ECE 3300L

Lab Report #6

Group A

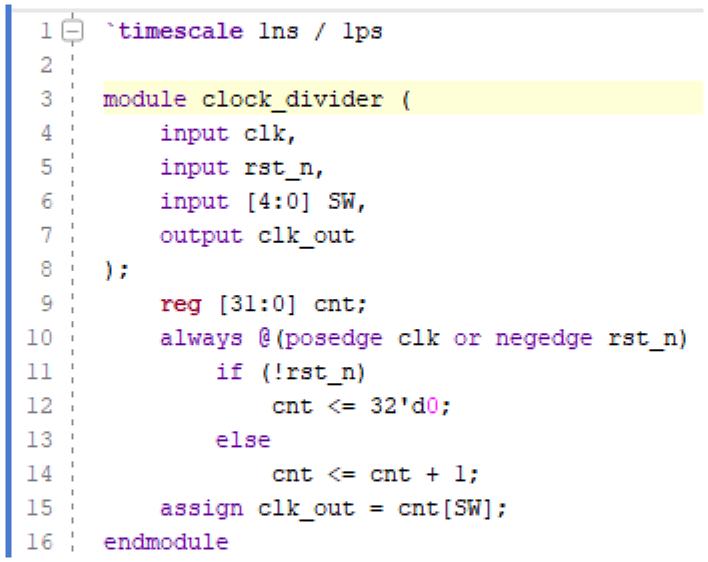
Edwin Estrada (#015897050)

Michelle Lau (#016027427)

July 28, 2025

Design

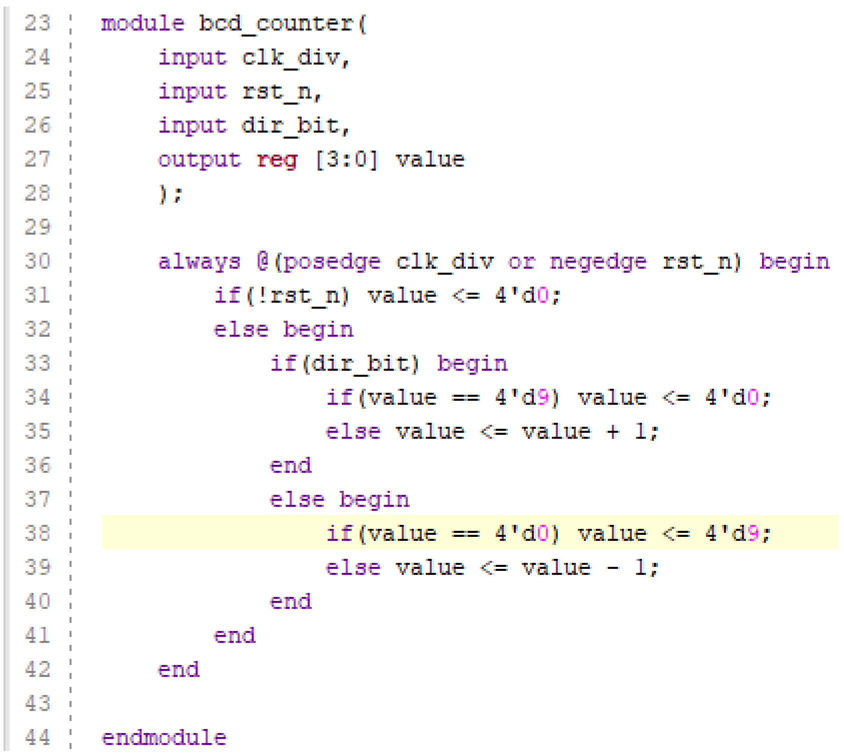
Clock Divider



```
`timescale 1ns / 1ps

module clock_divider (
    input clk,
    input rst_n,
    input [4:0] SW,
    output clk_out
);
    reg [31:0] cnt;
    always @(posedge clk or negedge rst_n)
        if (!rst_n)
            cnt <= 32'd0;
        else
            cnt <= cnt + 1;
    assign clk_out = cnt[SW];
endmodule
```

Similar to the last lab, a free-running clock is implemented where 5 switches are used to control its speed. Additionally, the 32-bit counter and 32x1 mux are combined into 1 module to enhance readability. With a 100Mhz clock from the board, Selecting bit 0 results in a 100Mhz clock and selecting bit 31 results in a clock that's positive edge triggered every ~43 seconds.

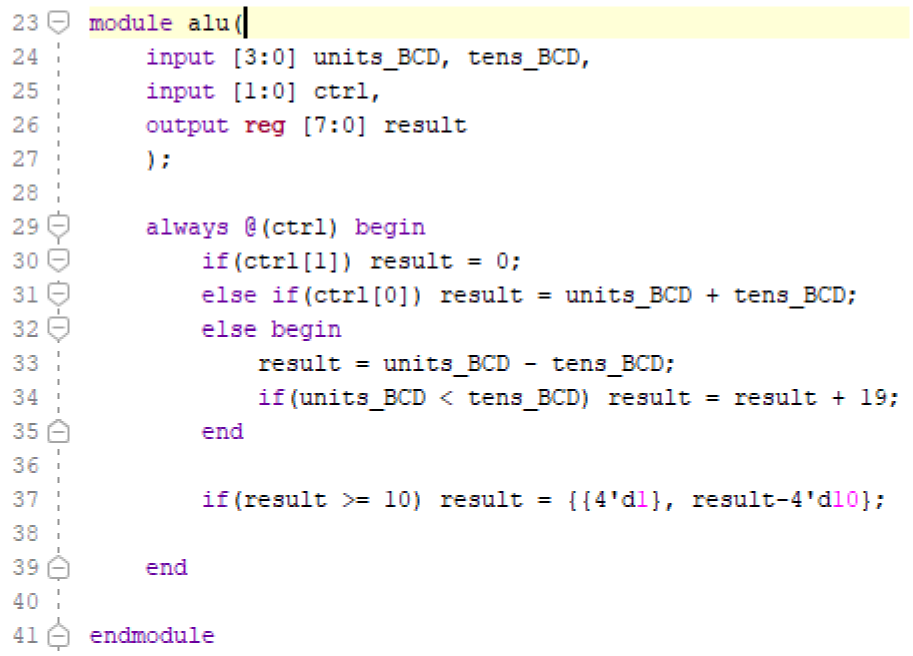
BCD Counter



```verilog
module bcd_counter(
    input clk_div,
    input rst_n,
    input dir_bit,
    output reg [3:0] value
    );

    always @(posedge clk_div or negedge rst_n) begin
        if(!rst_n) value <= 4'd0;
        else begin
            if(dir_bit) begin
                if(value == 4'd9) value <= 4'd0;
                else value <= value + 1;
            end
            else begin
                if(value == 4'd0) value <= 4'd9;
                else value <= value - 1;
            end
        end
    end

endmodule
```

For this project, we have two counters that inc/decremented independently but synchronously. Thanks to modularization, we can create a blueprint for such a counter and build it twice for our purposes. Since the two counters can only count BCD digits, they wrap after passing 0 and 9.

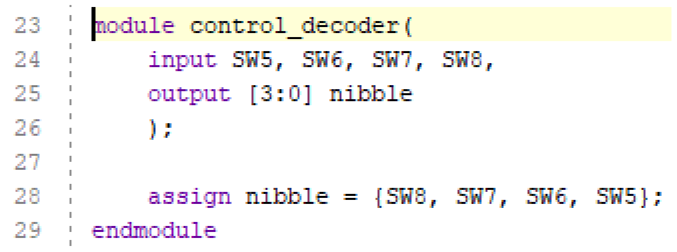
ALU



```verilog
module alu(
    input [3:0] units_BCD, tens_BCD,
    input [1:0] ctrl,
    output reg [7:0] result
    );

    always @(ctrl) begin
        if(ctrl[1]) result = 0;
        else if(ctrl[0]) result = units_BCD + tens_BCD;
        else begin
            result = units_BCD - tens_BCD;
            if(units_BCD < tens_BCD) result = result + 19;
        end

        if(result >= 10) result = {{4'd1}, result-4'd10};

    end

endmodule
```

The ALU either adds the BCD in the ones place and tens place together, or subtracts the digit in the tens place with the digit in the ones place. If underflow occurs, the difference wraps from 0 to 18. If the result, adding or subtracting, is greater than 10, the top 4 bits in the 8-bit result is set to represent decimal 1 while the bottom 4 bits represent the ones place by subtracting 10 from the result.

