ECE 3300 – Lab 5 Report
BCD Up/Down Counter on 7-Segment Display
Team Name: Group V
Date: July 21, 2025

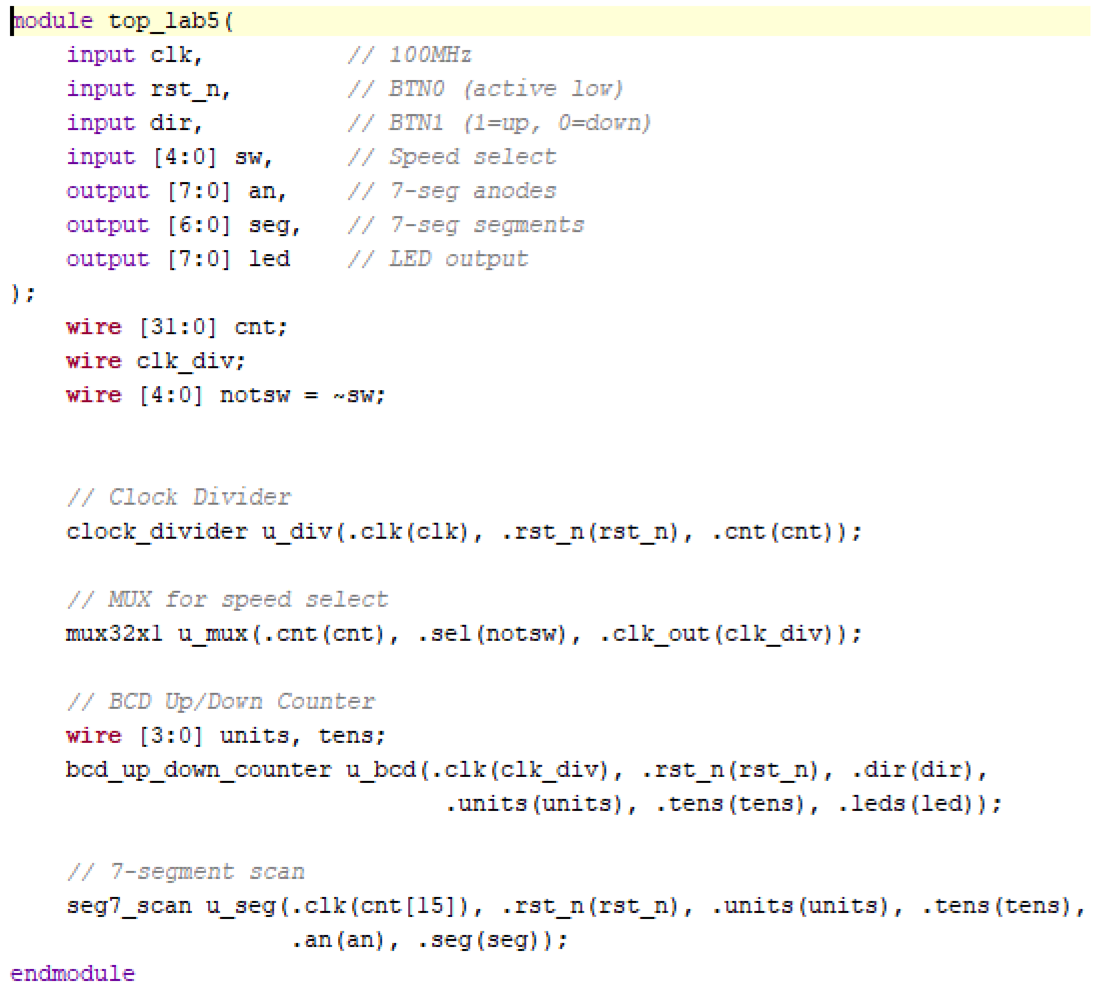
Group Members
Nathan Marlow, Khristian Chan

1. Objective

The objective of this lab was to design and implement a two-digit BCD up/down counter on the Nexys A7 FPGA using Verilog. The system features speed control via SW[4:0], direction control using BTN1, and reset with BTN0. A 32-bit counter and 32×1 Mux were used for clock division, and the output is displayed on a dual-digit 7-segment display with multiplexing, while also being mirrored on LED[7:0] for debugging.

