CSE100 Lab 4

1 Lab Overview

In this laboratory we were tasked to develop a counter that receives button and switch inputs to increase, decrease, load, and reset numerical counters on the Basys3 board. The goal was to have btnU and btnD be the increase and decrease signals respectively, btnR be the reset signal, btnL be the loading signal (which loads a 16 bit number indicated by the flipped switches), and btnC be a signal for increasing the count rapidly until stopped (stopping automatically at FFFC). LEDs 1 and 15 are reserved for indicating UTC and DTC signals when the count is at 0000 or FFFF.

The purpose of this lab assignment is to get acquainted with using flip-flops, experiment with the system clock, and understand the logic behind a counter, which introduces the concept of sequential logic. Several challenges were made for this lab: we could only implement it using normal logical gates and the assign function; use a ring counter with a selector for the 7-segment display and an edge detector; and develop a counter which can decrease and increase when prompted.

1.1 countUD4L

The 16-bit counter breaks down into 4 4-bit counters, passing the appropriate signal as high and the other signals as low. As inputs, the module countUD16L receives the signals Up (increase), Dw (decrease), clk (system clock), LD (load), Din (value to load). The module outputs Q (the values of the registers), UTC and DTC (the 0000 and FFFF signals). This same inputs and outputs are also passed down to the 4 countUD4L modules, which break down the 16 bits into 4 bits.

1.2 Ring Counter

A ring counter is a synchronous indicator of the position of a single high bit within a series of low bits. In other words, a 4-bit ring counter shifts in the sequence 1000, 0100, 0010, and 0001. This will be used in the 7-segment display.

1.3 Selector

The selector takes in the bit bus resulting from the ring counter and picks the appropriate signals coming from the registers of the counter. Say Q[15:0] represents the 16 bits held in the counter register. When the ring counter outputs 0001, then the selector extracts Q[3:0]; when the ring counter outputs 0010 the selector extracts Q[7:4]; and 0100 extracts Q[11:8] or 1000 extracts Q[15:12]. The selector helps setting the 7-segment as it alternates from each display giving it the result.

1.4 7-Segment Display

The seven segment display functions in the same fashion as in previous lab assignments, so it won't be explained in this report. It again functions using multiplexers for each segment A through

G. Yet, there is a difference on how the module hex7seg handles inputs and outputs. The module receives only 3 bits picked by the selector and sends them to a specific display.

1.5 Top Module

The top module is a bit complicated in design, yet it is standard for counters using a ring counter. Thankfully, the top level design is given in the lab assignment description and all of the details, parts, and connections are described in detail. The following image was provided by the lab assignment description illustrating the architecture of this sequential logic system