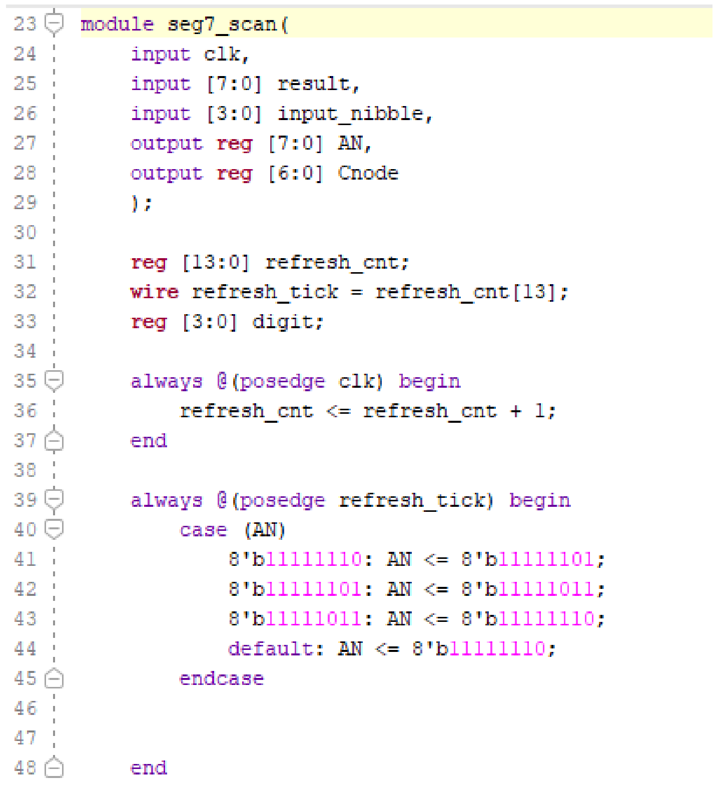
Control Decoder



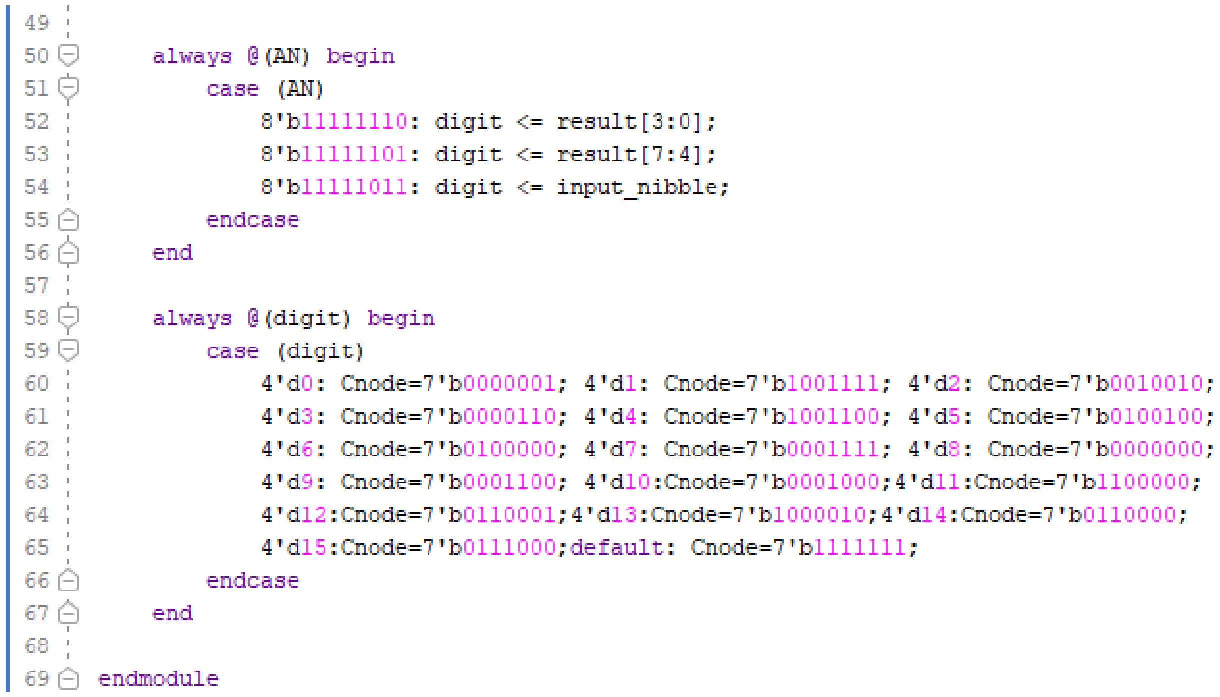
```
module control_decoder(
    input SW5, SW6, SW7, SW8,
    output [3:0] nibble
    );

    assign nibble = {SW8, SW7, SW6, SW5};
endmodule
```

This module takes the 4 input switches (SW[5:8]) and stores them in a 4-bit nibble.

Seg7 Scan



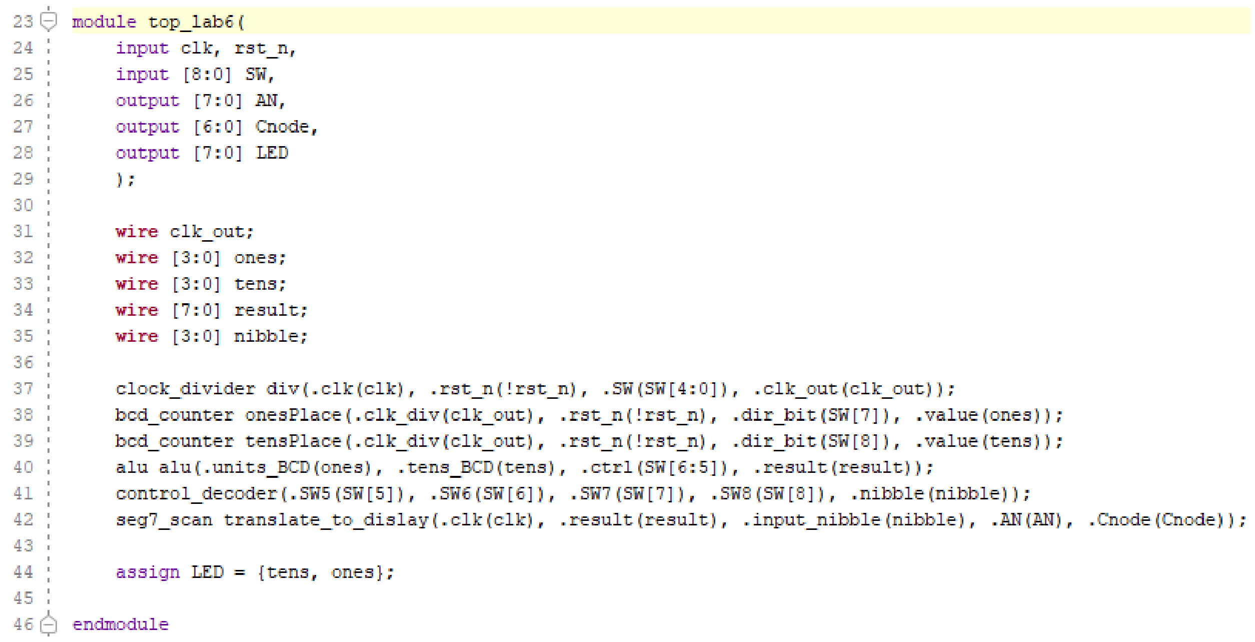
```verilog
module seg7_scan(
    input clk,
    input [7:0] result,
    input [3:0] input_nibble,
    output reg [7:0] AN,
    output reg [6:0] Cnode
    );

    reg [13:0] refresh_cnt;
    wire refresh_tick = refresh_cnt[13];
    reg [3:0] digit;

    always @(posedge clk) begin
        refresh_cnt <= refresh_cnt + 1;
    end

    always @(posedge refresh_tick) begin
        case (AN)
            8'b11111110: AN <= 8'b11111101;
            8'b11111101: AN <= 8'b11111011;
            8'b11111011: AN <= 8'b11111110;
            default: AN <= 8'b11111110;
        endcase


    end
```



```verilog

    always @(AN) begin
        case (AN)
            8'b11111110: digit <= result[3:0];
            8'b11111101: digit <= result[7:4];
            8'b11111011: digit <= input_nibble;
        endcase
    end

    always @(digit) begin
        case (digit)
            4'd0: Cnode=7'b0000001; 4'd1: Cnode=7'b1001111; 4'd2: Cnode=7'b0010010;
            4'd3: Cnode=7'b0000110; 4'd4: Cnode=7'b1001100; 4'd5: Cnode=7'b0100100;
            4'd6: Cnode=7'b0100000; 4'd7: Cnode=7'b0001111; 4'd8: Cnode=7'b0000000;
            4'd9: Cnode=7'b0001100; 4'd10:Cnode=7'b0001000;4'd11:Cnode=7'b1100000;
            4'd12:Cnode=7'b0110001;4'd13:Cnode=7'b1000010;4'd14:Cnode=7'b0110000;
            4'd15:Cnode=7'b0111000;default: Cnode=7'b1111111;
        endcase
    end

endmodule
```

We add a second clock divider to switch between the 3 digits on the display to turn on/off fast enough to create the illusion that they're all on with their unique value. I didn't take part of the counter to use as my select for which digit to turn on/off because I have 3 digits, and using part would mean one of the digits is on for longer than the other two digits, making it brighter than the others. What I did instead is make the turn/off each digit in order, triggered by only the positive edge of the MSB to create the illusion while giving each display the same brightness. The digit to display at any given point depends on what digit I currently have selected. This way, every digit stays on while having their unique values. The digit is translated to the proper segments to turn on to represent that digit on the display.

Top Level Implementation



```verilog
module top_lab6(
    input clk, rst_n,
    input [8:0] SW,
    output [7:0] AN,
    output [6:0] Cnode,
    output [7:0] LED
    );

    wire clk_out;
    wire [3:0] ones;
    wire [3:0] tens;
    wire [7:0] result;
    wire [3:0] nibble;

    clock_divider div(.clk(clk), .rst_n(!rst_n), .SW(SW[4:0]), .clk_out(clk_out));
    bcd_counter onesPlace(.clk_div(clk_out), .rst_n(!rst_n), .dir_bit(SW[7]), .value(ones));
    bcd_counter tensPlace(.clk_div(clk_out), .rst_n(!rst_n), .dir_bit(SW[8]), .value(tens));
    alu alu(.units_BCD(ones), .tens_BCD(tens), .ctrl(SW[6:5]), .result(result));
    control_decoder(.SW5(SW[5]), .SW6(SW[6]), .SW7(SW[7]), .SW8(SW[8]), .nibble(nibble));
    seg7_scan translate_to_dislay(.clk(clk), .result(result), .input_nibble(nibble), .AN(AN), .Cnode(Cnode));

    assign LED = {tens, ones};

endmodule
```