2. Verilog Code

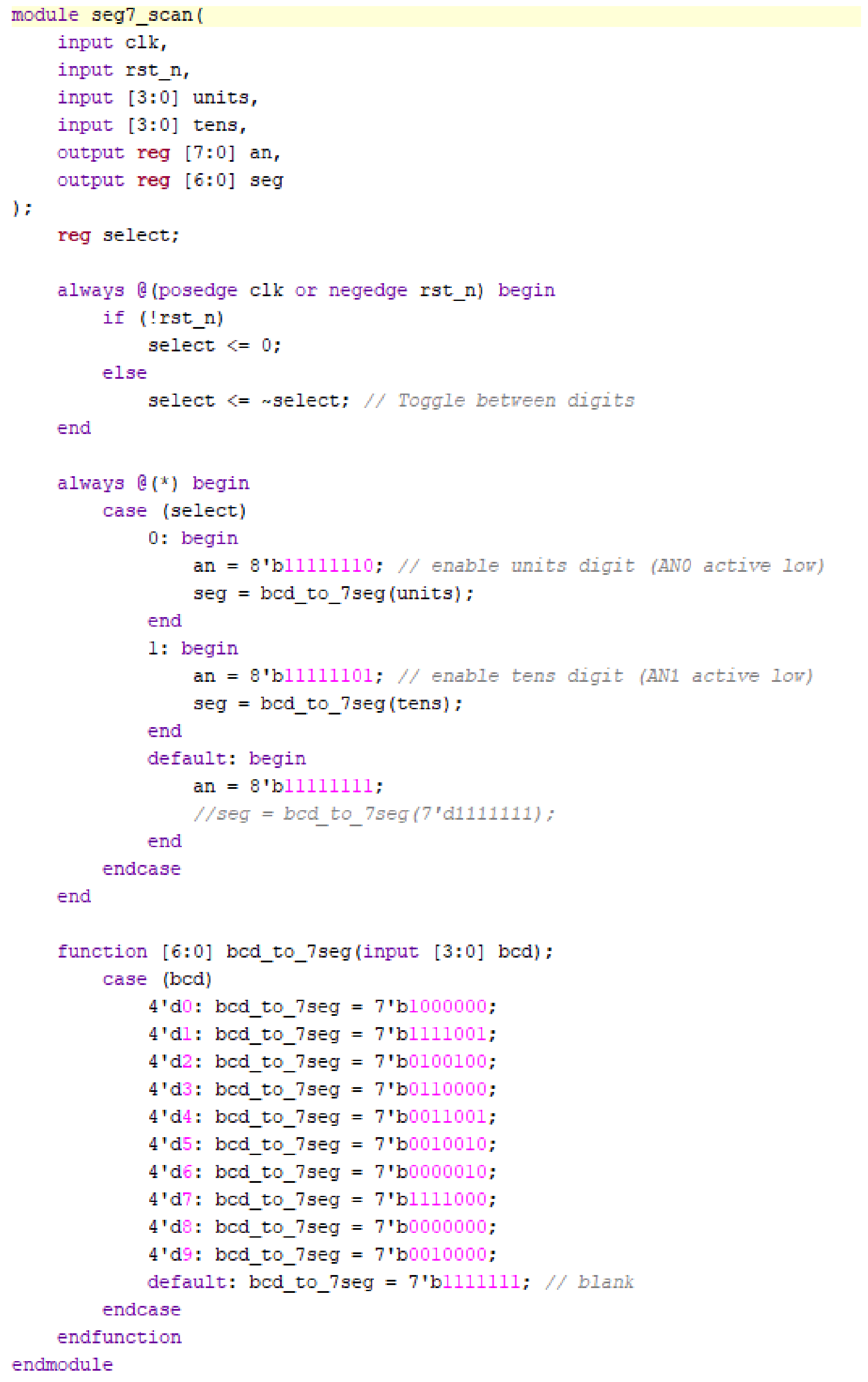
Top Module:



```
module top_lab5(
    input clk,          // 100MHz
    input rst_n,        // BTN0 (active low)
    input dir,          // BTN1 (1=up, 0=down)
    input [4:0] sw,     // Speed select
    output [7:0] an,    // 7-seg anodes
    output [6:0] seg,   // 7-seg segments
    output [7:0] led    // LED output
);
    wire [31:0] cnt;
    wire clk_div;
    wire [4:0] notsw = ~sw;


    // Clock Divider
    clock_divider u_div(.clk(clk), .rst_n(rst_n), .cnt(cnt));

    // MUX for speed select
    mux32x1 u_mux(.cnt(cnt), .sel(notsw), .clk_out(clk_div));

    // BCD Up/Down Counter
    wire [3:0] units, tens;
    bcd_up_down_counter u_bcd(.clk(clk_div), .rst_n(rst_n), .dir(dir),
                               .units(units), .tens(tens), .leds(led));

    // 7-segment scan
    seg7_scan u_seg(.clk(cnt[15]), .rst_n(rst_n), .units(units), .tens(tens),
                    .an(an), .seg(seg));
endmodule
```

