ECE 3300L

Lab Report #5

Group A

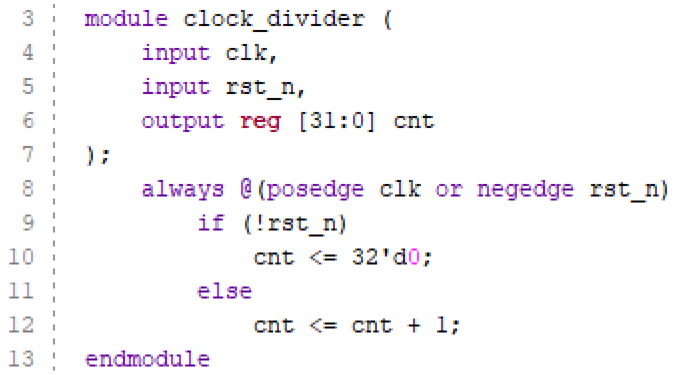
Edwin Estrada (#015897050)

Michelle Lau (#016027427)

July 20, 2025

Design

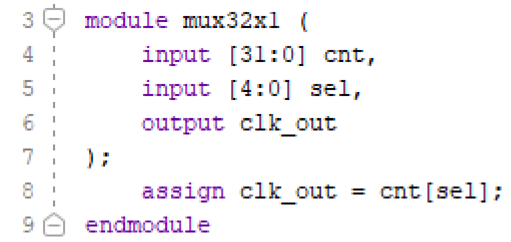
Clock Divider



```verilog
module clock_divider (
    input clk,
    input rst_n,
    output reg [31:0] cnt
);
    always @(posedge clk or negedge rst_n)
        if (!rst_n)
            cnt <= 32'd0;
        else
            cnt <= cnt + 1;
endmodule
```

Our program has two clocks. This clock handles increment/decrements once per cycle. To create a clock with 32 levels of speed, a 32 bit register to hold the count is created. If that's the input for a 32x1 mux, selected by 5 switches, the speed of the clock can be manipulated depending on the bit selected. If the LSB of cnt is selected, the increment/decrement operation is done 100 million times per second (since the board has 100Mhz base clock). If the MSB of cnt is selected, the increment/decrement operation is done once every ~43 seconds.

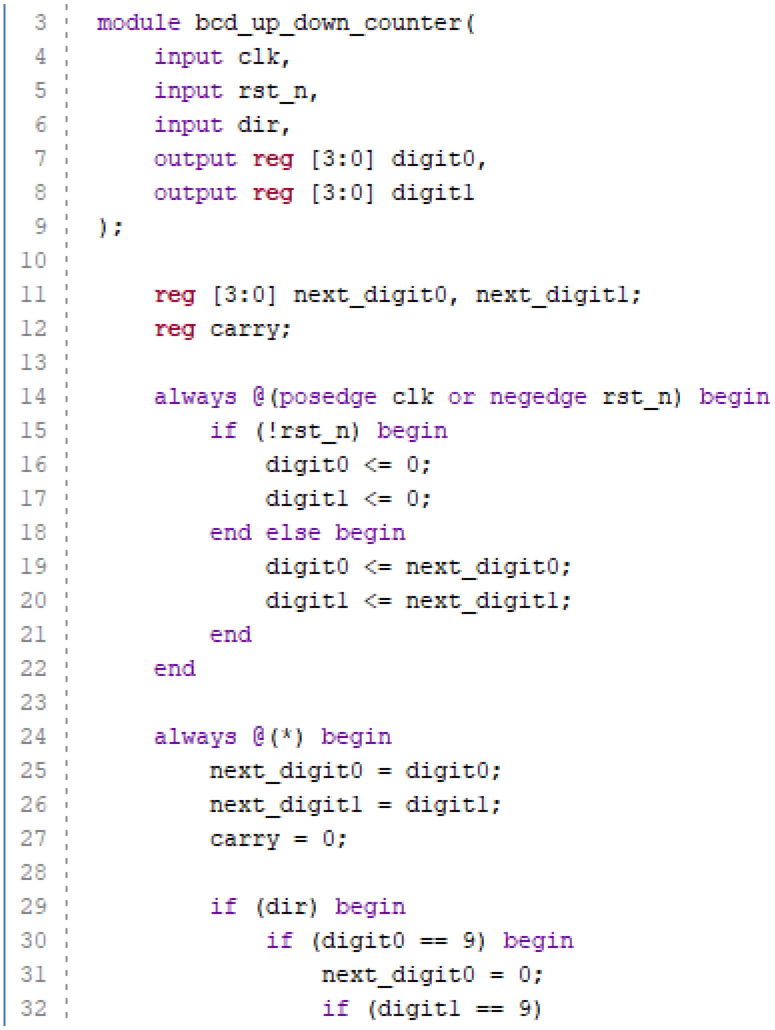
32x1 Mux



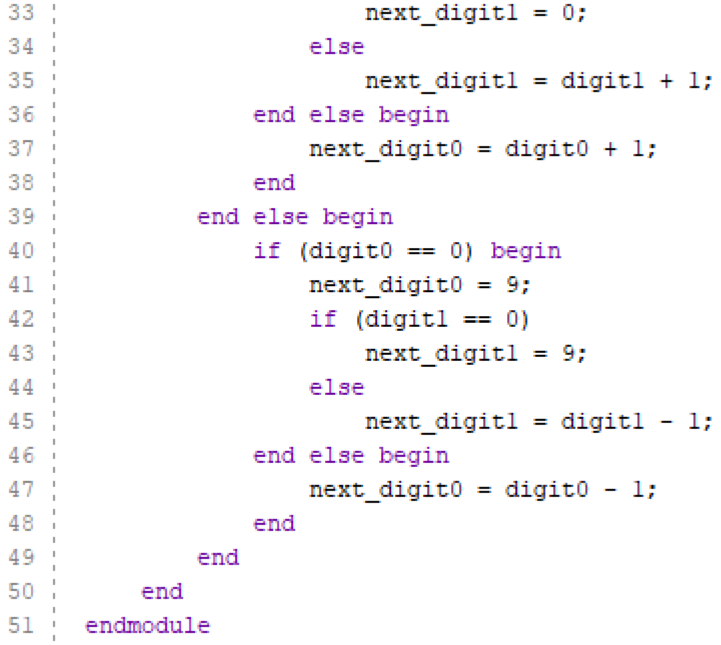
```verilog
module mux32x1 (
    input [31:0] cnt,
    input [4:0] sel,
    output clk_out
);
    assign clk_out = cnt[sel];
endmodule
```

As explained in the module above, the 32x1 mux is used to control the speed of the clock dedicated to incrementing/decrementing the counter.

