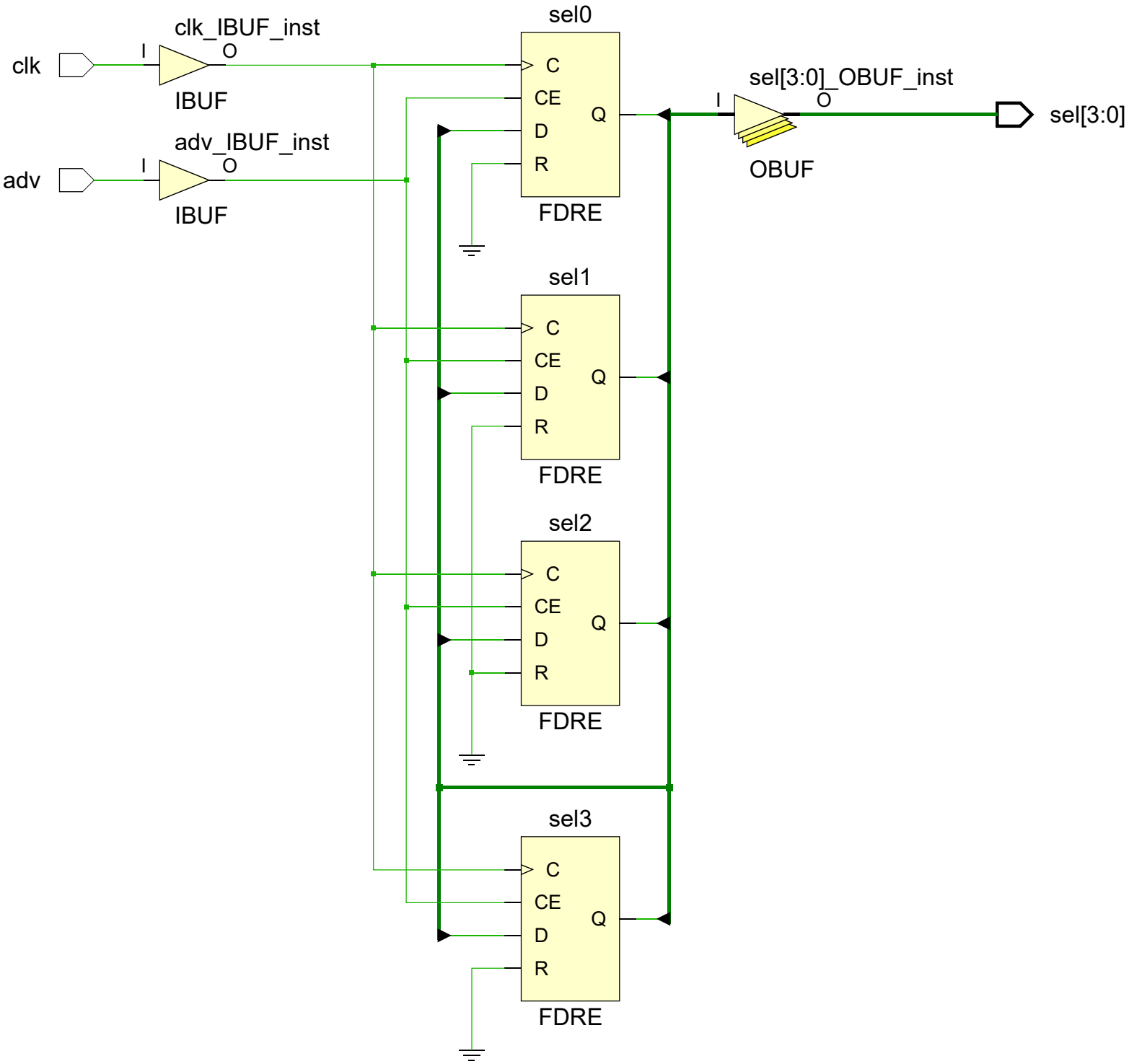
```
DTC0), .LD(LD), .Din(D[11:8]), .Q(Q[11:8]), .UTC(UTC2), .DTC(DTC2));
    countUD4L count16(.clk(clk), .Up(Up & ~Dw & UTC2 & UTC1 & UTC0), .Dw(Dw & ~Up &
DTC2 & DTC1 & DTC0), .LD(LD), .Din(D[15:12]), .Q(Q[15:12]), .UTC(UTC3), .DTC(DTC3));

    assign UTC = UTC0 & UTC1 & UTC2 & UTC3;
    assign DTC = DTC0 & DTC1 & DTC2 & DTC3;

endmodule
```