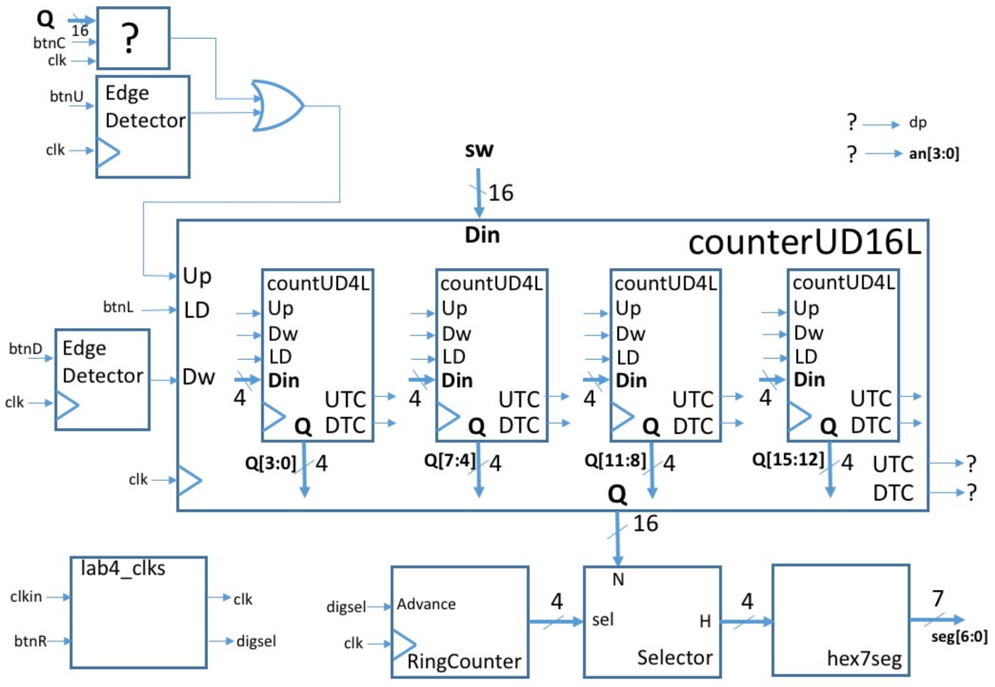


Figure 1: Source: https://classes.soe.ucsc.edu/cse100/Spring21/lab/lab4S20/lab4.html

2 Implementation and Logic

The implementation for this assignment was very straight forward. For the ring counter, multiplexer, hex7seg, and selector a truth table was created and a simple sum of products was implemented. These truth tables will be avoided in this report because it has been repetitively used on previous assignments. The truth table from the hex7seg is a bit trickier, but its explanation is also in previous lab reports.

The main complexity of the assignment relied on the counter logic. Luckily, the course book throughout section 7.11 explains in detail the logic of counters and the ring counter specifically. The following diagram's logic was implemented in Verilog

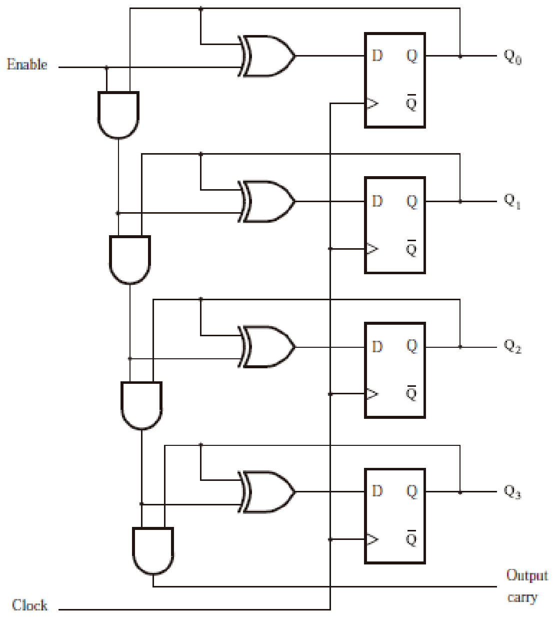


Figure 2: Brown, Vranesic. *Fundamentals of Digital Logic with Verilog Design.* Pg 410.

To know if a specific countUD4L has to raise its count then all preivous flip-flops have to be set to 1, and for a decrease all previous bits have to be set to 0. Once this connections are established the flip-flops are stated using the following line

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

where x is 0, 1, 2, and 3. The UTC and DTC signals are set individaully for each countUD4L. If all 4 DTC or UTC are set to high, then the general DTC or UTC signals are set to high. For the edge detector there are two flip-flops which hold the values of DTC and UTC. Whenever a signal is sent from the Basys3 board inputs the program checks if the command is available. The three checks are