BCD Up/Down Counter



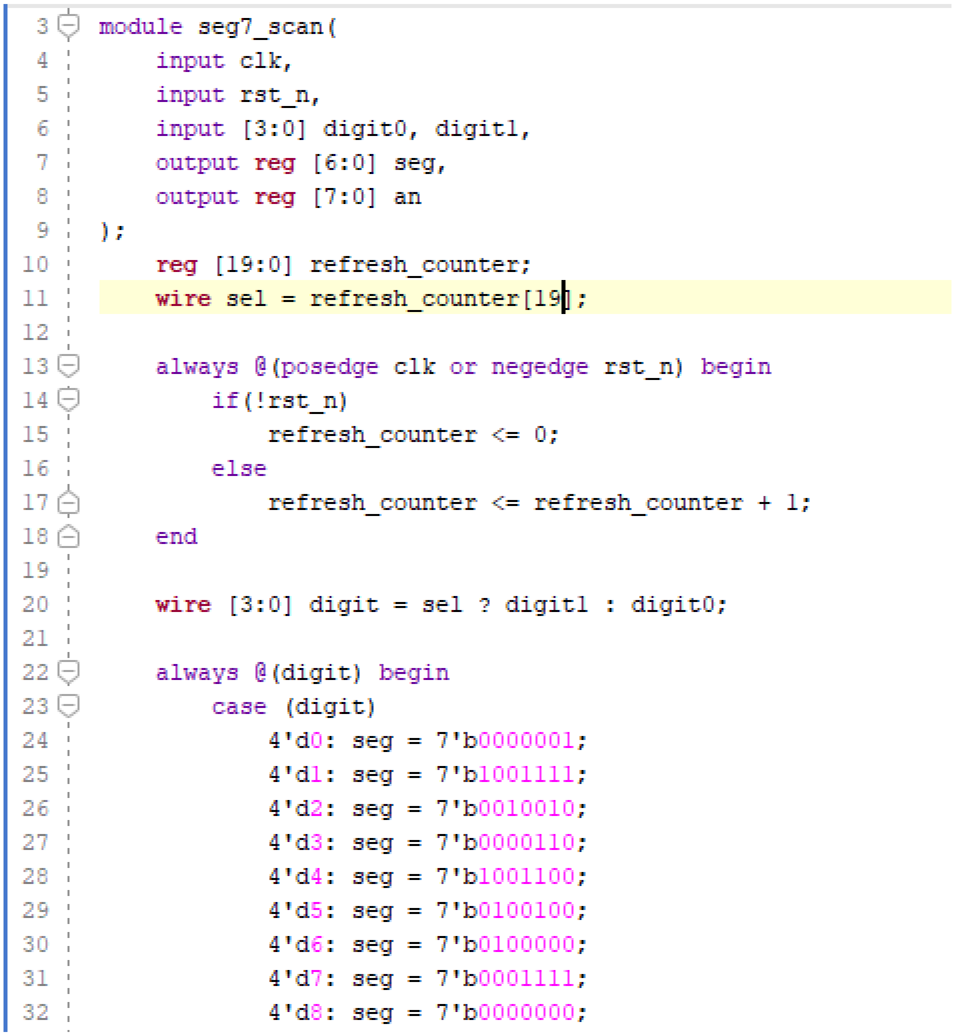
```verilog
module bcd_up_down_counter(
    input clk,
    input rst_n,
    input dir,
    output reg [3:0] digit0,
    output reg [3:0] digit1
);

    reg [3:0] next_digit0, next_digit1;
    reg carry;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            digit0 <= 0;
            digit1 <= 0;
        end else begin
            digit0 <= next_digit0;
            digit1 <= next_digit1;
        end
    end

    always @(*) begin
        next_digit0 = digit0;
        next_digit1 = digit1;
        carry = 0;

        if (dir) begin
            if (digit0 == 9) begin
                next_digit0 = 0;
                if (digit1 == 9)
```



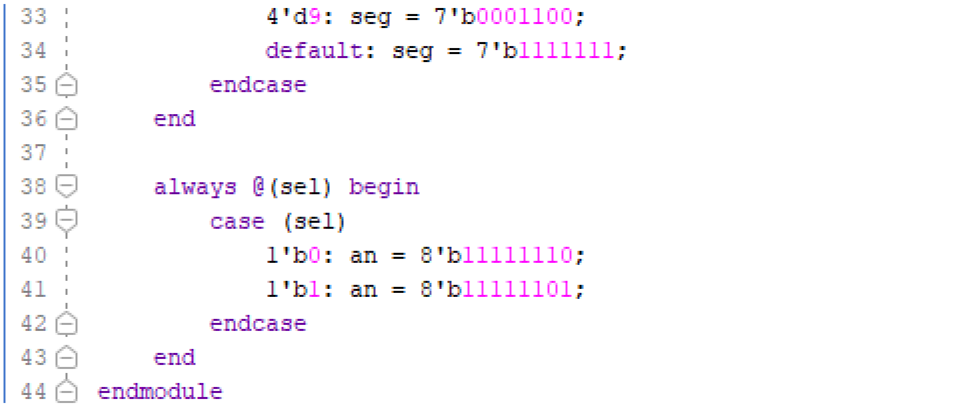
```verilog
                    next_digit1 = 0;
                else
                    next_digit1 = digit1 + 1;
            end else begin
                next_digit0 = digit0 + 1;
            end
        end else begin
            if (digit0 == 0) begin
                next_digit0 = 9;
                if (digit1 == 0)
                    next_digit1 = 9;
                else
                    next_digit1 = digit1 - 1;
            end else begin
                next_digit0 = digit0 - 1;
            end
        end
    end
endmodule
```

This module handles the next number to display when the clock mentioned above is positive edge triggered. If the counter is incrementing, check if the ones digit is a 9. If it is, set the ones digit to 0 and check if the tens digit is 9. If it's not, increment the tens digit. Otherwise, set the tens digit to 0. If the counter is decrementing, check if the ones digit is 0. If the ones digit is 0, set the ones digit to 9 and check if the tens digit is 0. If it is, set the tens digit to 9. Otherwise, decrement the tens digit.

7 Segment Display Driver



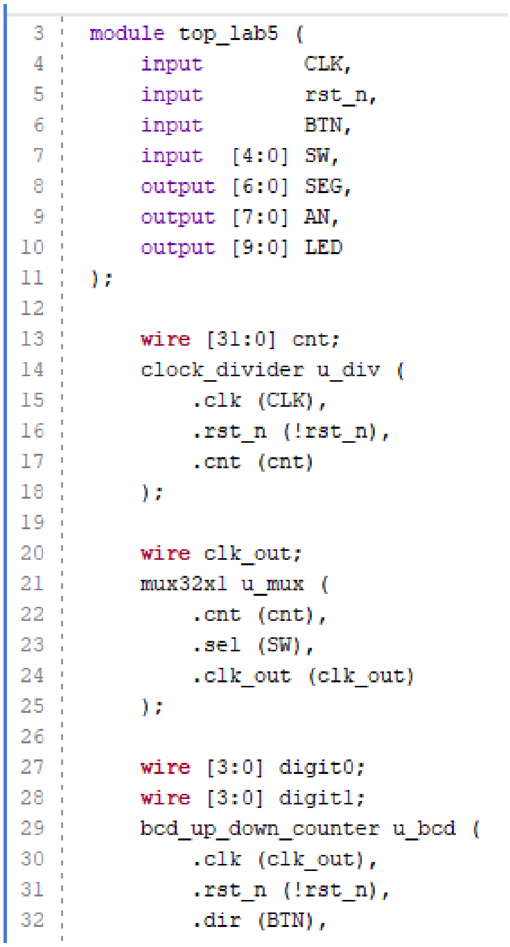
```
module seg7_scan(
    input clk,
    input rst_n,
    input [3:0] digit0, digit1,
    output reg [6:0] seg,
    output reg [7:0] an
);
    reg [19:0] refresh_counter;
    wire sel = refresh_counter[19];

    always @(posedge clk or negedge rst_n) begin
        if(!rst_n)
            refresh_counter <= 0;
        else
            refresh_counter <= refresh_counter + 1;
    end

    wire [3:0] digit = sel ? digit1 : digit0;

    always @(digit) begin
        case (digit)
            4'd0: seg = 7'b0000001;
            4'd1: seg = 7'b1001111;
            4'd2: seg = 7'b0010010;
            4'd3: seg = 7'b0000110;
            4'd4: seg = 7'b1001100;
            4'd5: seg = 7'b0100100;
            4'd6: seg = 7'b0100000;
            4'd7: seg = 7'b0001111;
            4'd8: seg = 7'b0000000;
```



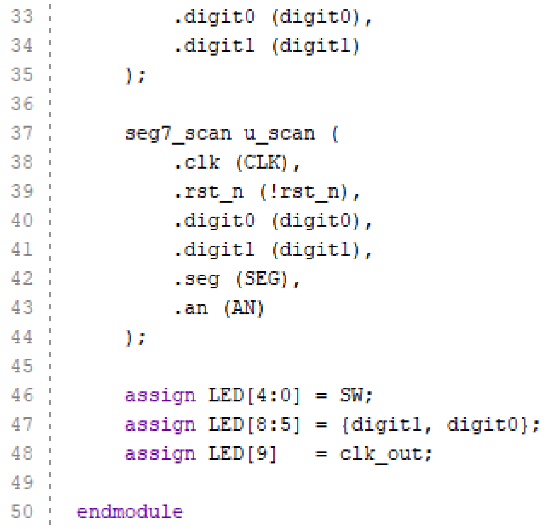
```
            4'd9: seg = 7'b0001100;
            default: seg = 7'b1111111;
        endcase
    end

    always @(sel) begin
        case (sel)
            1'b0: an = 8'b11111110;
            1'b1: an = 8'b11111101;
        endcase
    end
endmodule
```

The seven seg driver includes the second clock. This clock handles which digit on the seven seg display to turn on, alternating between two digits. If the driver alternates between the two digits fast enough, an optical illusion occurs where it looks like the display is displaying two different numbers at the same time. In addition to selecting the digit on the display to turn on, the sel register also decides the number to display on its respective digit.