LD
Dw
Up
D[15:0]
clk
Dw0_i
RTL_AND
Up3_i
IBUF
clk_IBUF_inst
count4
countUD4L
Dw0_i_0
Up0_i
count8
Dw1_i
Dw0_i_1
count12
Dw2_i
Dw1_i_0
DTC1_i
Dw0_i_2
DTC0_i
count16
DTC_i
DTC
Q[15:0]_OBUF_inst
OBUF
Q[15:0]
UTC_i
UTC
Up1_i
Up0_i_0
Up2_i
Up1_i_0
Up0_i_1
UTC1_i
UTC0_i

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2021 10:03:00 PM
// Design Name:
// Module Name: edge_detector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module edge_detector(
    input in,
    input clk,
    input out
    );
    wire D, D2;
    FDRE #(.INIT(1'b0) ) Q0 (.C(clk), .CE(1'b1), .D(in), .Q(D));
    FDRE #(.INIT(1'b0) ) Q1 (.C(clk), .CE(1'b1), .D(D), .Q(D2));
    assign out = D & ~D2;
endmodule
```

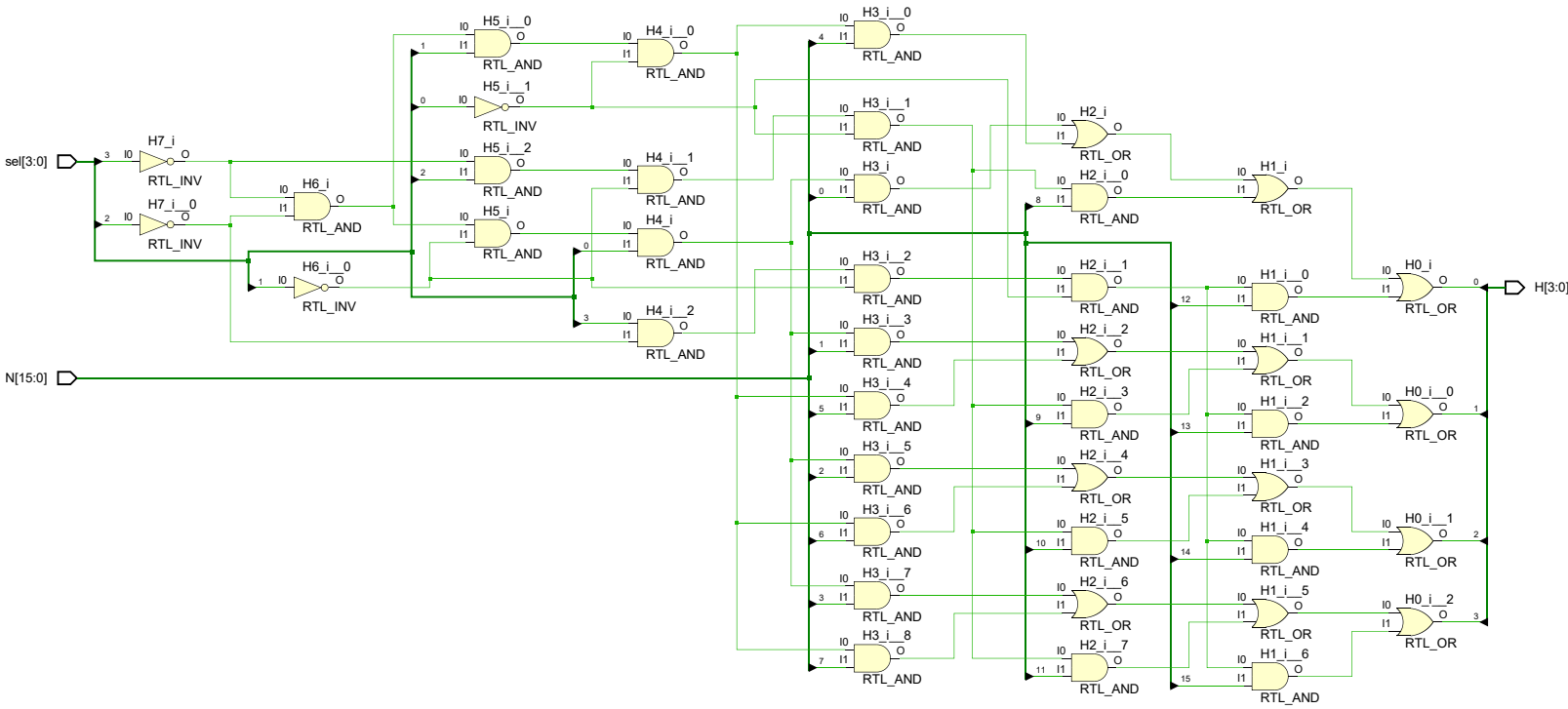

clk
clk_IBUF_inst
I
O
IBUF
in
in_IBUF_inst
I
O
IBUF
Q0
C
CE
D
R
Q
FDRE
Q1
C
CE
D
R
Q
FDRE
RTL_AND
I0
I1
O
RTL_AND
out

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2021 08:02:32 PM
// Design Name:
// Module Name: ring_counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ring_counter(
    input clk,
    input adv,
    output [3:0] sel
    );
    //wire [3:0] A;

    //assign A[0] = (adv & ~sel[1] & ~sel[2] & ~sel[3]);
    //assign A[1] = (adv & ~sel[0] & ~sel[2] & ~sel[3]);
    //assign A[2] = (adv & ~sel[1] & ~sel[0] & ~sel[3]);
    //assign A[3] = (adv & ~sel[1] & ~sel[2] & ~sel[0]);

    FDRE #(.INIT(1'b1) ) sel0 (.C(clk), .R(1'b0), .CE(adv), .D(sel[3]), .Q(sel[0]));
    FDRE #(.INIT(1'b0) ) sel1 (.C(clk), .CE(adv), .D(sel[0]), .Q(sel[1]));
    FDRE #(.INIT(1'b0) ) sel2 (.C(clk), .CE(adv), .D(sel[1]), .Q(sel[2]));
    FDRE #(.INIT(1'b0) ) sel3 (.C(clk), .CE(adv), .D(sel[2]), .Q(sel[3]));
endmodule
```