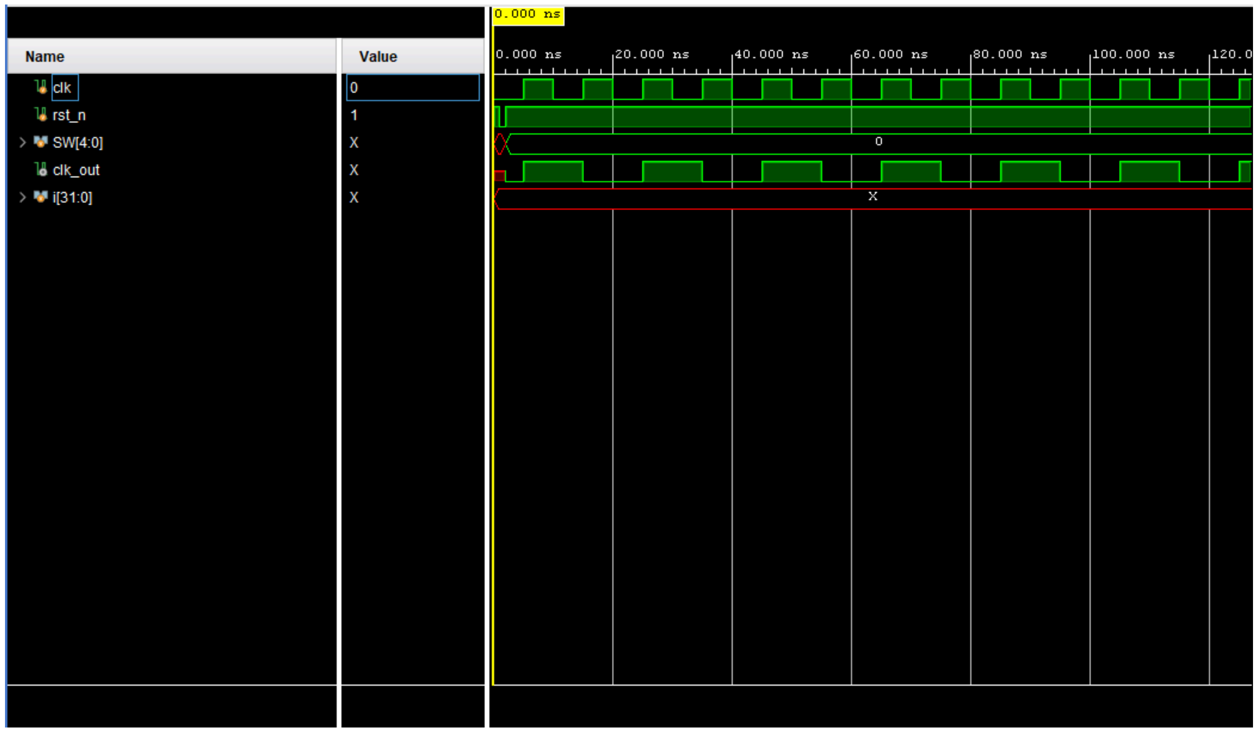
Now, we connect everything together. We grab the base clock from the development board and put that into the clock divider for its speed to be controlled by 5 switches on our board. That resulting clock goes into both BCD counters. Whether they increment or decrement depend on two other switches on the board. The resulting digits then get put into an ALU to be added/subtracted depending on the input of two more switches on the board. The control decoder handles the input of the two sets of two switches mentioned to be displayed with the 0-18 result on the board. Both the result and control get sent to the seg7_scan module to be translated to which segments on the display to turn on to represent their respective values. Finally, the two values being operated on are represented by 8 LED's on the board.

Simulation Waveform

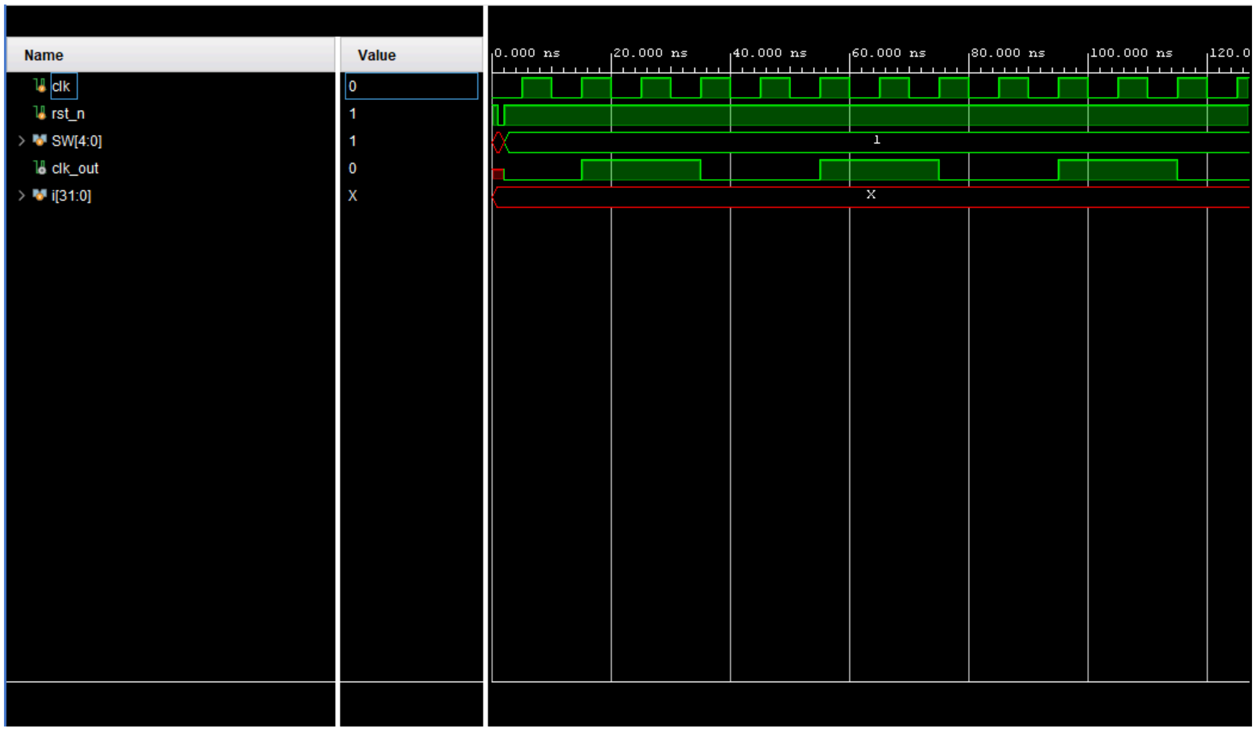
Clock Divider

To test the speed-varying clock divider, we show the reset works. Then, show the clock output when no switches are on (bit 0 is selected). From there, flip SW0 switch (bit 1 is selected), and show the clock output. Repeat this for every switch.