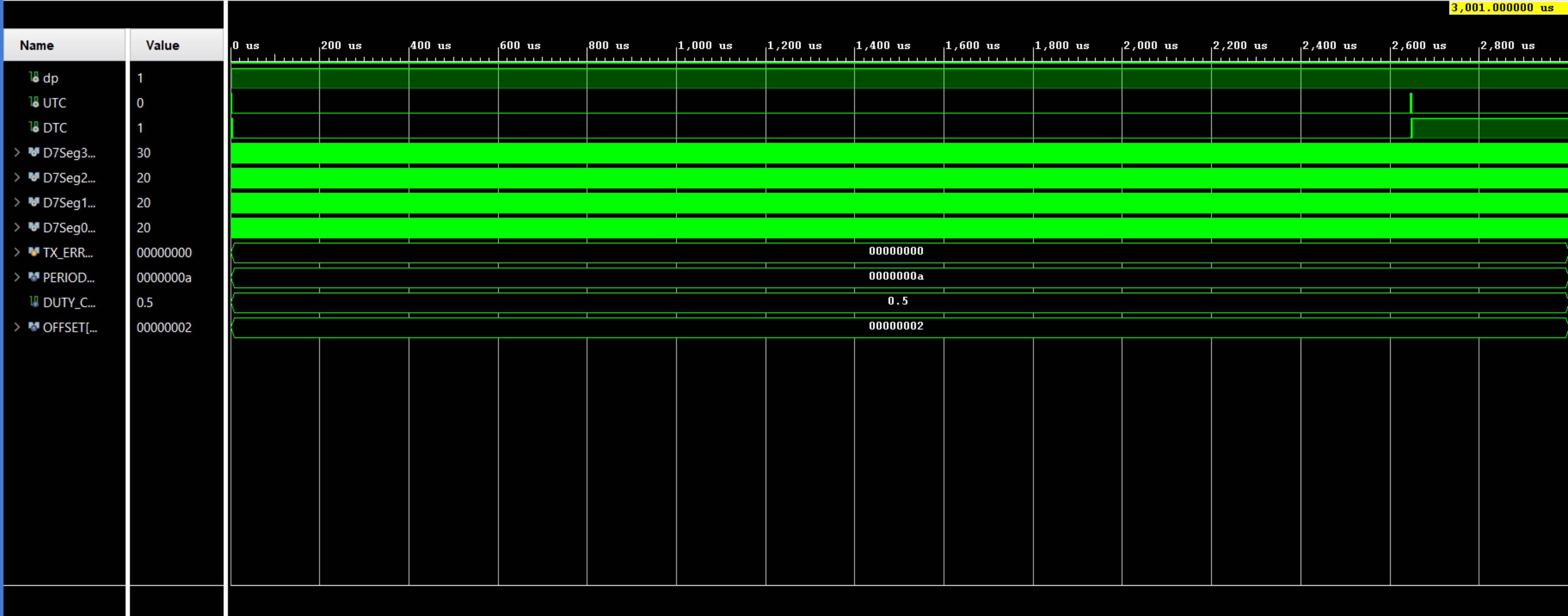
- Up signal checks if count is not at FFFF
- btnC button is held checks if count is lower than FFFC
- Dw signal checks if count is not at 0000

After this 3 checks have been performed the signals are passed to the counter, where the corresponding signal gets set to