clk
clk_IBUF_inst
I
O
IBUF
adv
adv_IBUF_inst
I
O
IBUF
sel0
C
CE
D
R
Q
FDRE
sel1
C
CE
D
R
Q
FDRE
sel2
C
CE
D
R
Q
FDRE
sel3
C
CE
D
R
Q
FDRE
sel[3:0]_OBUF_inst
I
O
OBUF
sel[3:0]

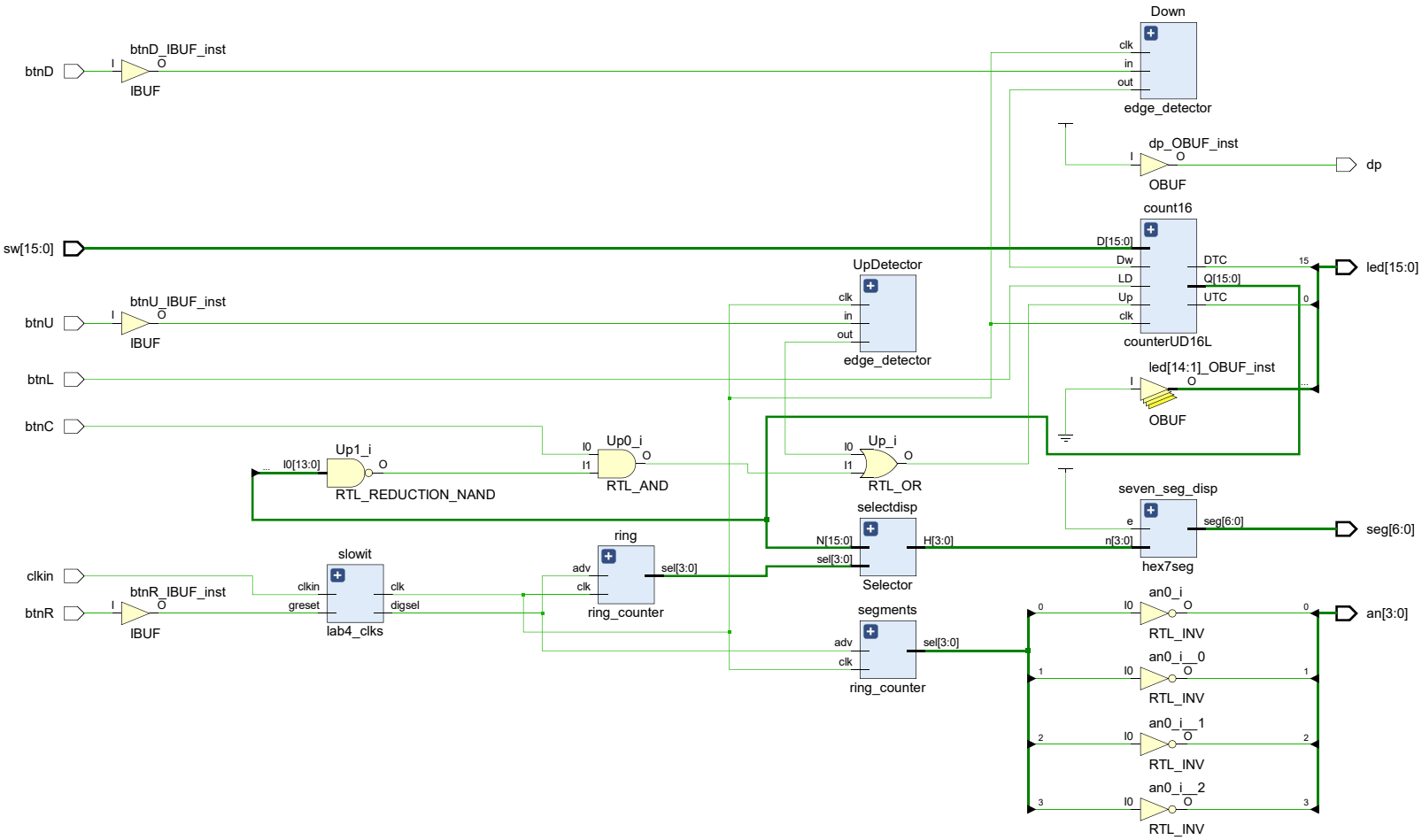
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2021 07:49:37 PM
// Design Name:
// Module Name: Selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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