No Switches On

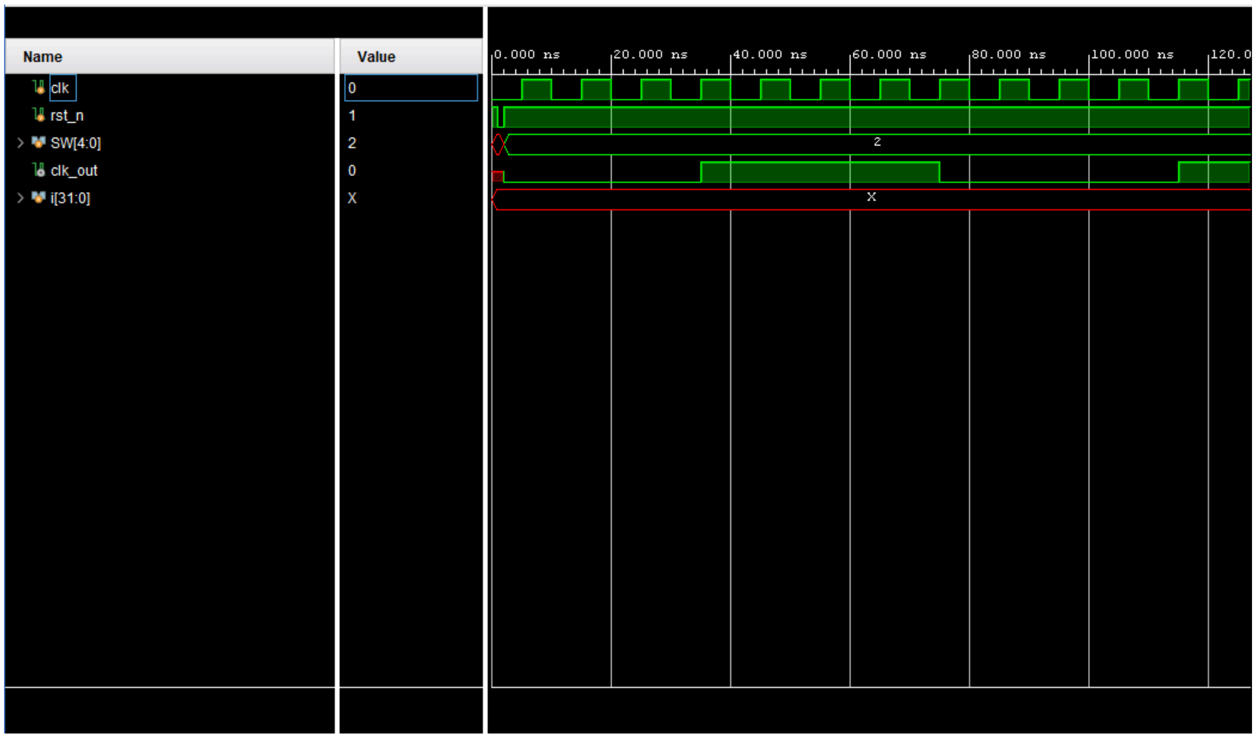


SW0 On



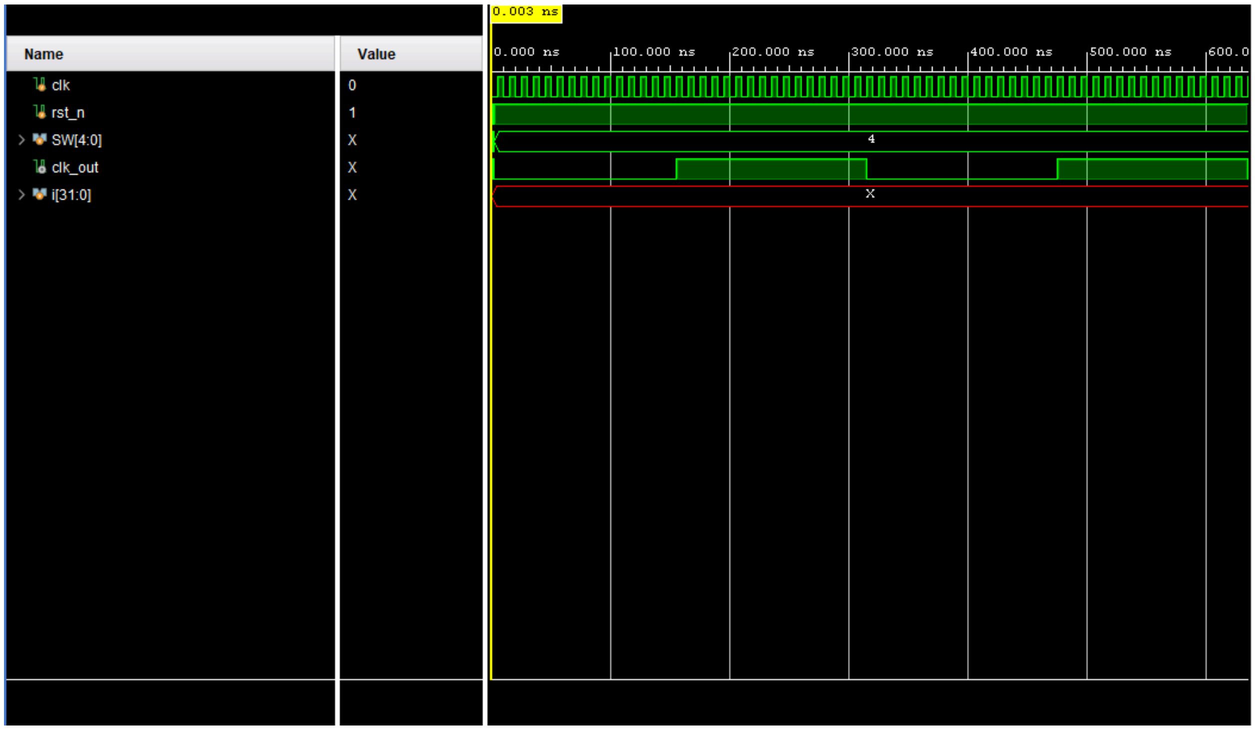
Time between positive edge trigger: 20ns

SW1 On



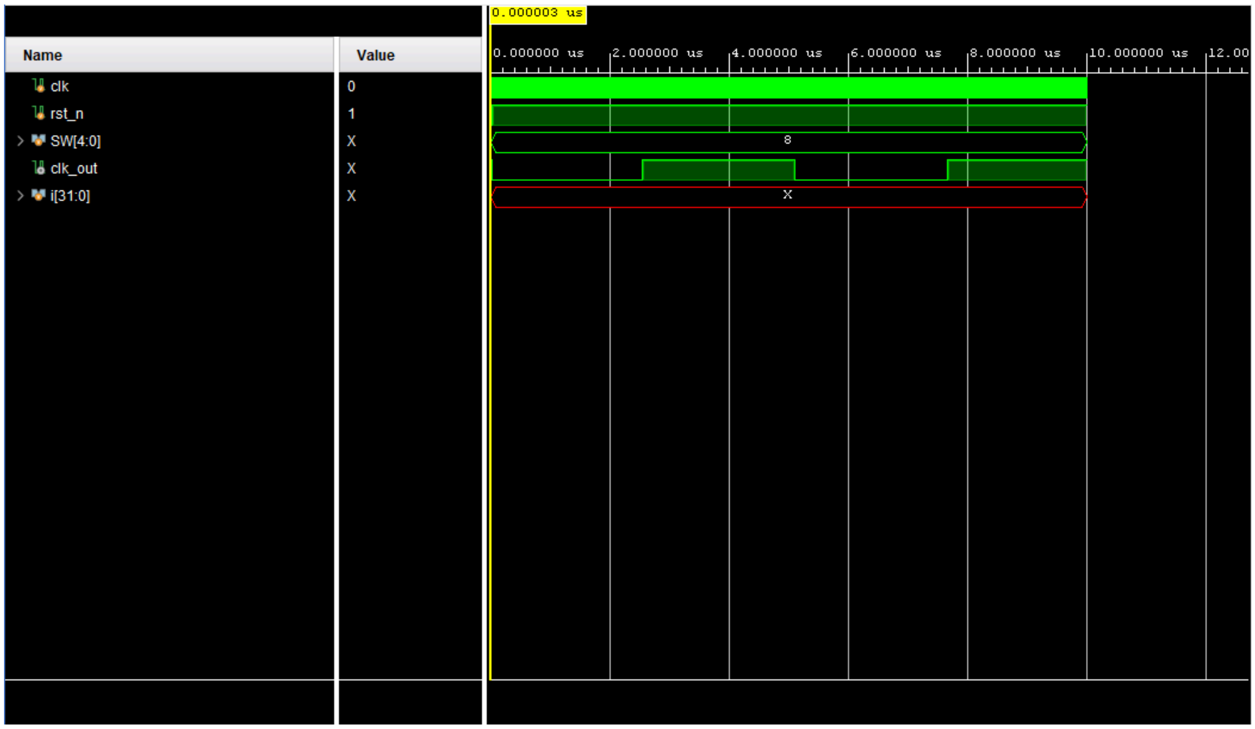
Time between positive edge trigger: 80ns

SW2 On



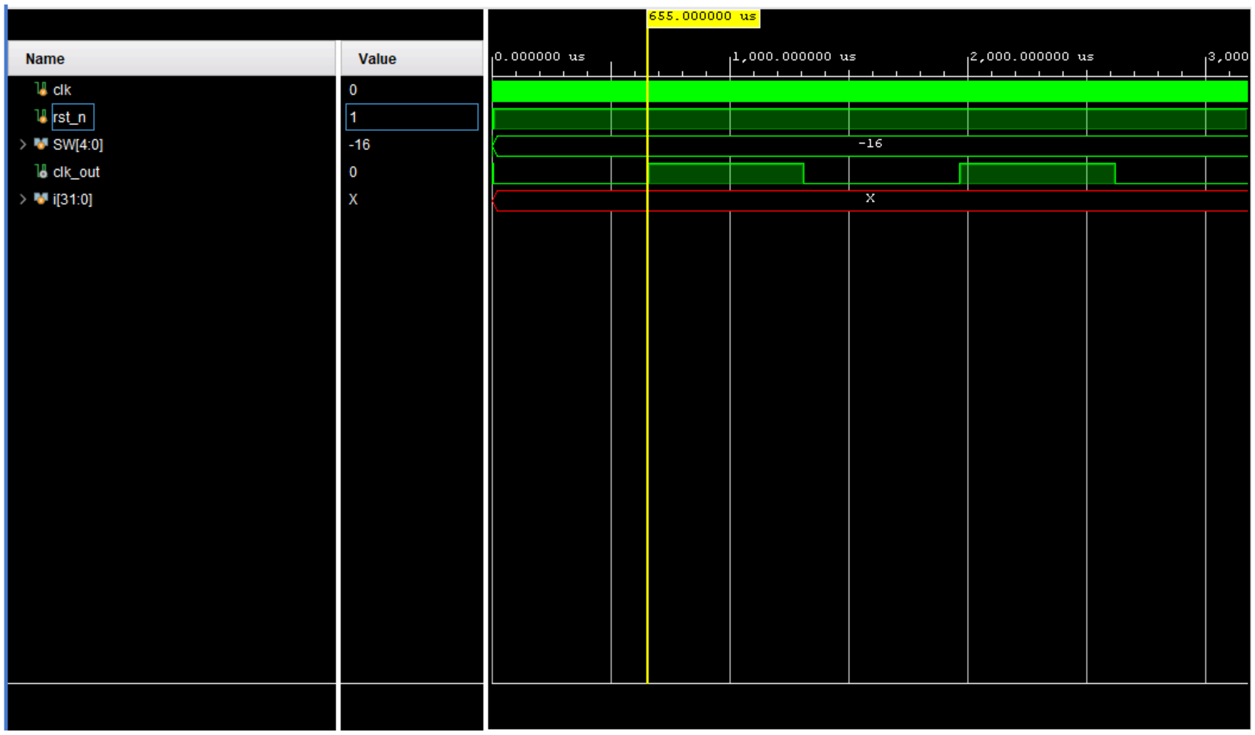
Time between positive edge trigger: 320ns