High Level Implementation



```
module top_lab5 (
    input        CLK,
    input        rst_n,
    input        BTN,
    input  [4:0] SW,
    output [6:0] SEG,
    output [7:0] AN,
    output [9:0] LED
);

    wire [31:0] cnt;
    clock_divider u_div (
        .clk (CLK),
        .rst_n (!rst_n),
        .cnt (cnt)
    );

    wire clk_out;
    mux32x1 u_mux (
        .cnt (cnt),
        .sel (SW),
        .clk_out (clk_out)
    );

    wire [3:0] digit0;
    wire [3:0] digit1;
    bcd_up_down_counter u_bcd (
        .clk (clk_out),
        .rst_n (!rst_n),
        .dir (BTN),
```

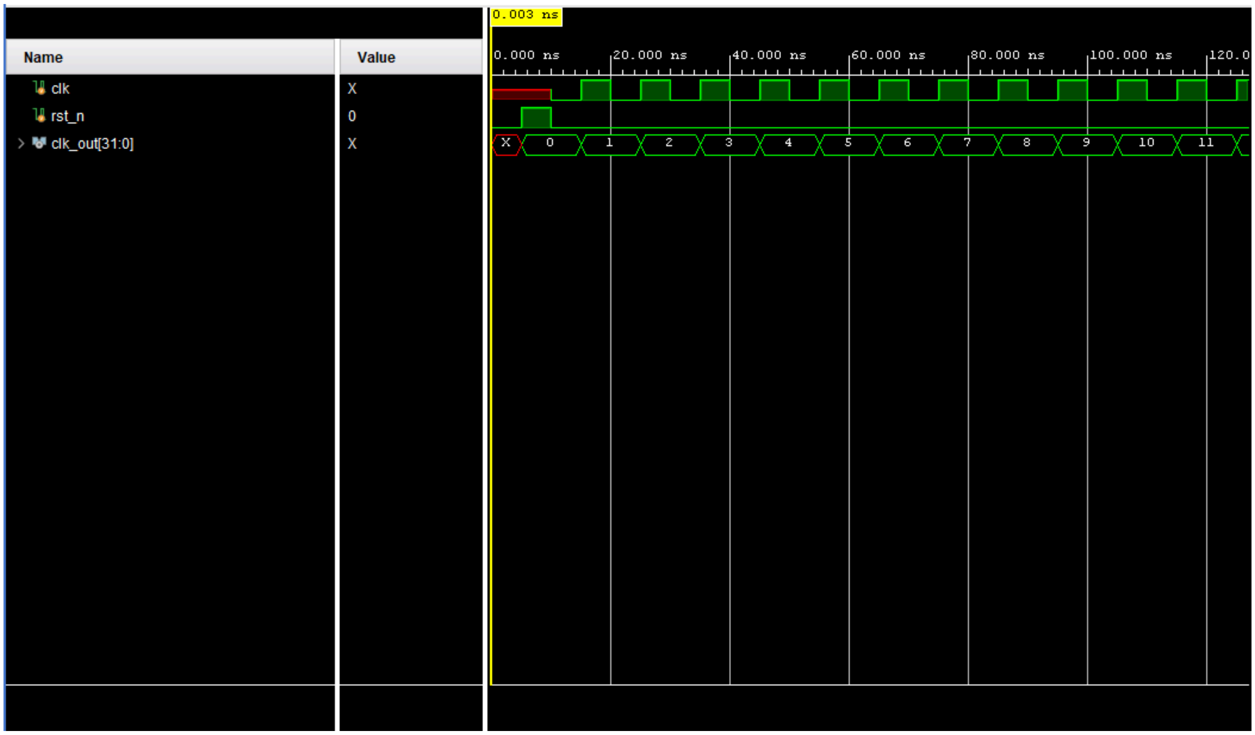


```
        .digit0 (digit0),
        .digit1 (digit1)
    );

    seg7_scan u_scan (
        .clk (CLK),
        .rst_n (!rst_n),
        .digit0 (digit0),
        .digit1 (digit1),
        .seg (SEG),
        .an (AN)
    );

    assign LED[4:0] = SW;
    assign LED[8:5] = {digit1, digit0};
    assign LED[9]   = clk_out;

endmodule
```

Putting everything together. The base clock on the board goes into the clock divider module, which goes into the 32x1 mux to select the speed of the clock outputted from the divider. This new clock is then input to the BCD increment/decrement counter, and the digit output from that counter as well as the second clock in our project are sent to the 7 seg display driver to return the signals required to send to our anodes/cathodes of the display to output the counter increment/decrementing depending on what mode we choose.