Our top module takes all of the other modules and instantiates them to work all at the same time.

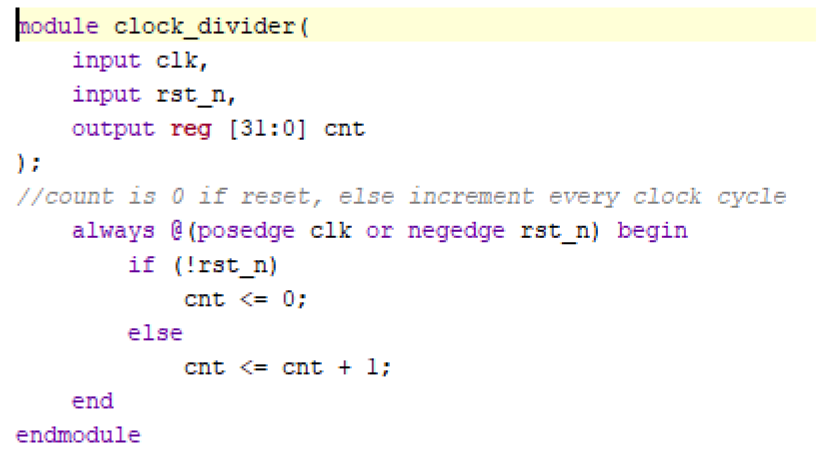
7Seg Driver:



```verilog
module seg7_scan(
    input clk,
    input rst_n,
    input [3:0] units,
    input [3:0] tens,
    output reg [7:0] an,
    output reg [6:0] seg
);
    reg select;

    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            select <= 0;
        else
            select <= ~select; // Toggle between digits
    end

    always @(*) begin
        case (select)
            0: begin
                an = 8'b11111110; // enable units digit (AN0 active low)
                seg = bcd_to_7seg(units);
            end
            1: begin
                an = 8'b11111101; // enable tens digit (AN1 active low)
                seg = bcd_to_7seg(tens);
            end
            default: begin
                an = 8'b11111111;
                //seg = bcd_to_7seg(7'd1111111);
            end
        endcase
    end

    function [6:0] bcd_to_7seg(input [3:0] bcd);
        case (bcd)
            4'd0: bcd_to_7seg = 7'b1000000;
            4'd1: bcd_to_7seg = 7'b1111001;
            4'd2: bcd_to_7seg = 7'b0100100;
            4'd3: bcd_to_7seg = 7'b0110000;
            4'd4: bcd_to_7seg = 7'b0011001;
            4'd5: bcd_to_7seg = 7'b0010010;
            4'd6: bcd_to_7seg = 7'b0000010;
            4'd7: bcd_to_7seg = 7'b1111000;
            4'd8: bcd_to_7seg = 7'b0000000;
            4'd9: bcd_to_7seg = 7'b0010000;
            default: bcd_to_7seg = 7'b1111111; // blank
        endcase
    endfunction
endmodule
```