SW3 On



<u>Time between positive edge trigger: 5.12us</u>

<u>SW4 ON</u>

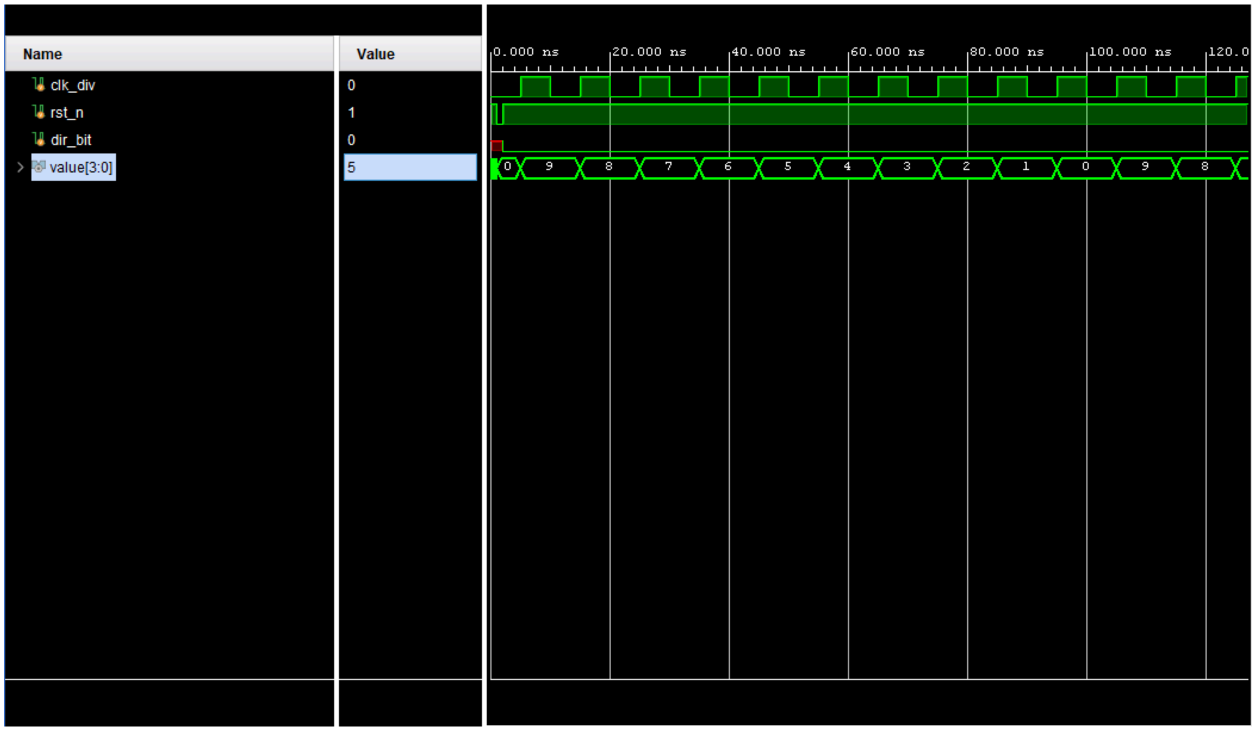


Time between positive edge trigger: 1310us

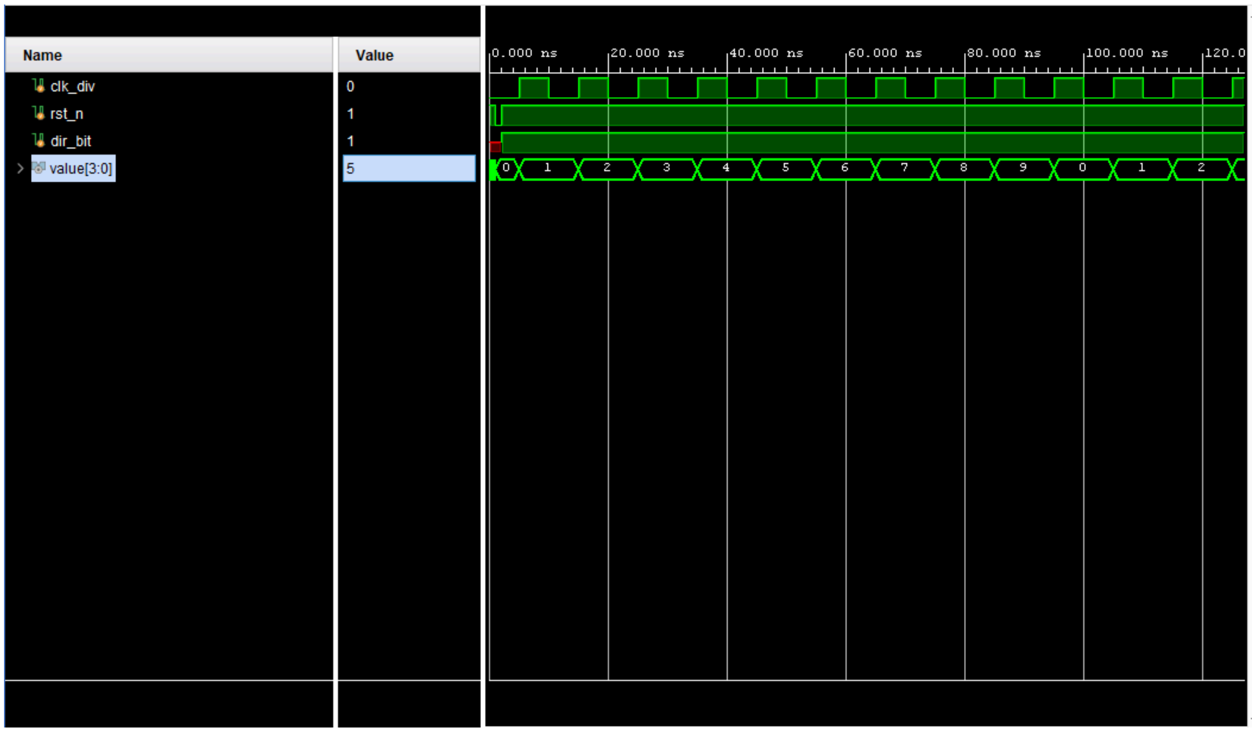
Turning on the leftmost switches increases the time between positive edge triggers for the output clock. The less switches on, the less time between positive edge triggers will result for the output clock. These times don't represent actual times when implemented on the board, since that uses a 100Mhz main clock while we're using a clock that's billions of times faster due to time scaling. In every test, we negative-edge trigger rst_n to simulate the reset in every scenario.

BCD Counter

We will demonstrate the BCD decrementing from 9 to 0, resetting to 9 when decrementing and the current value is 0.

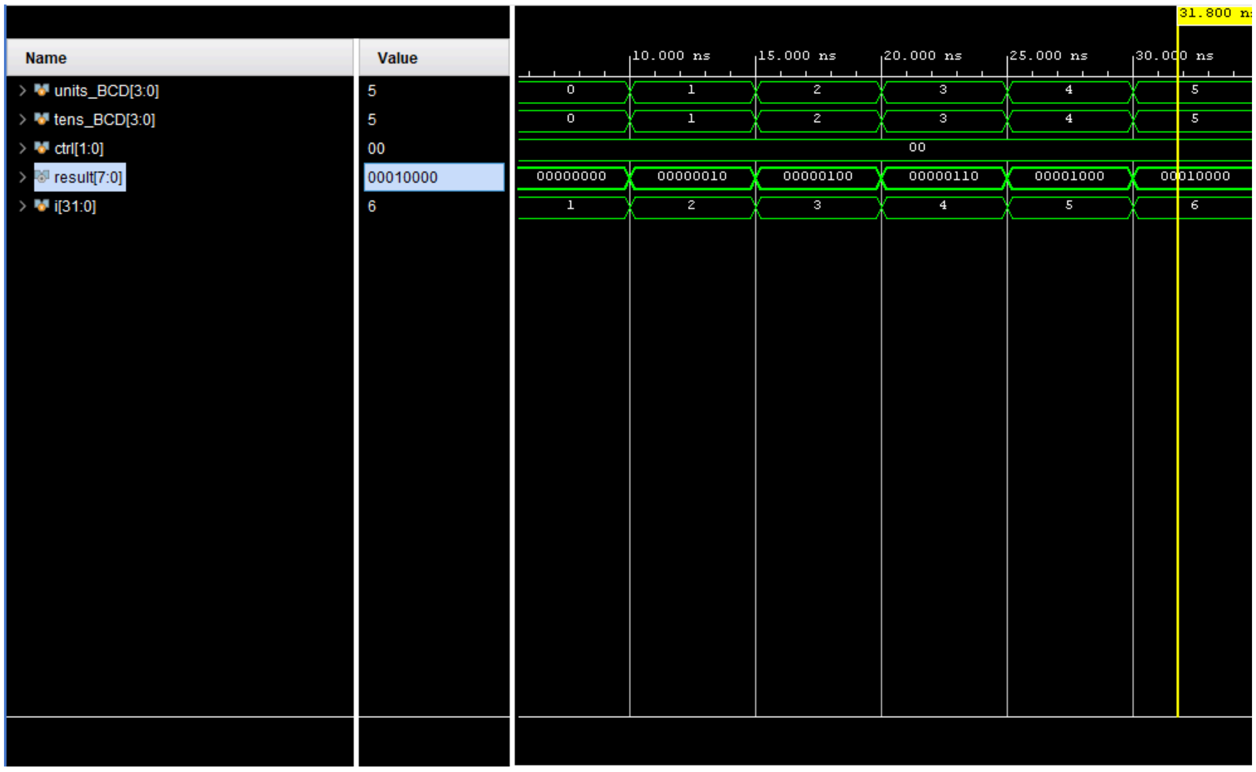


This waveform demonstrates the BCD decrementing from 9 to 0, resetting to 9 when decrementing and the current value is 0.