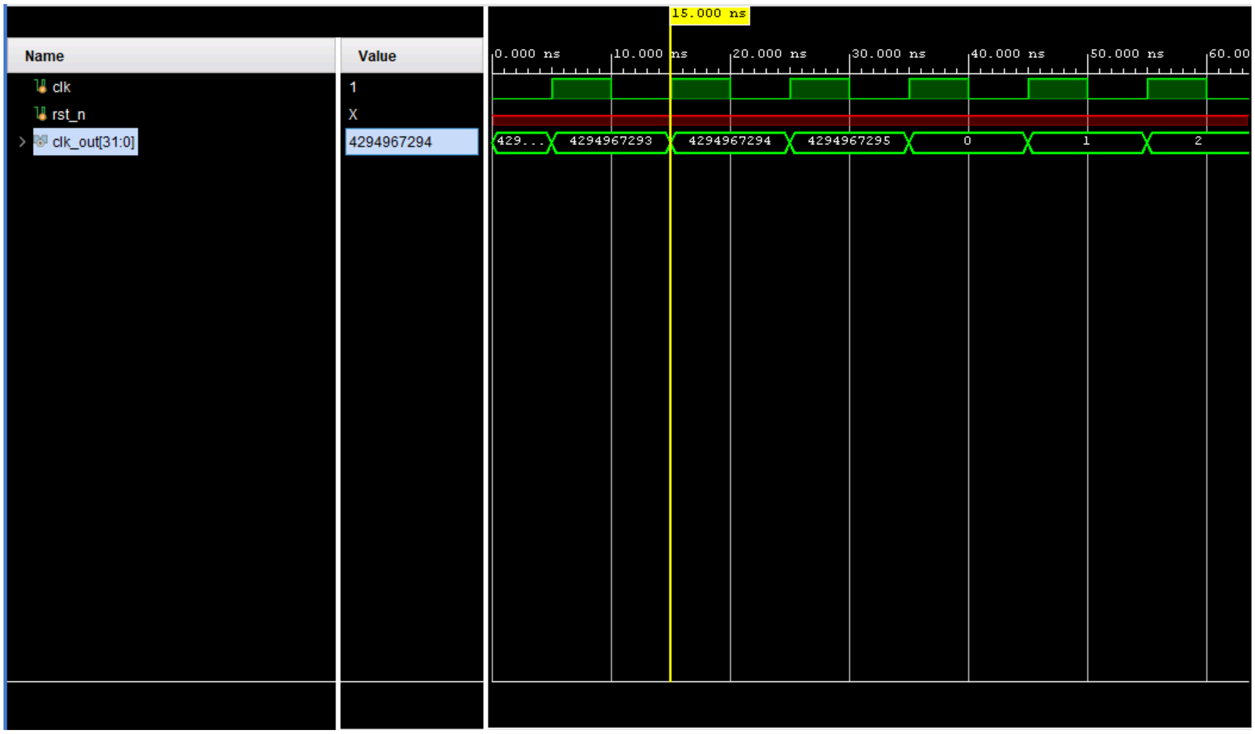
Simulation Waveform

Clock Divider



Name	Value
clk	X
rst_n	0
clk_out[31:0]	X

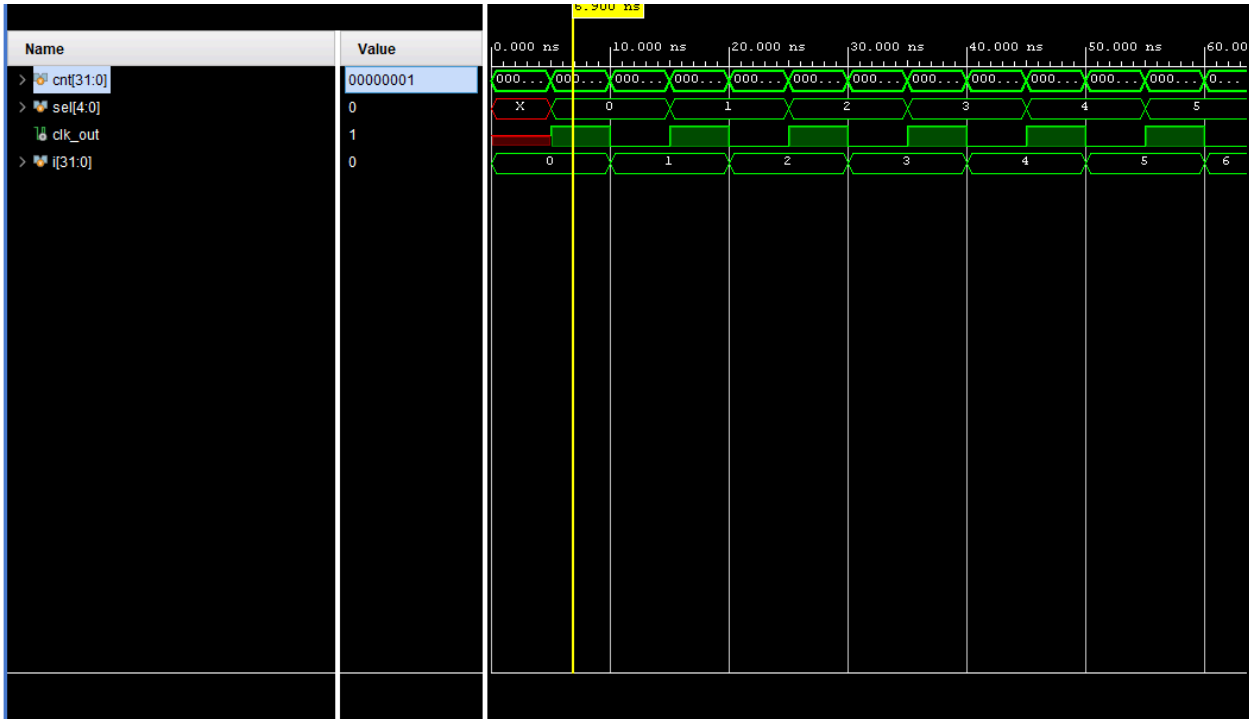
Upon starting the program, the counter starts at 0



Name	Value
clk	1
rst_n	X
clk_out[31:0]	4294967294

Once the counter reaches the value of (2^32) - 1, or 4,294,967,295, the counter is automatically reset to 0 to count all over again.

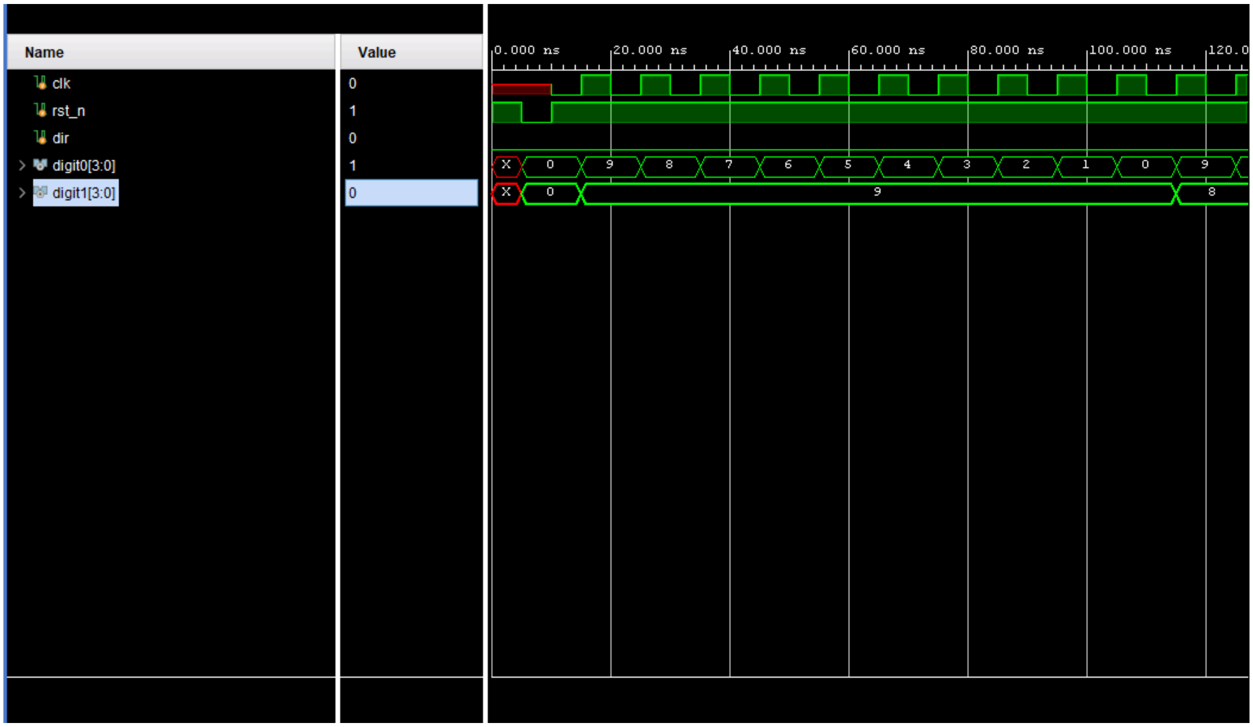
32x1 Mux



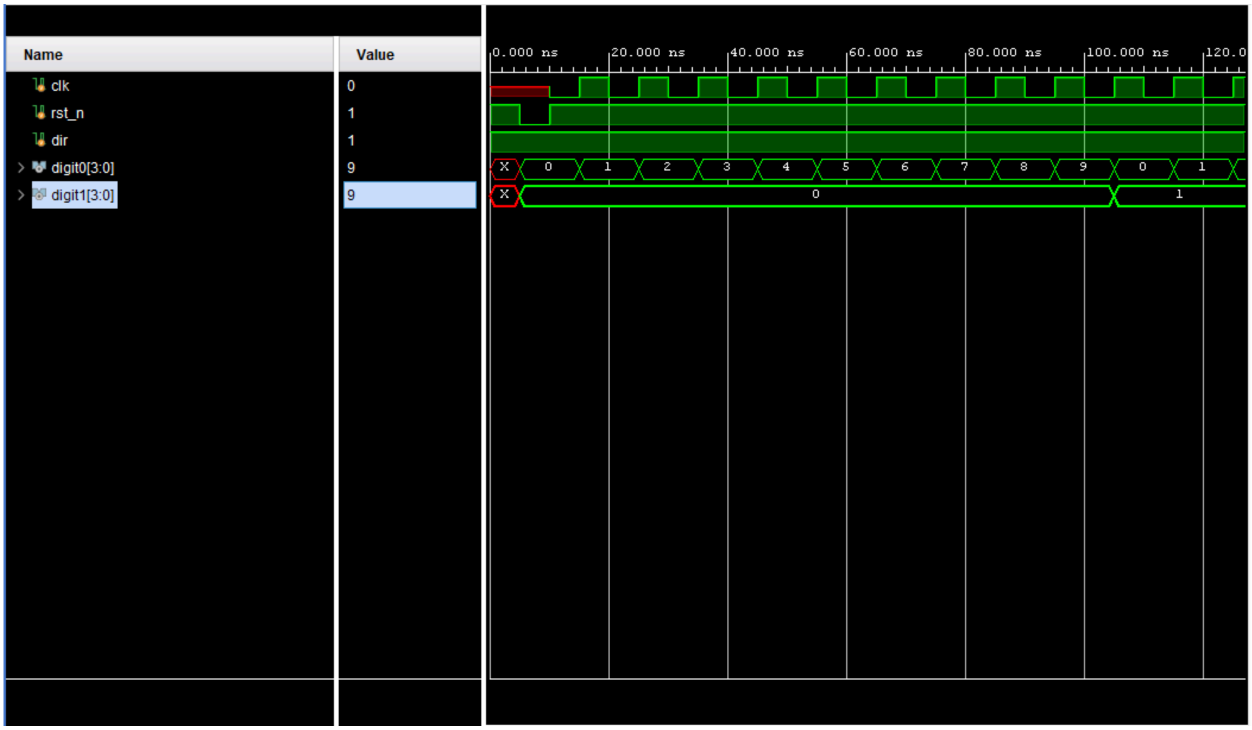
Name	Value
cnt[31:0]	00000001
sel[4:0]	0
clk_out	1
i[31:0]	0

The following process I'll explain was done for every input, but to reduce redundancy, I will only explain selecting the LSB of cnt. To begin, I put the LSB of cnt on high and setting sel to 00001, selecting the LSB. This results in the clk_out being high. Then I put the LSB of cnt low. With the LSB of cnt selected, this results in clk_out being low.