Similar to our previous lab, we are driving a seven segment display. This time, we only drive two of the anodes, so this simplifies the process a lot, and makes poll rate quicker for the driver. In our code, we have an 8 bit wide anode, instead of 2, so that we can set the other 6 anodes to high, turning them off.

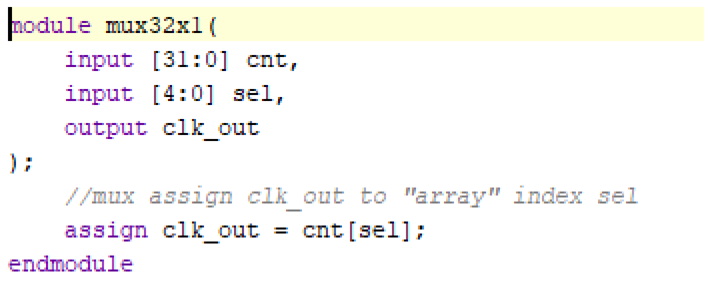
Clock Divider:



```verilog
module clock_divider(
    input clk,
    input rst_n,
    output reg [31:0] cnt
);
//count is 0 if reset, else increment every clock cycle
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            cnt <= 0;
        else
            cnt <= cnt + 1;
    end
endmodule
```

If reset is low, our counter is zero, if its high, it counts as normal

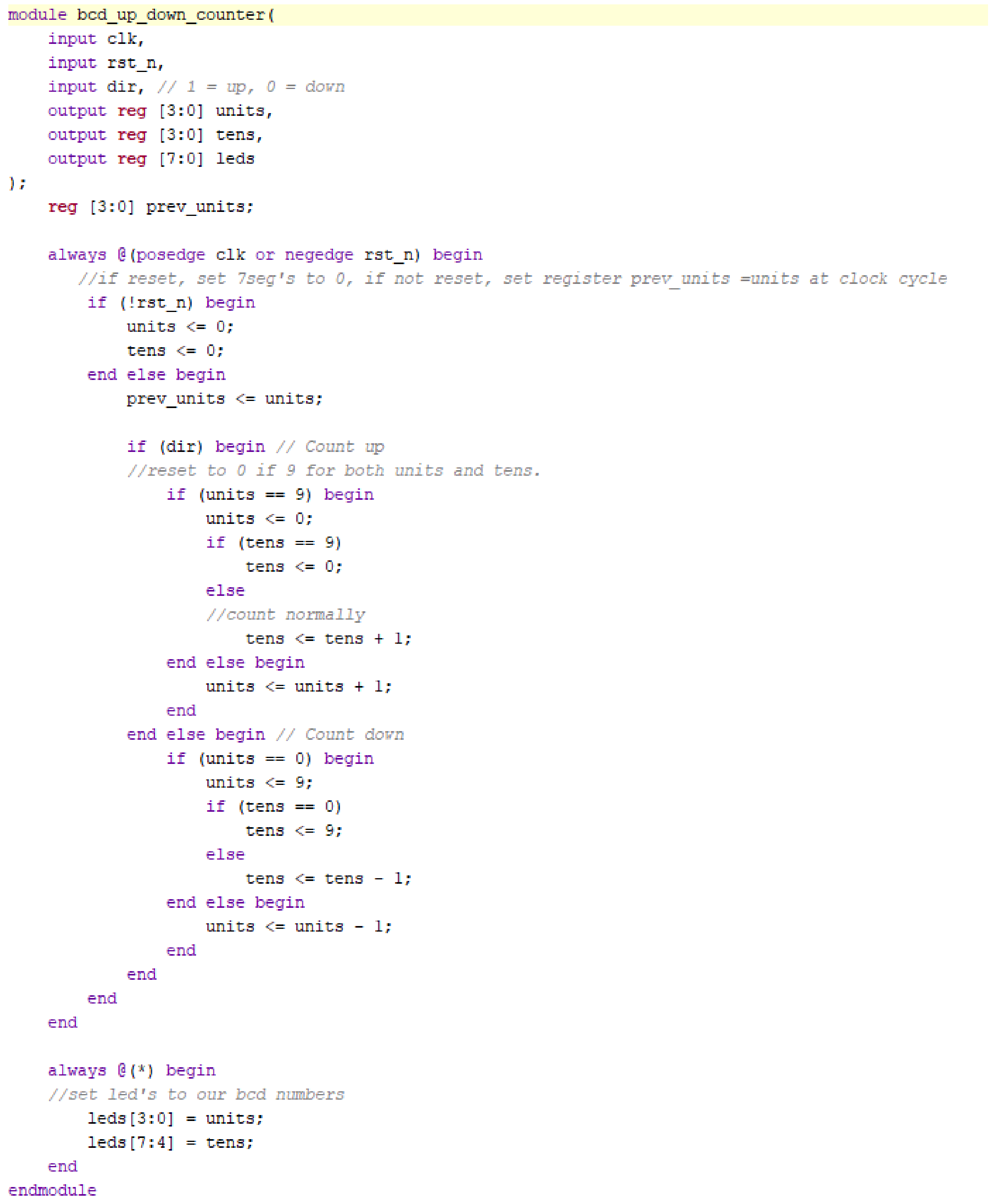
32x1 Mux:



```verilog
module mux32x1(
    input [31:0] cnt,
    input [4:0] sel,
    output clk_out
);
    //mux assign clk_out to "array" index sel
    assign clk_out = cnt[sel];
endmodule
```