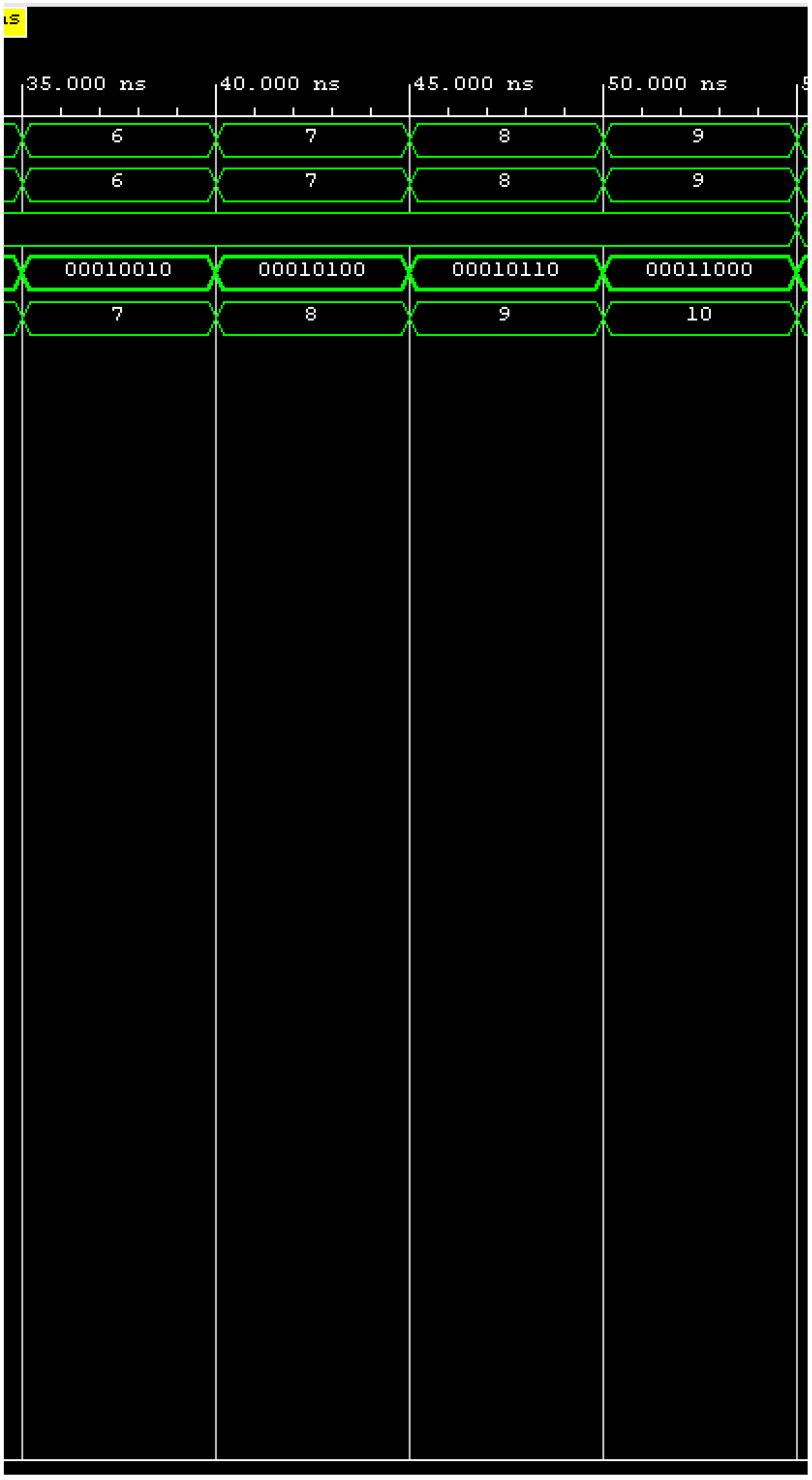
This waveform shows the BCD counter incrementing from 0 to 9, resetting to 0 when incrementing and the current value is 9.

ALU

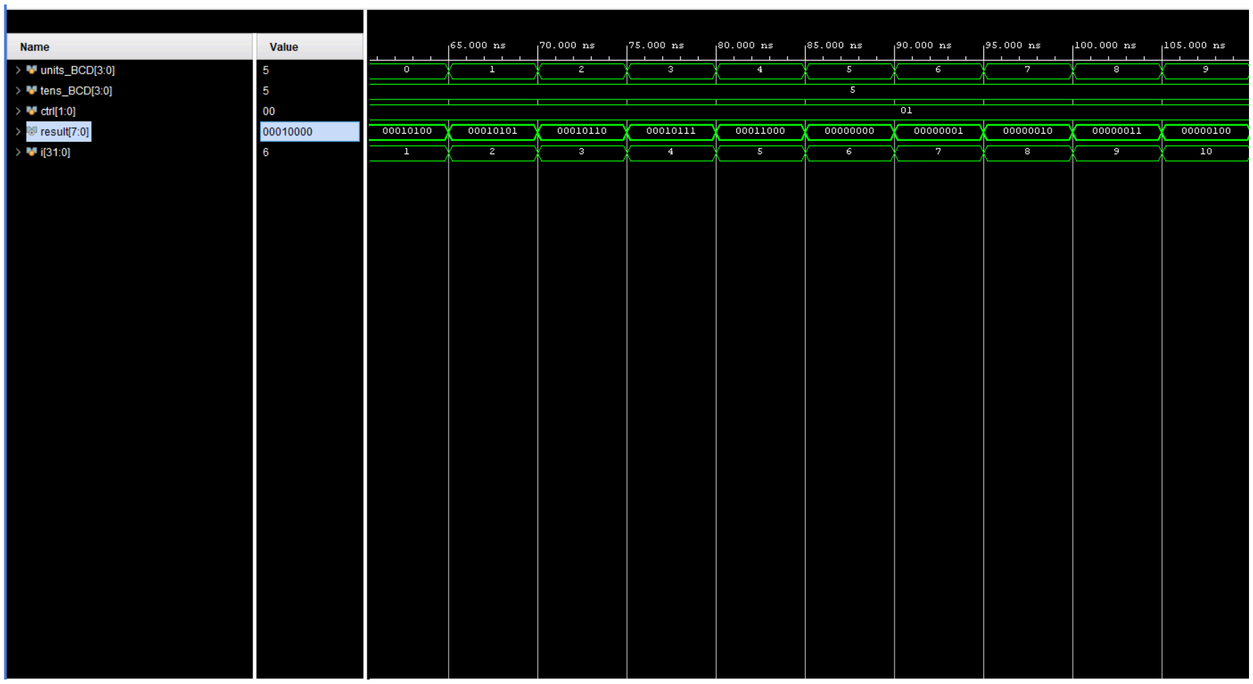


Name	Value
units_BCD[3:0]	5
tens_BCD[3:0]	5
ctrl[1:0]	00
result[7:0]	00010000
i[31:0]	6

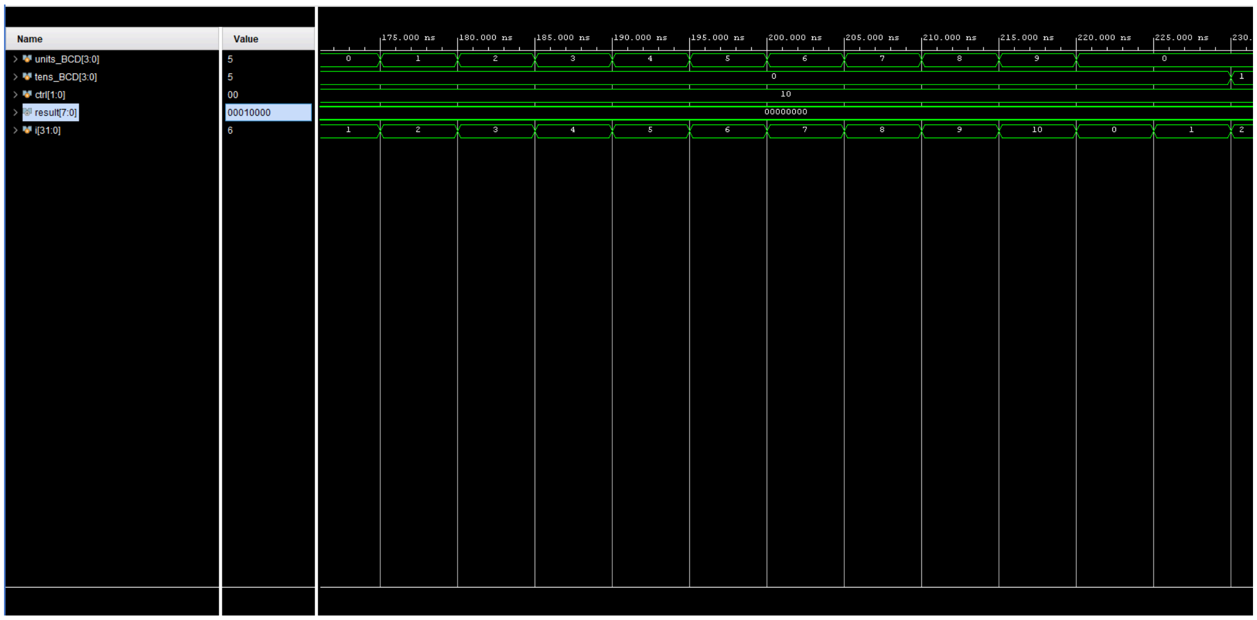
```
units_BCD  0         1         2         3         4         5
tens_BCD   0         1         2         3         4         5
ctrl                           00
result     00000000  00000010  00000100  00000110  00001000  00010000
i          1         2         3         4         5         6
```

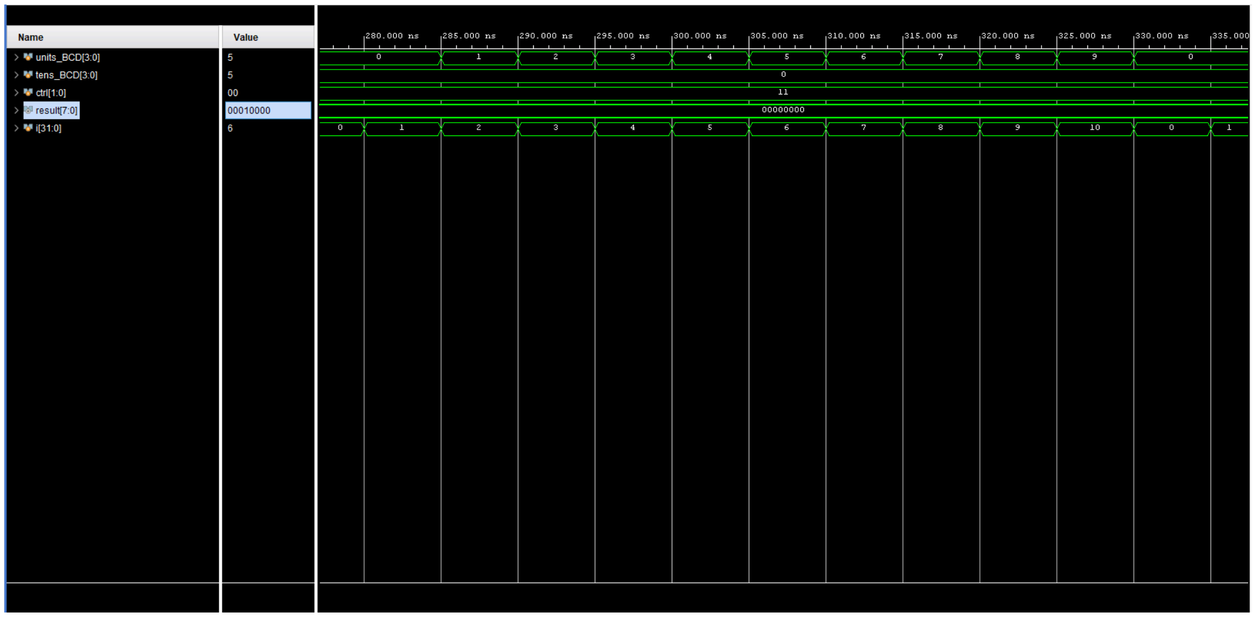


The first two waveforms show when control is 00 (add the two BCD numbers) and both numbers are incrementing. Every time both numbers increment, the result also increases by a value of decimal 2.