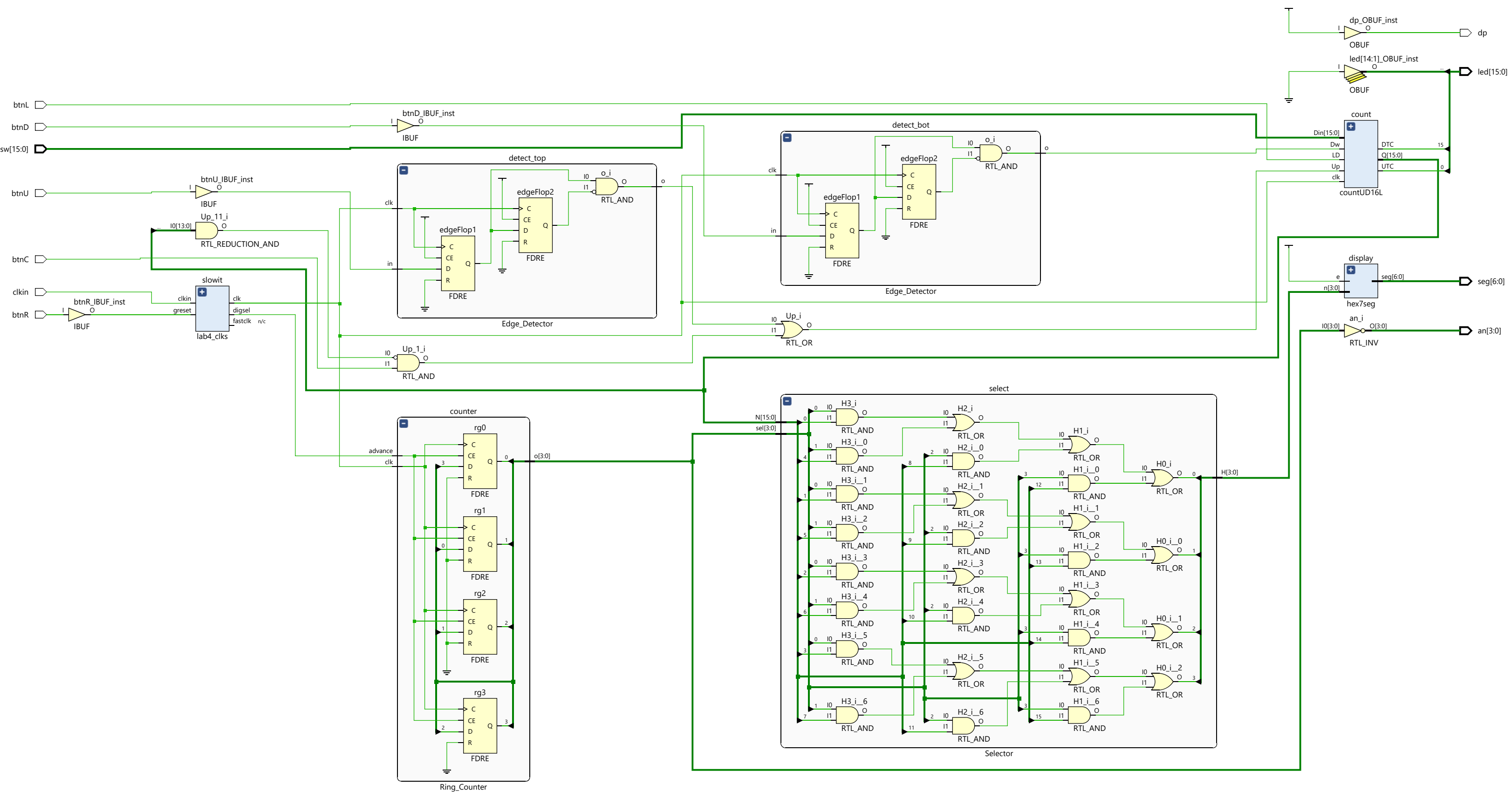
3 Conclusion and Comments

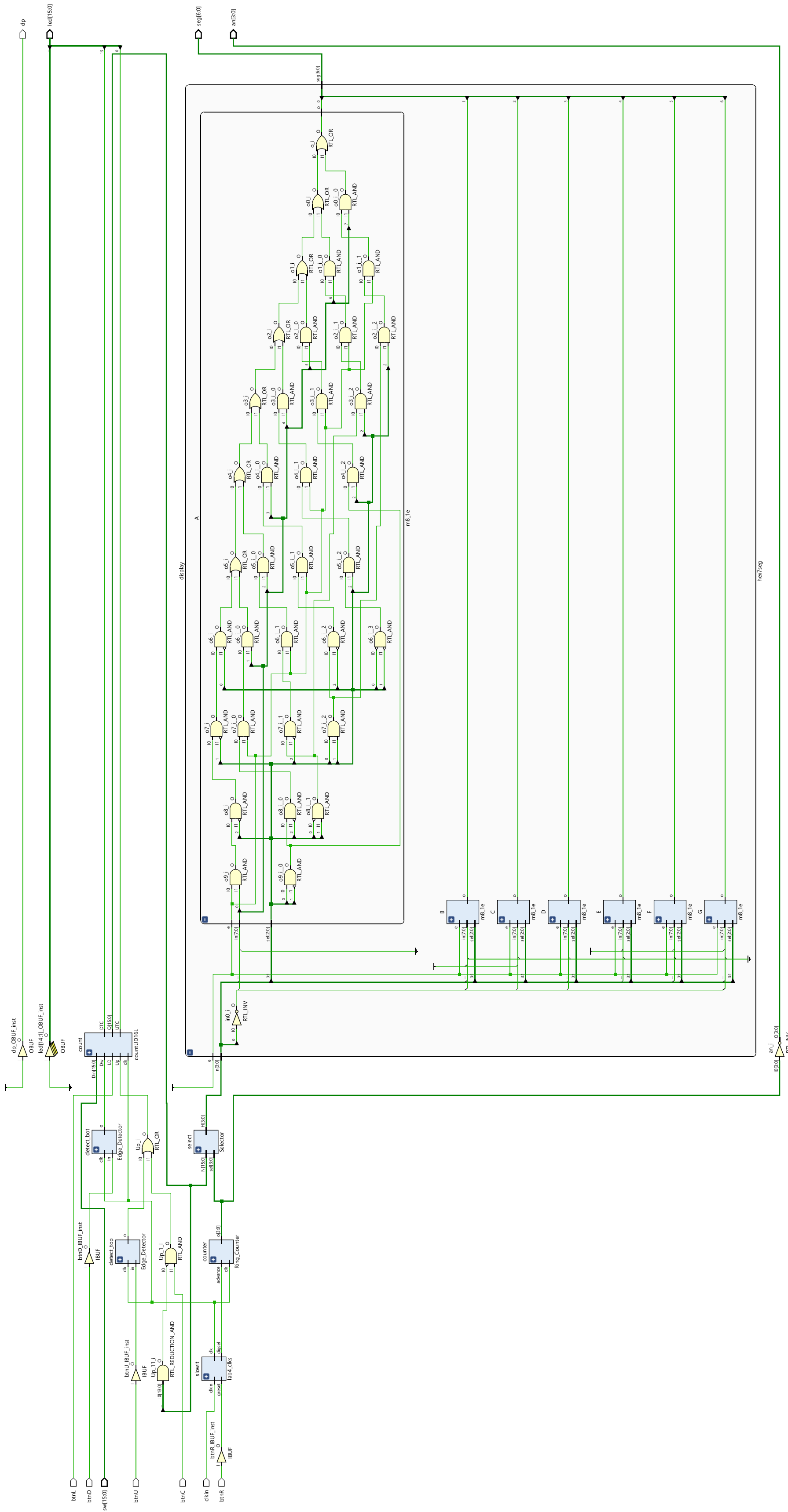
This laboratory assignments was a perfect way of introducing ourselves to sequential logic and registers. Also, knowledge about counters give us insight as to how much more complex logic functions. This assignment aside from being very straightforward it was also particularly long. Multiple roadblocks kept me from making progress due to the advancing complexity of these assignments, but the course book served as a wonderful way of making this entire assignment understandable and intuitive.