sel[3:0]
N[15:0]
H[3:0]
H7_i
H7_i__0
H6_i
H6_i__0
H5_i__0
H5_i__1
H5_i__2
H5_i
H4_i__0
H4_i__1
H4_i
H4_i__2
H3_i__0
H3_i__1
H3_i
H3_i__2
H3_i__3
H3_i__4
H3_i__5
H3_i__6
H3_i__7
H3_i__8
H2_i
H2_i__0
H2_i__1
H2_i__2
H2_i__3
H2_i__4
H2_i__5
H2_i__6
H2_i__7
H1_i
H1_i__0
H1_i__1
H1_i__2
H1_i__3
H1_i__4
H1_i__5
H1_i__6
H0_i
H0_i__0
H0_i__1
H0_i__2
RTL_INV
RTL_AND
RTL_OR

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2021 10:10:45 PM
// Design Name:
// Module Name: lab4_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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 Up, Dw, LD, UTC, DTC, digsel, clk;
    wire [15:0] Q;
    wire [3:0] sel;
    wire [3:0] H;
    wire UpQ, UpEdge;
    wire [3:0] an_dummy;

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

```
    edge_detector UpDetector (.clk(clk), .in(btnU), .out(UpEdge));
    //continuous_blocker UpBlocker (.clk(clk), .C(btnC), .Q(Q), .Up(UpQ));
    //assign UpQ = btnC & clk & (~(&Q[15:2] & ((Q[1] & Q[0]) | (Q[1] & ~Q[0]) |
(~Q[1] & Q[0]) | (~Q[1] & ~Q[0]))));
    //continuous_blocker ContinuousDetector (.clk(clk), .C(btnC), .in(clk & (~(&Q[15:
    //& ~((~Q[2] & ~Q[1] & ~Q[0]) | (~Q[2] & ~Q[1] & Q[0]) | (~Q[2] & Q[1] &
~Q[0])))))), .out(UpQ));
    assign Up = UpEdge | (btnC & ~&Q[15:2]);

    edge_detector Down (.clk(clk), .in(btnD), .out(Dw));


    counterUD16L count16 (.clk(clk), .Up(Up), .Dw(Dw), .D(sw), .Q(Q), .UTC(UTC),
.DTC(DTC), .LD(btnL));


    ring_counter ring (.clk(clk), .adv(digsel), .sel(sel));
    ring_counter segments(.clk(clk), .adv(digsel), .sel(an_dummy));
    Selector selectdisp (.N(Q), .sel(sel), .H(H));
    hex7seg seven_seg_disp (.n(H), .e(1'b1), .seg(seg));


    assign dp = 1'b1;
    assign an[0] = ~an_dummy[0];
    assign an[1] = ~an_dummy[1];
    assign an[2] = ~an_dummy[2];
    assign an[3] = ~an_dummy[3];
    //Flip these if they function normally.
    assign led[0] = UTC;
    assign led[15] = DTC;
    assign led[14:1] = 1'b0;

endmodule
```