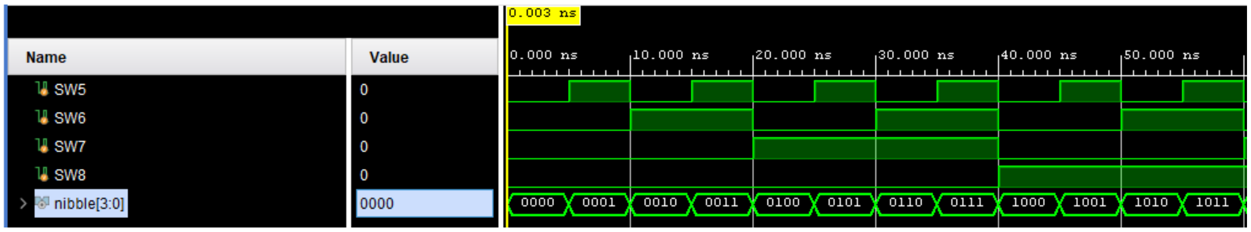
This waveform tests when the control is 01 (subtract the tens BCD from the units BCD). Values 0-4 in the units BCD handles cases where underflow occurs. The result wraps around 0 and subtracts from 18. For example, 0-5 in this case is 14. Values past 4 will result in 0 or positive result when subtracting 5 so we don't have to worry about underflow in this case.

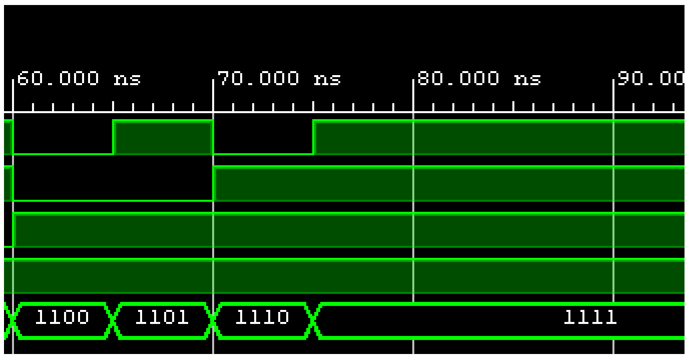




The two waveforms above represent the result when the control bits are either 10 or 11. We expect the result to be 0 and the actual result is 0.

Control Decoder





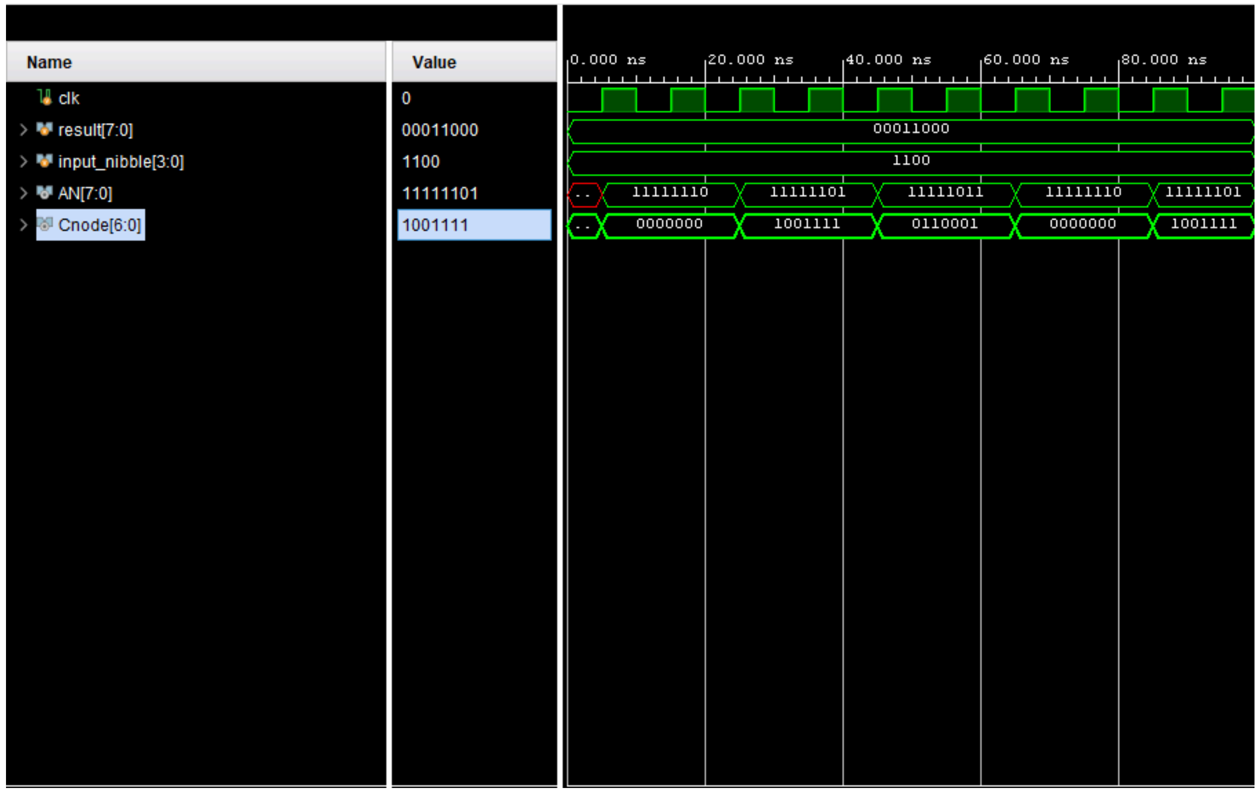
The two waveforms above represent the 4-bit nibble given all possible outputs from SW[8:5]

Seg 7 Scan

We demonstrate the Seg 7 Scan module by giving an example with correct output.

Example - SW8 and SW7 are up, SW6 and SW5 are down. Result is 18.

C18 is expected to be on the seven segment display



```
Name               Value
clk                0
result[7:0]        00011000
input_nibble[3:0]  1100
AN[7:0]            11111101
Cnode[6:0]         1001111
```

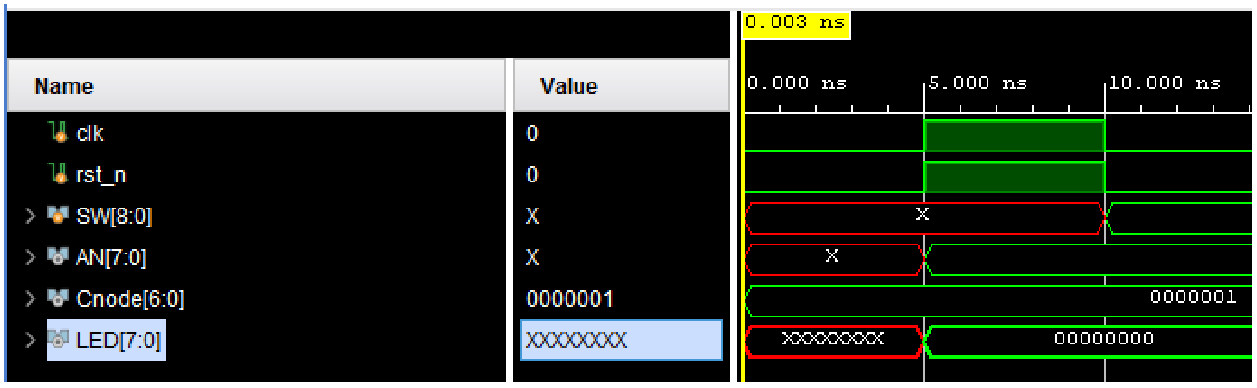
```
result[7:0]        00011000
input_nibble[3:0]  1100
AN[7:0]            ..  11111110  11111101  11111011  11111110  11111101
Cnode[6:0]         ..  0000000   1001111   0110001   0000000   1001111
```

The rightmost digit displays “8”. The middle digit displays “1”. The leftmost digit displays “C”. Therefore the output on the seven segment display is C18, as expected.

Top Level Implementation

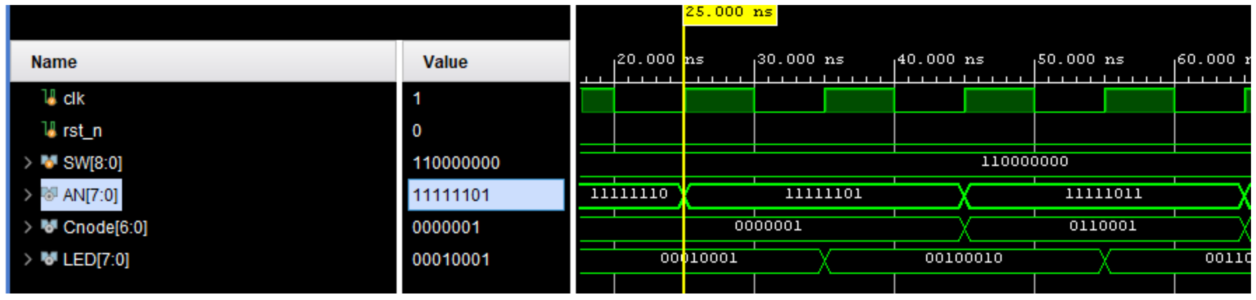
Due to the many possibilities to control the interface, I will give a couple of examples to demonstrate the top level implementation works.