This is a normal 32x1 multiplexer with a 5 bit select. As done before, we can use an assign statement.

BCD UP/Down Counter:

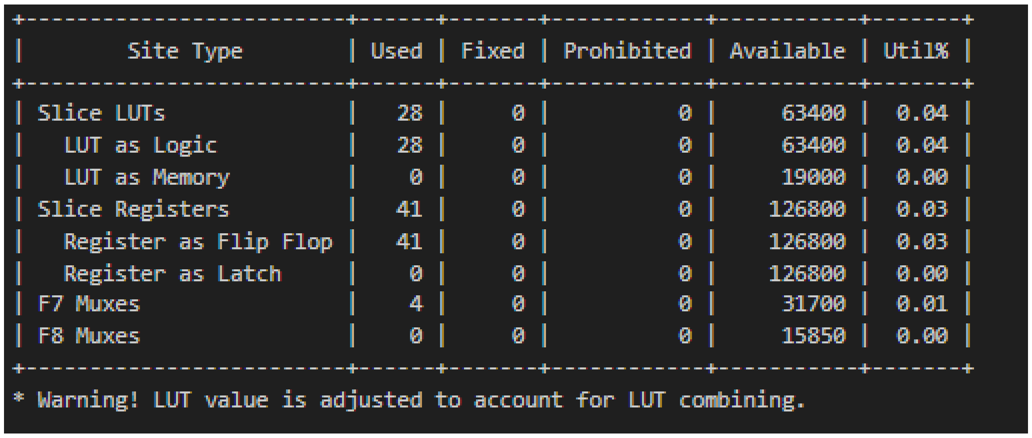


```
module bcd_up_down_counter(
    input clk,
    input rst_n,
    input dir, // 1 = up, 0 = down
    output reg [3:0] units,
    output reg [3:0] tens,
    output reg [7:0] leds
);
    reg [3:0] prev_units;

    always @(posedge clk or negedge rst_n) begin
       //if reset, set 7seg's to 0, if not reset, set register prev_units =units at clock cycle
        if (!rst_n) begin
            units <= 0;
            tens <= 0;
        end else begin
            prev_units <= units;

            if (dir) begin // Count up
            //reset to 0 if 9 for both units and tens.
                if (units == 9) begin
                    units <= 0;
                    if (tens == 9)
                        tens <= 0;
                    else
                    //count normally
                        tens <= tens + 1;
                end else begin
                    units <= units + 1;
                end
            end else begin // Count down
                if (units == 0) begin
                    units <= 9;
                    if (tens == 0)
                        tens <= 9;
                    else
                        tens <= tens - 1;
                end else begin
                    units <= units - 1;
                end
            end
        end
    end

    always @(*) begin
    //set led's to our bcd numbers
        leds[3:0] = units;
        leds[7:4] = tens;
    end
endmodule
```