bcd_up_down_counter



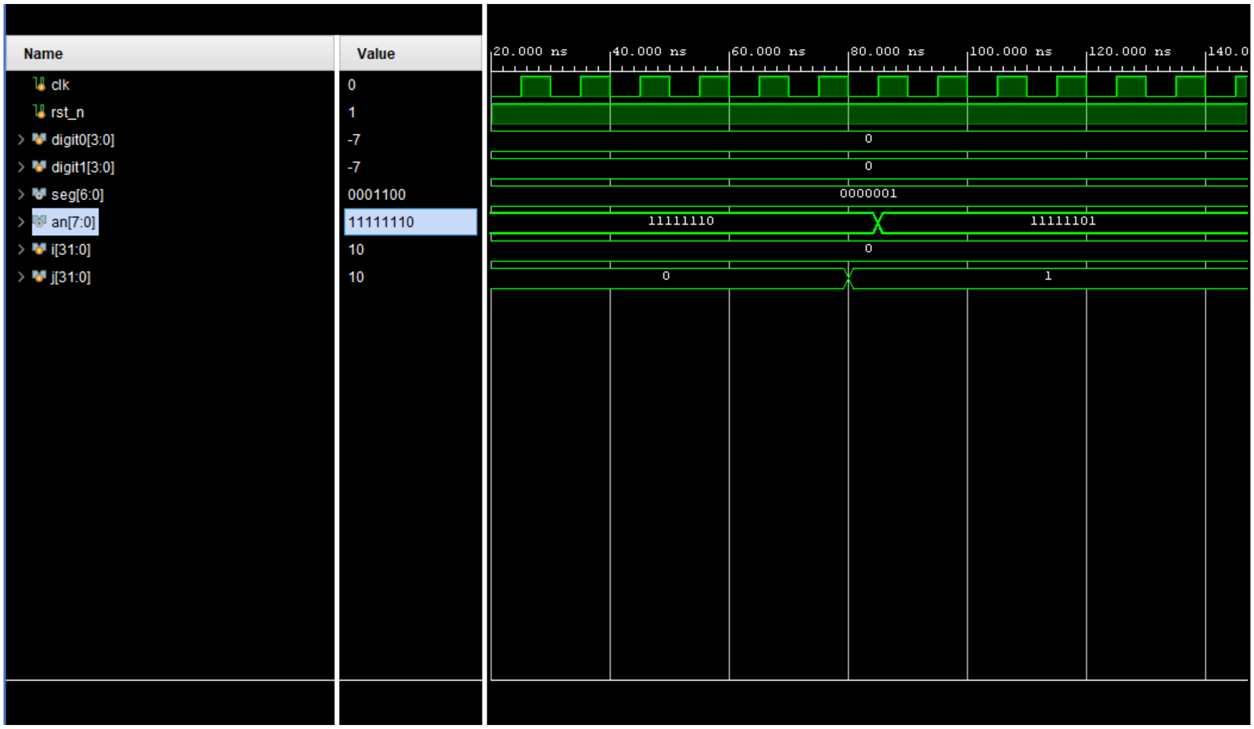
NOTE: digit1 = tens place, digit0 = ones place
When dir = 0, the counter is decrementing.



Name	Value
clk	0
rst_n	1
dir	1
digit0[3:0]	9
digit1[3:0]	9

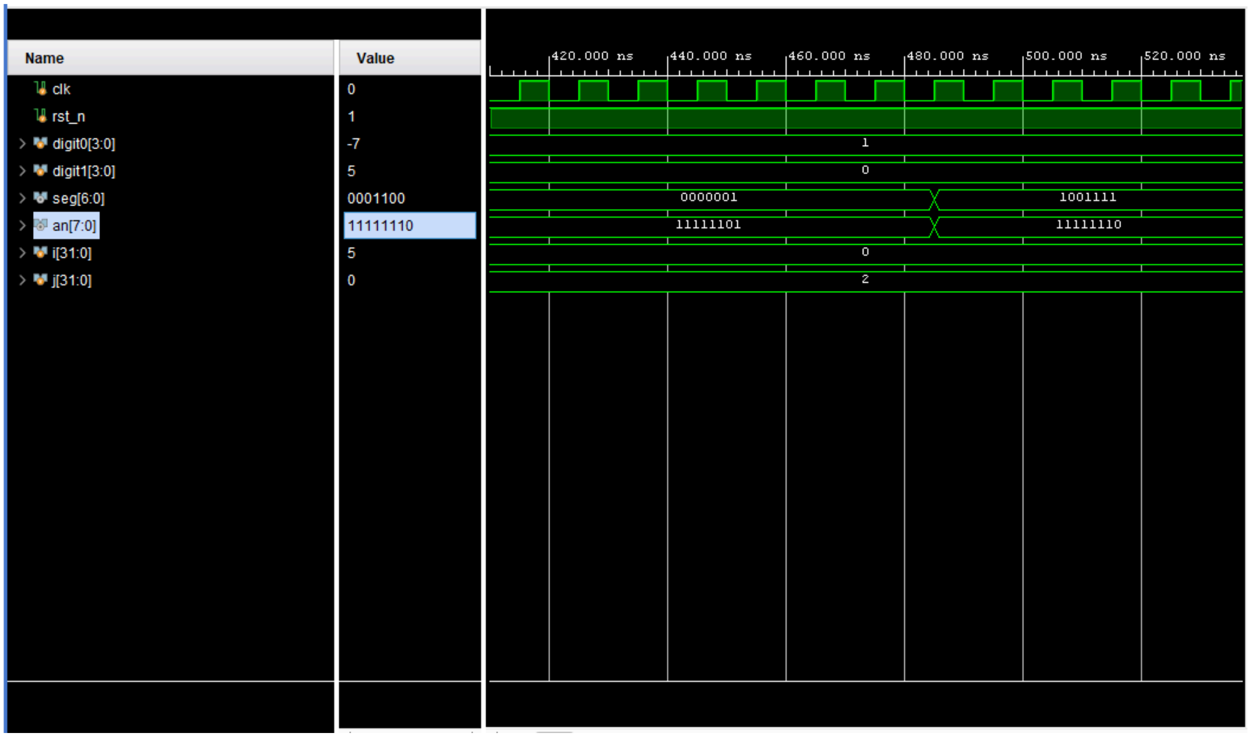
When dir = 1, the counter is incrementing.

<u>seg7_scan</u>



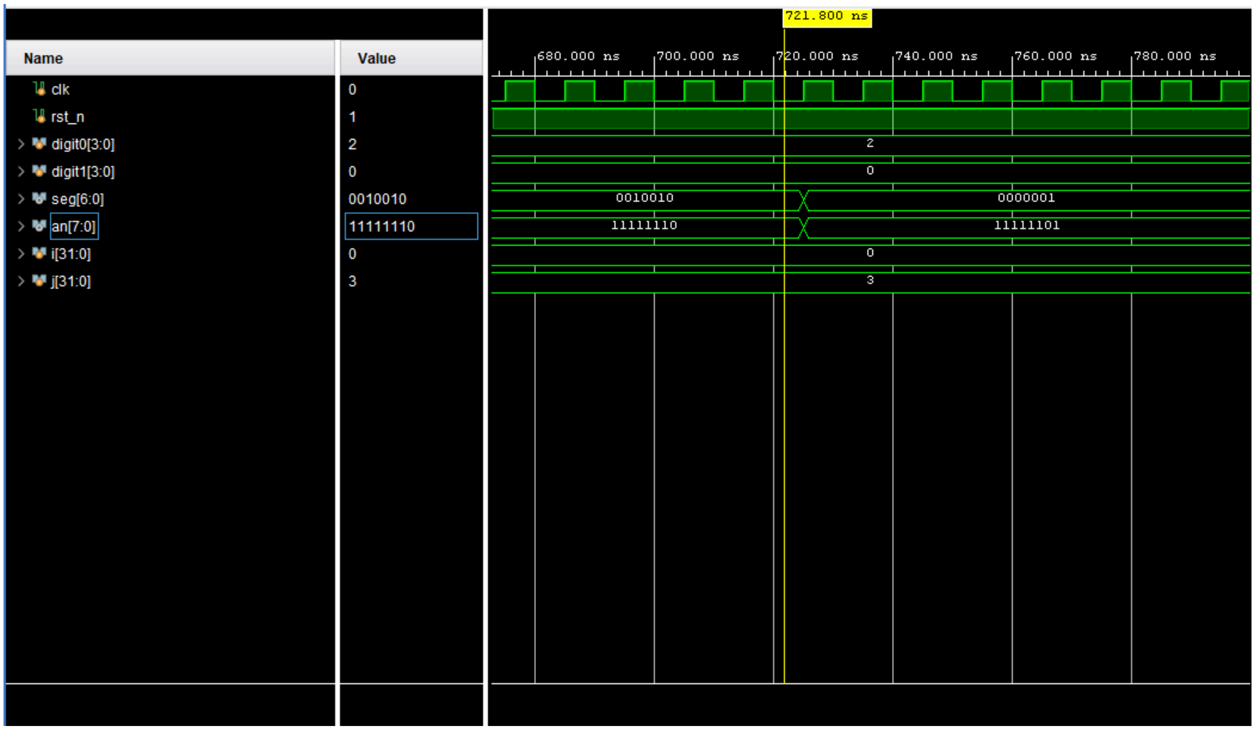
Name	Value
clk	0
rst_n	1
digit0[3:0]	-7
digit1[3:0]	-7
seg[6:0]	0001100
an[7:0]	11111110
i[31:0]	10
j[31:0]	10