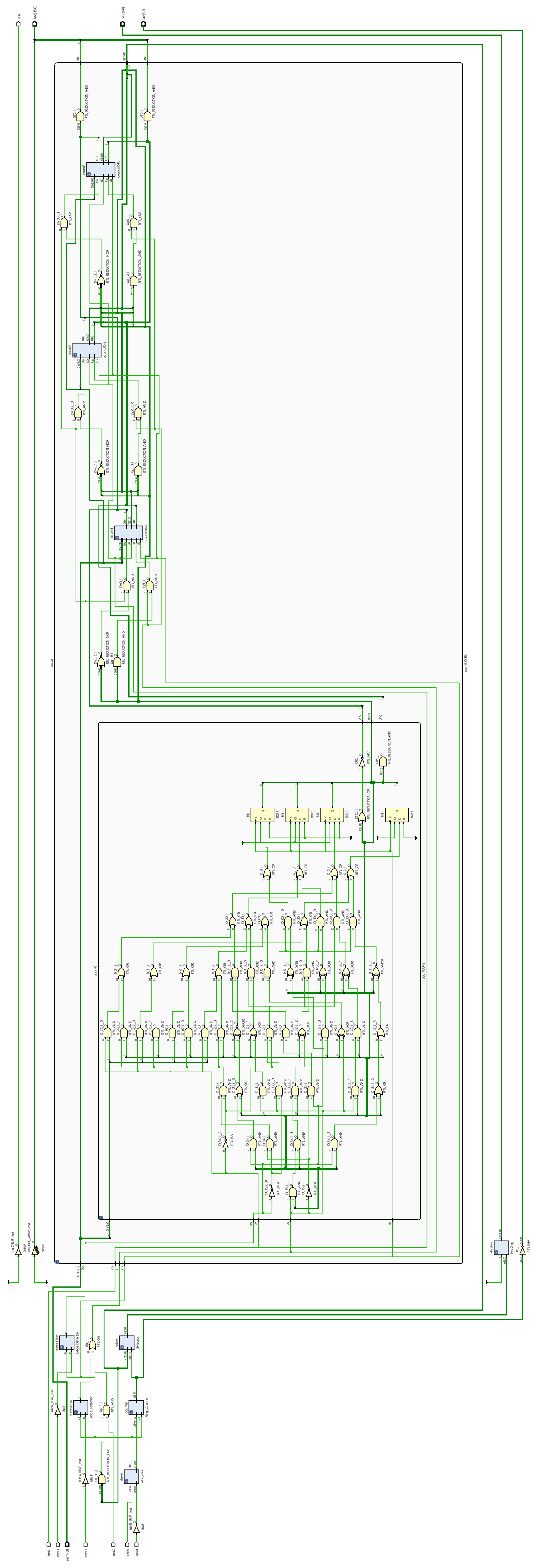
Appendix

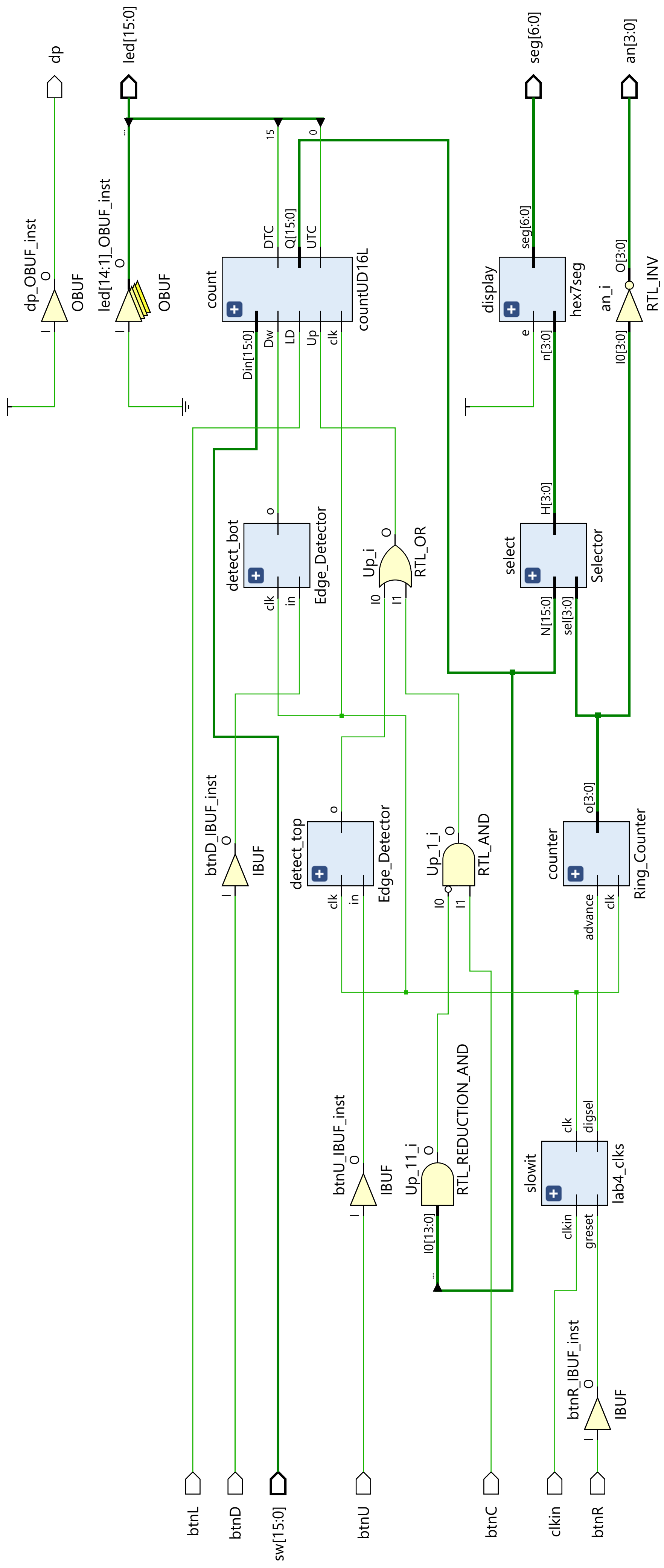


Name	Value
dp	1
UTC	0
DTC	1
D7Seg3...	30
D7Seg2...	20
D7Seg1...	20
D7Seg0...	20
TX_ERR...	00000000
PERIOD...	0000000a
DUTY_C...	0.5
OFFSET[...	00000002