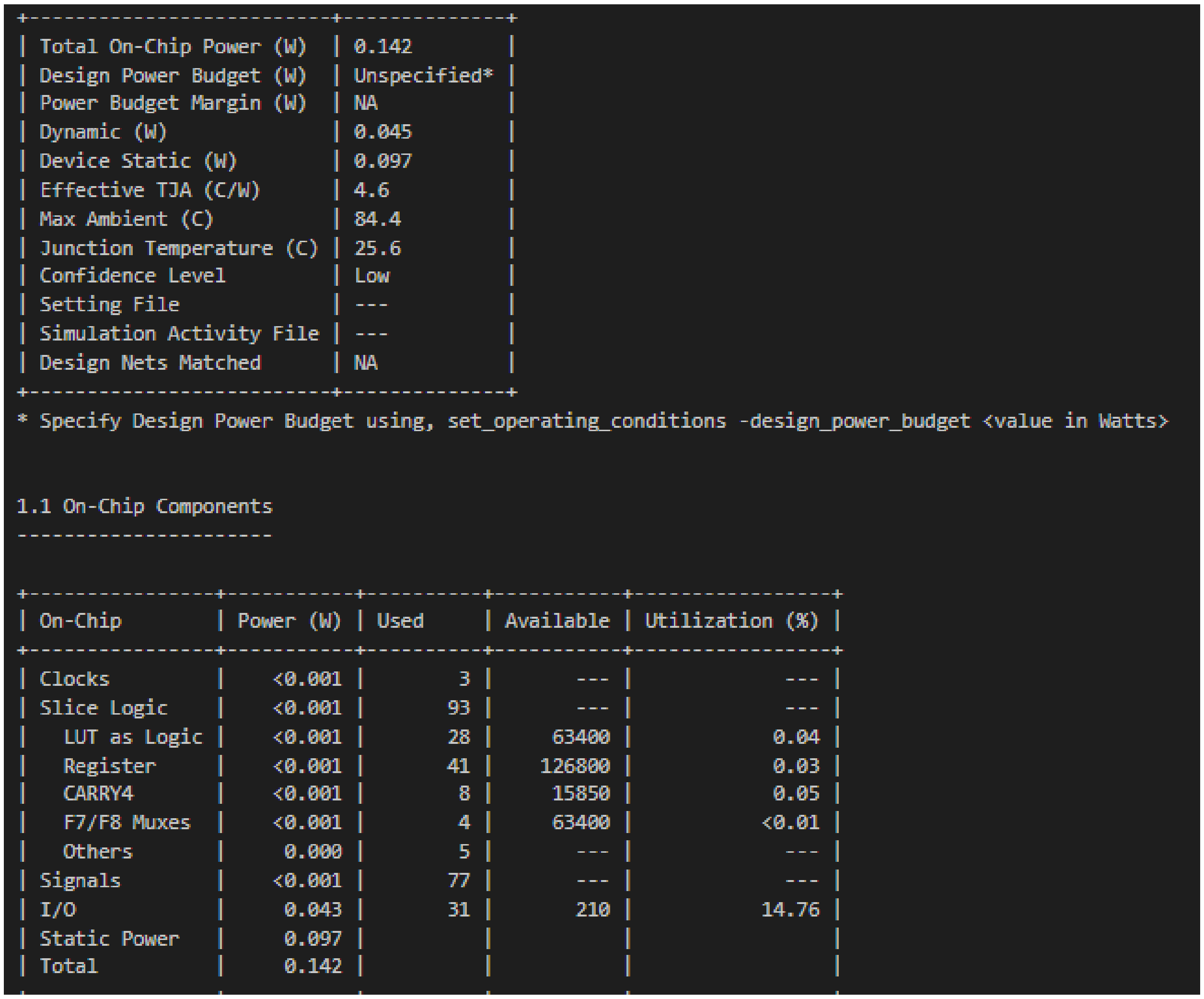
Lastly, this module essentially is our counter. It naturally counts down due to our dir naturally being low. When high, it will count up, with rollover when either tens or units go to 9. When dir is low, it will roll over the other direction, from 0 back to 9.