When the rightmost digit is selected by AN and digit0 = 0, seg returns 000001 (the number 0).
When the leftmost digit is selected by AN and digit1 = 0, seg returns 000001 (the number 0).
“00” is displayed.



Name	Value
clk	0
rst_n	1
digit0[3:0]	-7
digit1[3:0]	5
seg[6:0]	0001100
an[7:0]	11111110
i[31:0]	5
j[31:0]	0

“01” is displayed.



Name	Value
clk	0
rst_n	1
digit0[3:0]	2
digit1[3:0]	0
seg[6:0]	0010010
an[7:0]	11111110
i[31:0]	0
j[31:0]	3

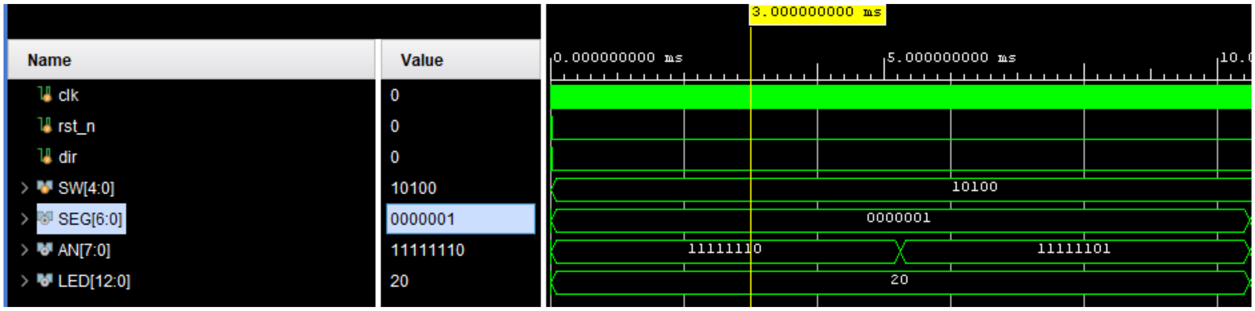
“02” is displayed.

Top Level

To simulate, I will show the counter decrementing 5 times and incrementing 5 times

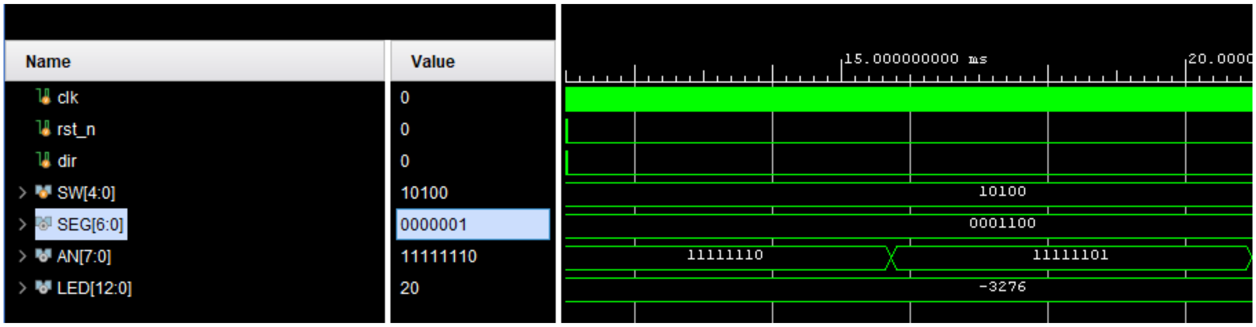
Decrementing

Dir signal is low.



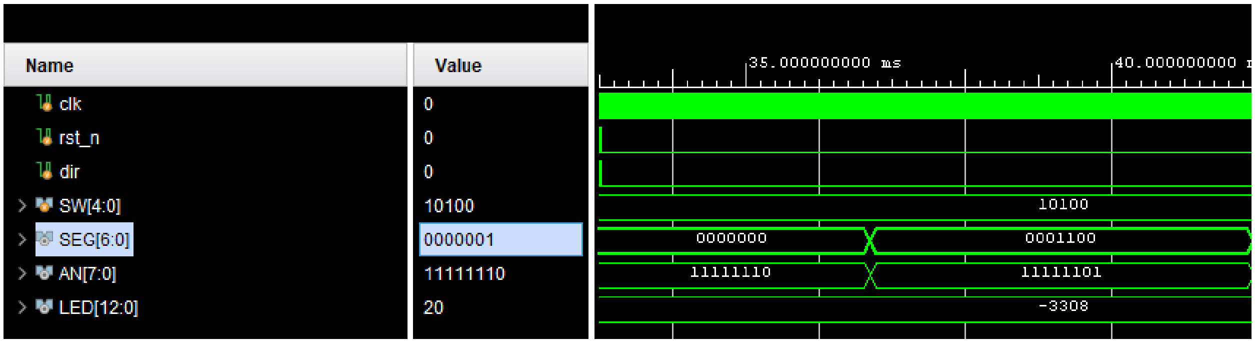
Name	Value
clk	0
rst_n	0
dir	0
SW[4:0]	10100
SEG[6:0]	0000001
AN[7:0]	11111110
LED[12:0]	20

The counter begins at “00”



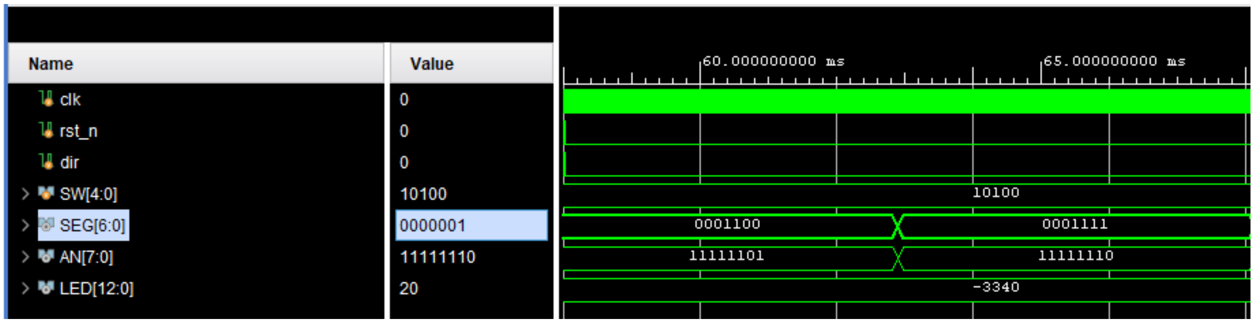
Name	Value
clk	0
rst_n	0
dir	0
SW[4:0]	10100
SEG[6:0]	0000001
AN[7:0]	11111110
LED[12:0]	20

Counter decrements, wrapping to “99”



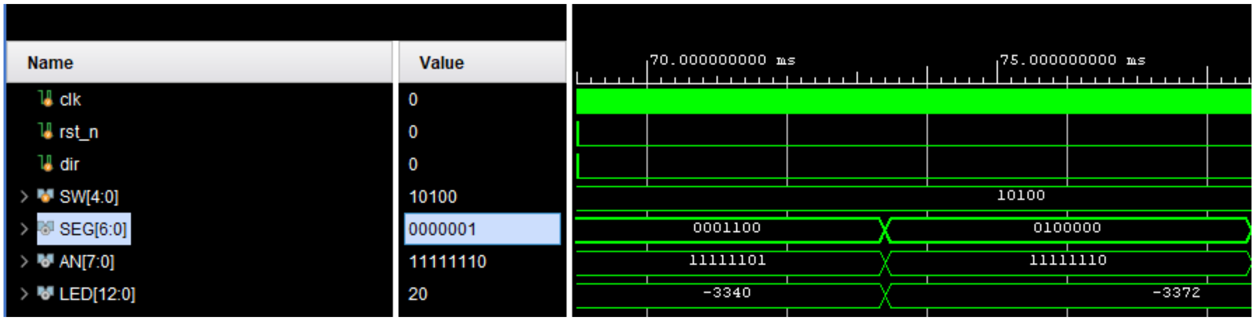
Name	Value
clk	0
rst_n	0
dir	0
SW[4:0]	10100
SEG[6:0]	0000001
AN[7:0]	11111110
LED[12:0]	20

Counter decrements. Right digit displays “8”, left digit displays “9”, resulting in 97.