btnL
btnD
sw[15:0]
btnU
btnC
clkin
btnR
btnD_IBUF_inst
btnU_IBUF_inst
btnR_IBUF_inst
IBUF
Up_11_i
I0[13:0]
RTL_REDUCTION_AND
slowit
clkin
greset
clk
digsel
fastclk
n/c
lab4_clks
detect_top
edgeFlop1
edgeFlop2
FDRE
o_i
RTL_AND
Edge_Detector
detect_bot
Up_i
RTL_OR
Up_1_i
RTL_AND
counter
advance
rg0
rg1
rg2
rg3
o[3:0]
Ring_Counter
select
N[15:0]
sel[3:0]
H3_i
H2_i
H1_i
H0_i
H[3:0]
Selector
dp_OBUF_inst
OBUF
dp
led[14:1]_OBUF_inst
led[15:0]
count
Din[15:0]
Dw
LD
Up
clk
DTC
Q[15:0]
UTC
countUD16L
display
e
n[3:0]
seg[6:0]
hex7seg
an_i
I0[3:0]
O[3:0]
RTL_INV
an[3:0]


dp
led[15:0]
seg[6:0]
an[3:0]
btnL
btnD
sw[15:0]
btnU
btnC
clkin
btnR
btnD_IBUF_inst
btnU_IBUF_inst
btnR_IBUF_inst
IBUF
dp_OBUF_inst
led[14:1]_OBUF_inst
OBUF
count
countUD16L
detect_bot
detect_top
Edge_Detector
Up_i
Up_i_i
Up_11_i
RTL_REDUCTION_AND
counter
Ring_Counter
slowit
lab4_clks
select
Selector
display
hex7seg
A
B
C
D
E
F
G
m8_1e
in0_i
an_i
RTL_INV
RTL_AND
RTL_OR





dp
led[15:0]
seg[6:0]
an[3:0]
dp_OBUF_inst
OBUF
led[14:1]_OBUF_inst
OBUF
count
countUD16L
DTC
Q[15:0]
UTC
Din[15:0]
Dw
LD
Up
clk
display
hex7seg
seg[6:0]
e
n[3:0]
an_i
RTL_INV
O[3:0]
I0[3:0]
detect_bot
Edge_Detector
Up_i
RTL_OR
select
Selector
H[3:0]
N[15:0]
sel[3:0]
btnD_IBUF_inst
IBUF
detect_top
Edge_Detector
Up_1_i
RTL_AND
counter
Ring_Counter
o[3:0]
advance
btnU_IBUF_inst
IBUF
Up_11_i
RTL_REDUCTION_AND
I0[13:0]
slowit
lab4_clks
clkin
greset
digsel
btnR_IBUF_inst
IBUF
btnL
btnD
sw[15:0]
btnU
btnC
clkin
btnR

```
module Edge_Detector(
    input in,
    input clk,
    output o
    );
    wire [1:0] w;

    FDRE #(.INIT(1'b0) ) edgeFlop1 (.C(clk), .R(1'b0), .CE(1'b1),
.D(in), .Q(w[0]));
    FDRE #(.INIT(1'b0) ) edgeFlop2 (.C(clk), .R(1'b0), .CE(1'b1),
.D(w[0]), .Q(w[1]));

    assign o = w[0] & ~w[1];

endmodule
```

```
module Edge_Detector(
    input in,
    input clk,
    output o
    );
    wire [1:0] w;

    FDRE #(.INIT(1'b0) ) edgeFlop1 (.C(clk), .R(1'b0), .CE(1'b1),
.D(in), .Q(w[0]));
    FDRE #(.INIT(1'b0) ) edgeFlop2 (.C(clk), .R(1'b0), .CE(1'b1),
.D(w[0]), .Q(w[1]));

    assign o = w[0] & ~w[1];

endmodule
```

```
`timescale 1ns / 1ps

module Ring_Counter(
    input clk,
    input advance,
    output [3:0]o
    );
    FDRE #(.INIT(1'b1) ) rg0 (.C(clk), .R(0), .CE(advance),
.D(o[3]), .Q(o[0]));
    FDRE #(.INIT(1'b0) ) rg1 (.C(clk), .R(0), .CE(advance),
.D(o[0]), .Q(o[1]));
    FDRE #(.INIT(1'b0) ) rg2 (.C(clk), .R(0), .CE(advance),
.D(o[1]), .Q(o[2]));
    FDRE #(.INIT(1'b0) ) rg3 (.C(clk), .R(0), .CE(advance),
.D(o[2]), .Q(o[3]));
endmodule
```

```
`timescale 1ns / 1ps

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