Example 1 - Both counters are increasing and the two numbers will be added



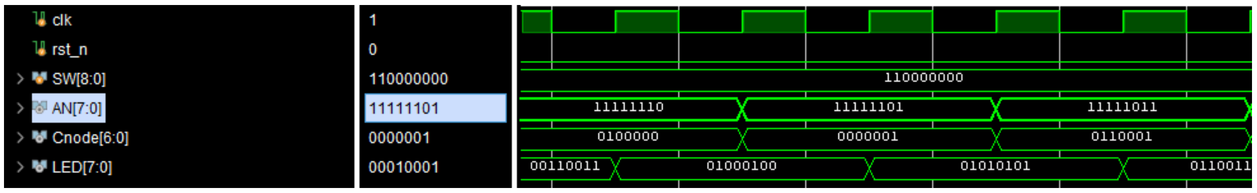
Name	Value
clk	0
rst_n	0
SW[8:0]	X
AN[7:0]	X
Cnode[6:0]	0000001
LED[7:0]	XXXXXXXX

This waveform shows that pressing the rst_n button resets the project and all outputs are now valid.



Name	Value
clk	1
rst_n	0
SW[8:0]	110000000
AN[7:0]	11111101
Cnode[6:0]	0000001
LED[7:0]	00010001

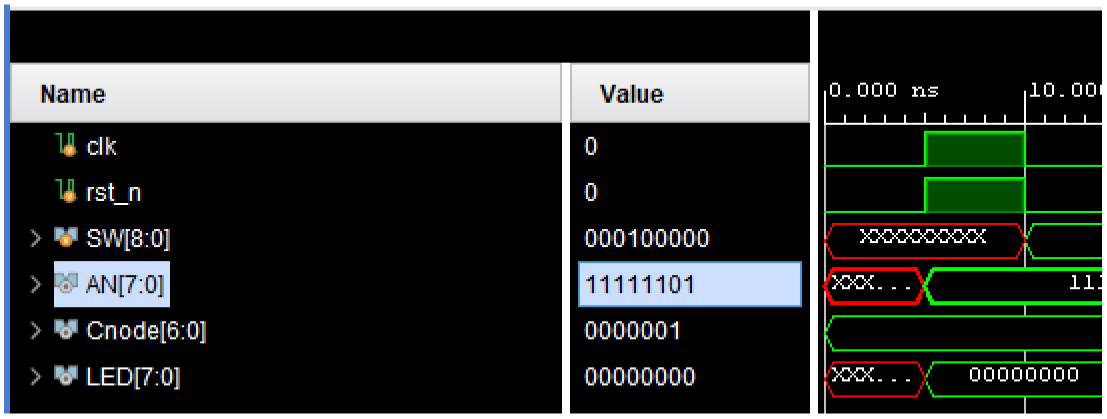
This waveform shows the result is “00” and the control displays “C”.



clk	1
rst_n	0
SW[8:0]	110000000
AN[7:0]	11111101
Cnode[6:0]	0000001
LED[7:0]	00010001

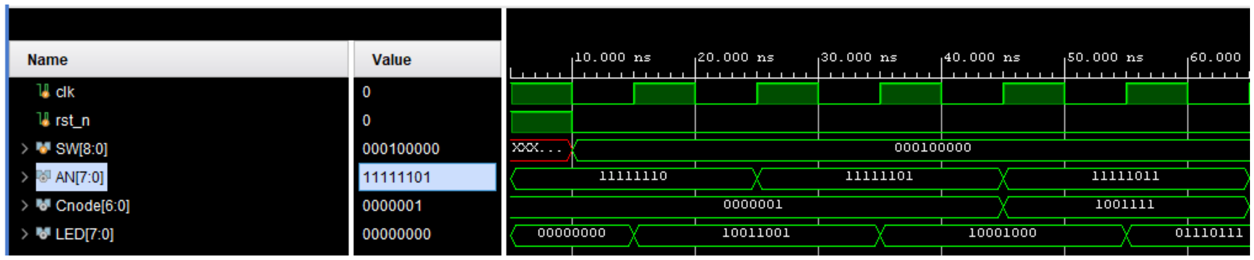
A couple of clock cycles later, the result is “06” and the control still displays “C”. We know this is true because The LED represents the first number to add being 3 and the second number to add also being 3.

Example 2 - Both counters are decreasing and the two numbers will be subtracted

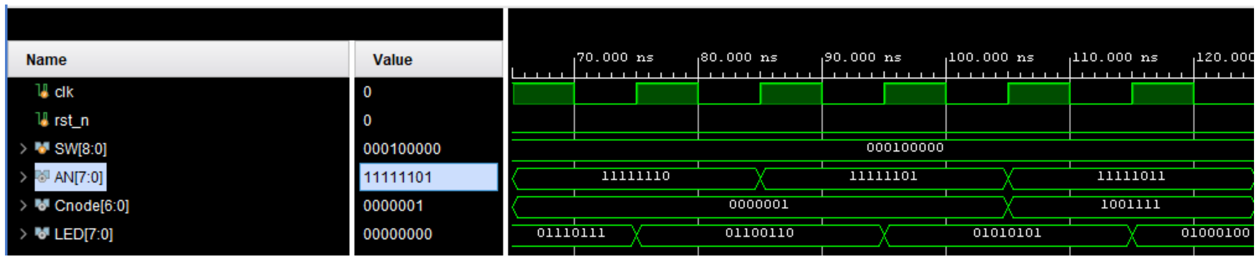


Name	Value
clk	0
rst_n	0
SW[8:0]	000100000
AN[7:0]	11111101
Cnode[6:0]	0000001
LED[7:0]	00000000

This waveform shows pressing the reset button for the testbench to have all valid outputs.



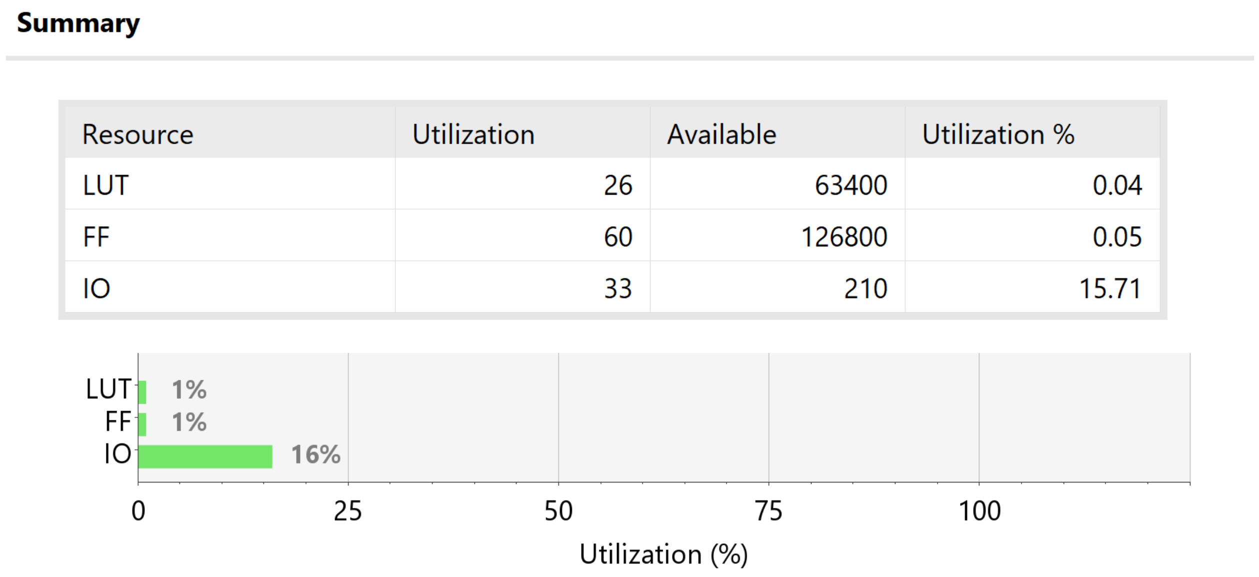
This waveform shows the result to be displayed is “00” with the digit representing the control displaying “1”.



The 7-segment display still shows “100”, making the control digit “1” and the result “00”. This is because the two numbers to operate on started at the same value, decrement at the same time, and are decrementing at the same rate. This means both numbers will always be the same, and since we’re in subtraction mode, the result will always be “00”.

Implementation

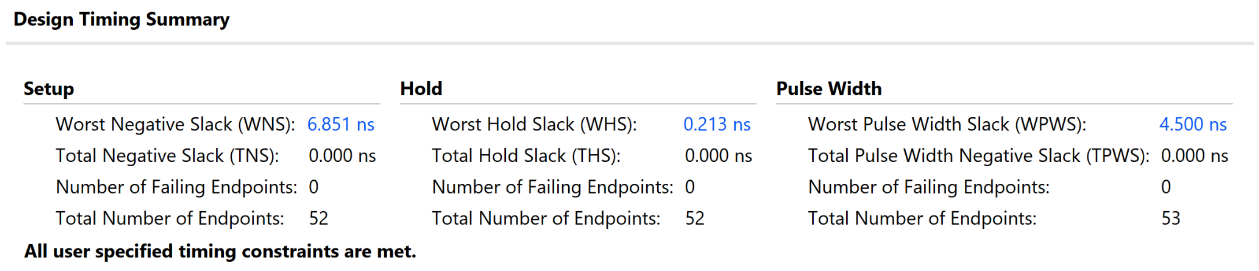
Utilization Table:



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