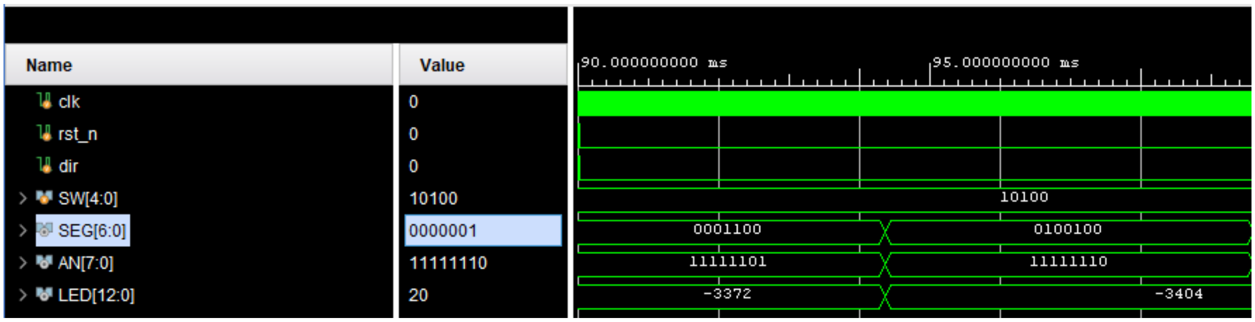
Name	Value
clk	0
rst_n	0
dir	0
SW[4:0]	10100
SEG[6:0]	0000001
AN[7:0]	11111110
LED[12:0]	20

Counter decrements. Left digit displays “9”, right digit displays “7”, resulting in “97”.



Name	Value
clk	0
rst_n	0
dir	0
SW[4:0]	10100
SEG[6:0]	0000001
AN[7:0]	11111110
LED[12:0]	20

Counter decrements. Left digit displays “9”, right digit displays “6”, resulting in “96”.

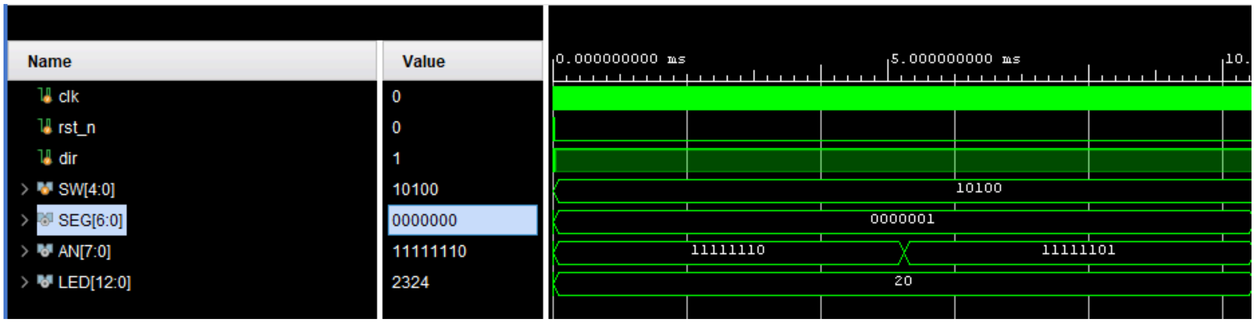


Name	Value
clk	0
rst_n	0
dir	0
SW[4:0]	10100
SEG[6:0]	0000001
AN[7:0]	11111110
LED[12:0]	20

Counter decrements. Left digit displays “9”, right digit displays “5”, resulting in “95”.

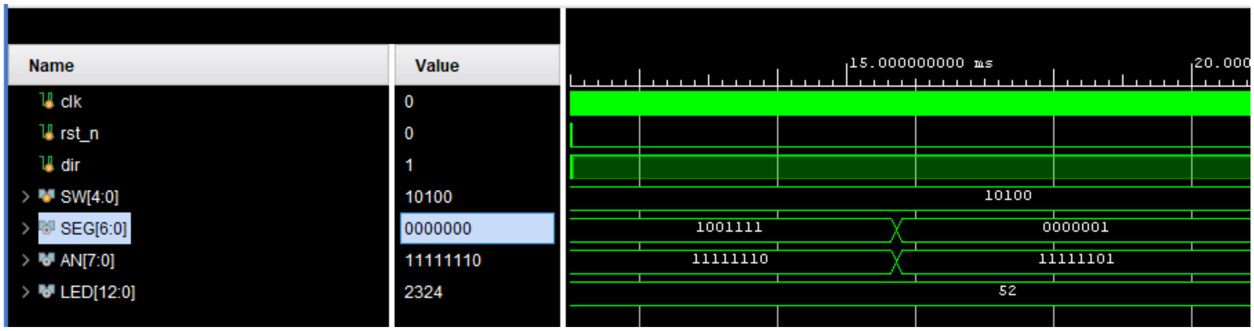
Incrementing

Dir signal is high.



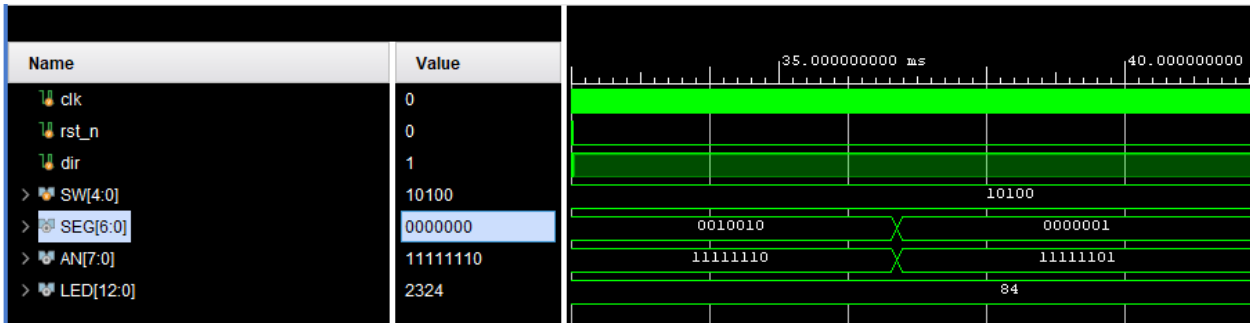
Name	Value
clk	0
rst_n	0
dir	1
SW[4:0]	10100
SEG[6:0]	0000000
AN[7:0]	11111110
LED[12:0]	2324

With the left and right digit being “0”, the counter starts at “00”.



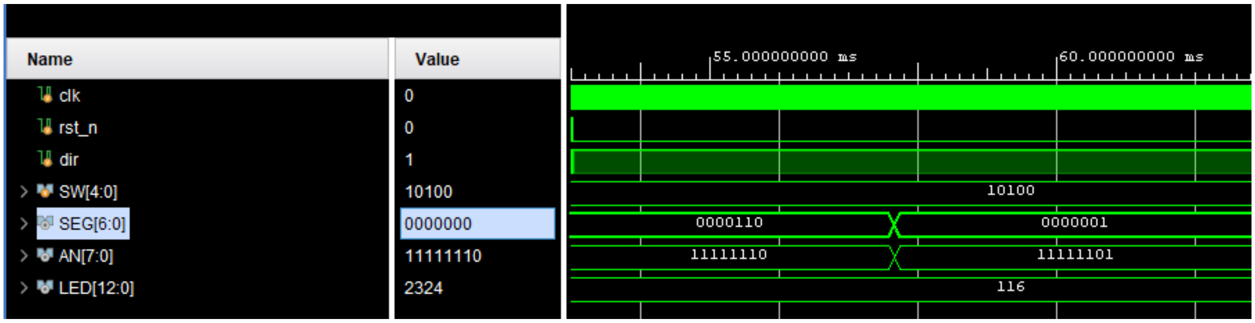
Name	Value
clk	0
rst_n	0
dir	1
SW[4:0]	10100
SEG[6:0]	0000000
AN[7:0]	11111110
LED[12:0]	2324

Counter increments. Right digit displays “1” and left digit displays “0”, resulting in “01”.