btnD
btnD_IBUF_inst
I
O
IBUF
Down
clk
in
out
edge_detector
dp_OBUF_inst
dp
OBUF
count16
sw[15:0]
D[15:0]
Dw
LD
Up
clk
DTC
Q[15:0]
UTC
15
0
led[15:0]
counterUD16L
UpDetector
btnU
btnU_IBUF_inst
edge_detector
led[14:1]_OBUF_inst
btnL
btnC
Up1_i
I0[13:0]
RTL_REDUCTION_NAND
Up0_i
I0
I1
RTL_AND
Up_i
RTL_OR
seven_seg_disp
e
n[3:0]
seg[6:0]
hex7seg
seg[6:0]
selectdisp
N[15:0]
sel[3:0]
H[3:0]
Selector
ring
adv
clk
sel[3:0]
ring_counter
clkin
btnR
btnR_IBUF_inst
slowit
clkin
greset
clk
digsel
lab4_clks
segments
ring_counter
an0_i
an0_i__0
an0_i__1
an0_i__2
RTL_INV
an[3:0]

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/27/2021 01:05:50 PM
// Design Name:
// Module Name: lab4_simulation
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module lab4_simulation();
    reg [15:0] sw = 16'b1111111111111001;
    reg btnR, btnU, btnD, btnL, btnC, clkin;
    wire [6:0] seg;
    wire [3:0] an;
    wire dp;
    wire [15:0] led;

    lab4_top UUT (.clkin(clkin), .btnR(btnR), .btnU(btnU), .btnD(btnD), .btnC(btnC),
    .btnL(btnL), .sw(sw), .seg(seg), .dp(dp), .an(an), .led(led));

    //wire [7:0] D7Seg3,D7Seg2,D7Seg1,D7Seg0; // Change the Radix of these signals
to ASCII
    //show_7segDisplay  showit (.seg(seg),.dp(dp),.an(an),
    // .D7Seg0(D7Seg0),.D7Seg1(D7Seg1),.D7Seg2(D7Seg2),.D7Seg3(D7Seg3));

    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;

    initial    // Clock process for clkin
    begin
        #OFFSET
          clkin = 1'b1;
```

```
    forever
    begin
        #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
    end
end
initial
begin
  // add your stimuli here
  // to set signal foo to value 0 use
  // foo = 1'b0;
  // to set signal foo to value 1 use
  // foo = 1'b1;
  //always advance time my multiples of 100ns
  // to advance time by 100ns use the following line
  btnU = 1'b0;
  btnR = 1'b0;
  btnL = 1'b0;
  btnC = 1'b0;
  btnD = 1'b0;
  #1000;
  btnL = 1'b1;  //Loading FFF9.
  #100;
  btnL = 1'b0;
  btnC = 1'b1;
  #100; //BtnC held for multiple cycles.
  #100;
  #100;
  #100;
  #100;
  btnC = 1'b0;
  #100; //It's now FFFC.
  btnU = 1'b1;
  #100; //FFFD
  btnU = 1'b0;
  #100;
  btnU = 1'b1;
  #100; //FFFE
  btnU = 1'b0;
  #100;
  btnU = 1'b1;
  #100; //FFFF
  btnU = 1'b0;
  #100;
  btnU = 1'b1;
  #100; //0000
  btnU = 1'b0;
```

```
    #100;
    btnD = 1'b1;
    #100; //FFFF
    #100; //BtnD held for multiple cycles.
    #100;
    btnD = 1'b0;
    #100;
    btnD = 1'b1;
    btnU = 1'b1;
    #100; //FFFF
    #100; //BtnU AND BtnD held for multiple cycles.
    #100;
    btnD = 1'b0;
    btnU = 1'b0;
    #100; //FFFF
    btnU = 1'b1;
    #100; //0000
    #100; //BtnU held for multiple cycles.
    #100;
    btnU = 1'b0;
    #100;
  end


endmodule
```