```
`timescale 1ns / 1ps
module countUD16L(
    input Up,
    input Dw,
    input clk,
    input LD,
    input [15:0] Din,
    output [15:0] Q,
    output UTC,
    output DTC
    );

    wire [3:0] Up_, Dw_, UTC_, DTC_;

    assign Dw_[0] = ~| Q[3:0];
    assign Dw_[1] = ~| Q[7:0];
    assign Dw_[2] = ~| Q[11:0];
    assign Dw_[3] = ~| Q[15:0];

    assign Up_[0] = & Q[3:0];
    assign Up_[1] = & Q[7:0];
    assign Up_[2] = & Q[11:0];
    assign Up_[3] = & Q[15:0];

    countUD4L count0(.clk(clk), .Up(Up),        .Dw(Dw),
.LD(LD), .Din(Din[3:0]),   .Q(Q[3:0]),   .UTC(UTC_[0]),
.DTC(DTC_[0]));
    countUD4L count1(.clk(clk), .Up(Up&Up_[0]), .Dw(Dw&Dw_[0]),
.LD(LD), .Din(Din[7:4]),   .Q(Q[7:4]),   .UTC(UTC_[1]),
.DTC(DTC_[1]));
    countUD4L count2(.clk(clk), .Up(Up&Up_[1]), .Dw(Dw&Dw_[1]),
.LD(LD), .Din(Din[11:8]),  .Q(Q[11:8]),  .UTC(UTC_[2]),
.DTC(DTC_[2]));
    countUD4L count3(.clk(clk), .Up(Up&Up_[2]), .Dw(Dw&Dw_[2]),
.LD(LD), .Din(Din[15:12]), .Q(Q[15:12]), .UTC(UTC_[3]),
.DTC(DTC_[3]));
```

```
    assign UTC = & UTC_[3:0];
    assign DTC = & DTC_[3:0];
endmodule
```

```
`timescale 1ns / 1ps
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] D;

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

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

    assign UTC =   &Q[3:0];
    assign DTC = ~(|Q[3:0]);
endmodule
```

```
`timescale 1ns / 1ps
module hex7seg(
    input [3:0] n,
    output [6:0] seg,
    input e
    );
    m8_1e A (.e(e), .sel(n[3:1]), .in({1'b0,  n[0],  n[0], 1'b0,
1'b0, ~n[0], 1'b0,  n[0]}), .o(seg[0]));
    m8_1e B (.e(e), .sel(n[3:1]), .in({1'b1, ~n[0],  n[0], 1'b0,
~n[0],  n[0], 1'b0,  1'b0}), .o(seg[1]));
    m8_1e C (.e(e), .sel(n[3:1]), .in({1'b1, ~n[0],  1'b0, 1'b0,
1'b0,  1'b0, ~n[0], 1'b0}), .o(seg[2]));
    m8_1e D (.e(e), .sel(n[3:1]), .in({n[0],  1'b0, ~n[0], 1'b0,
n[0], ~n[0], 1'b0,  n[0]}), .o(seg[3]));
    m8_1e E (.e(e), .sel(n[3:1]), .in({1'b0,  1'b0,  1'b0, n[0],
n[0],  1'b1, n[0],  n[0]}), .o(seg[4]));
    m8_1e F (.e(e), .sel(n[3:1]), .in({1'b0,  n[0],  1'b0, 1'b0,
n[0],  1'b0, 1'b1,  n[0]}), .o(seg[5]));
    m8_1e G (.e(e), .sel(n[3:1]), .in({1'b0, ~n[0],  1'b0, 1'b0,
n[0],  1'b0, 1'b0,  1'b1}), .o(seg[6]));
endmodule
```

```
`timescale 1ns / 1ps

module m8_1e(
    input [7:0] in,
    input [2:0] sel,
    input e,
    output o
    );
    assign o = (e & in[0] & ~sel[2] & ~sel[1] & ~sel[0] |
         sel[0] & ~sel[1] & ~sel[2] & e & in[1] |
        ~sel[0] &  sel[1] & ~sel[2] & e & in[2] |
         sel[0] &  sel[1] & ~sel[2] & e & in[3] |
        ~sel[0] & ~sel[1] &  sel[2] & e & in[4] |
         sel[0] & ~sel[1] &  sel[2] & e & in[5] |
        ~sel[0] &  sel[1] &  sel[2] & e & in[6] |
         sel[0] &  sel[1] &  sel[2] & e & in[7]
        );
endmodule
```