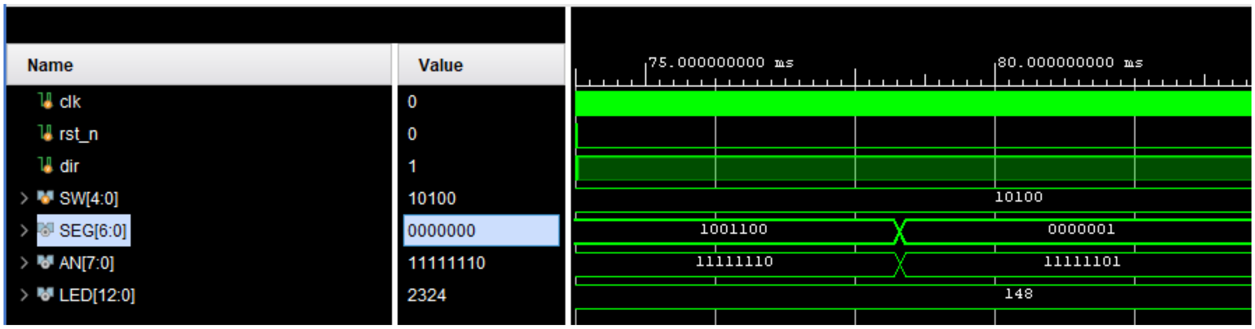
Name	Value
clk	0
rst_n	0
dir	1
SW[4:0]	10100
SEG[6:0]	0000000
AN[7:0]	11111110
LED[12:0]	2324

Counter increments. Right digit displays “2” and left digit displays “0”, resultinig in “02”.



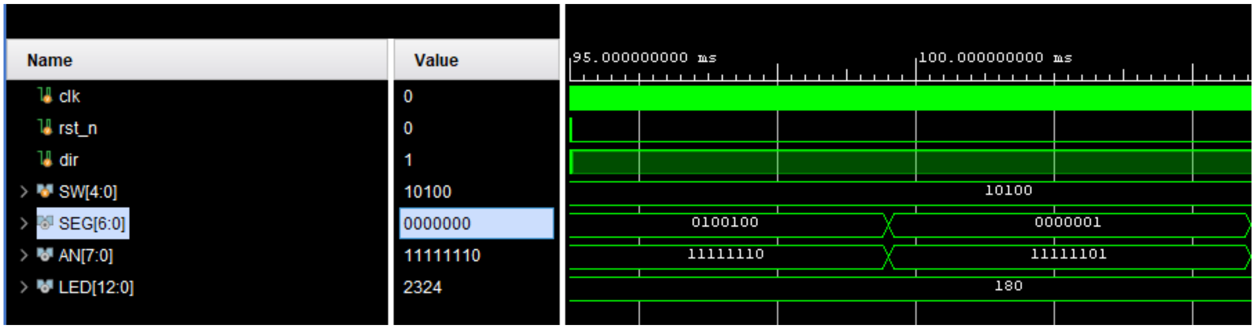
Name	Value
clk	0
rst_n	0
dir	1
SW[4:0]	10100
SEG[6:0]	0000000
AN[7:0]	11111110
LED[12:0]	2324

Counter increments. Right digit displays “3” and left digit displays “0”, resulting in “03”.



Name	Value
clk	0
rst_n	0
dir	1
SW[4:0]	10100
SEG[6:0]	0000000
AN[7:0]	11111110
LED[12:0]	2324

Counter increments. Right digit displays “4” and left digit displays “0”, resulting in “04”.

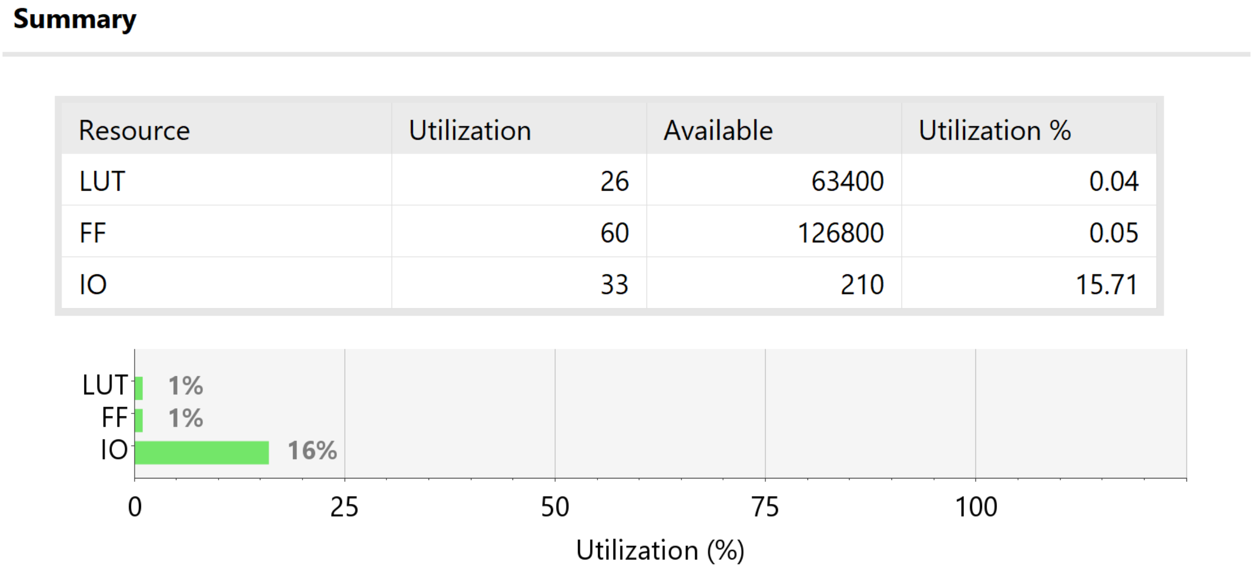


Name	Value
clk	0
rst_n	0
dir	1
SW[4:0]	10100
SEG[6:0]	0000000
AN[7:0]	11111110
LED[12:0]	2324

Counter increments. Right digit displays “5” and left digit displays “0”, resulting in “05”.

Implementation

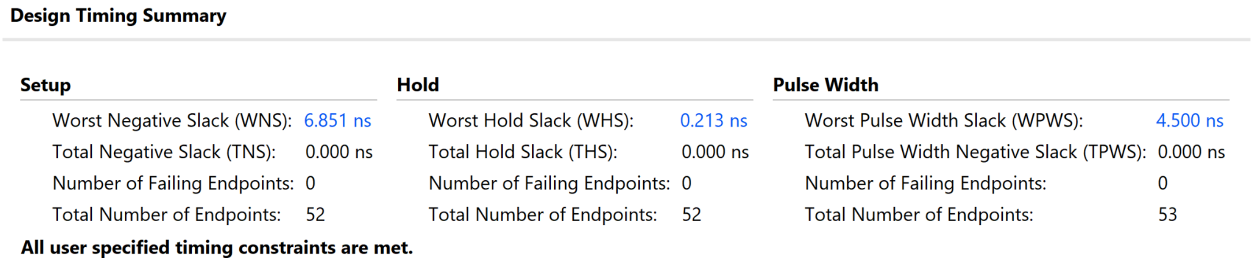
Utilization Table:



Summary

Resource	Utilization	Available	Utilization %
LUT	26	63400	0.04
FF	60	126800	0.05
IO	33	210	15.71

Timing Summary:



Design Timing Summary

Setup

Worst Negative Slack (WNS): 6.851 ns
Total Negative Slack (TNS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 52

Hold

Worst Hold Slack (WHS): 0.213 ns
Total Hold Slack (THS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 52

Pulse Width

Worst Pulse Width Slack (WPWS): 4.500 ns
Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 53

All user specified timing constraints are met.

Contribution

Michelle Lau (50%) - Implementation and board demo
Edwin Estrada (50%) - Programming, and test benching