3. Implementation Results

After synthesizing and simulating the design in Vivado, the following results were observed:



```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs              |   28 |     0 |          0 |     63400 |  0.04 |
|   LUT as Logic          |   28 |     0 |          0 |     63400 |  0.04 |
|   LUT as Memory         |    0 |     0 |          0 |     19000 |  0.00 |
| Slice Registers         |   41 |     0 |          0 |    126800 |  0.03 |
|   Register as Flip Flop |   41 |     0 |          0 |    126800 |  0.03 |
|   Register as Latch     |    0 |     0 |          0 |    126800 |  0.00 |
| F7 Muxes                |    4 |     0 |          0 |     31700 |  0.01 |
| F8 Muxes                |    0 |     0 |          0 |     15850 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
* Warning! LUT value is adjusted to account for LUT combining.
```

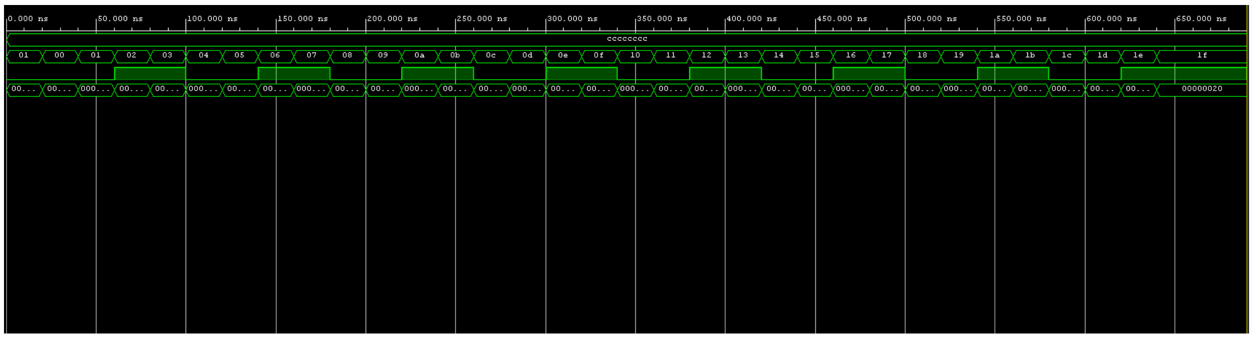


```
+--------------------------+--------------+
| Total On-Chip Power (W)  | 0.142        |
| Design Power Budget (W)  | Unspecified* |
| Power Budget Margin (W)  | NA           |
| Dynamic (W)              | 0.045        |
| Device Static (W)        | 0.097        |
| Effective TJA (C/W)      | 4.6          |
| Max Ambient (C)          | 84.4         |
| Junction Temperature (C) | 25.6         |
| Confidence Level         | Low          |
| Setting File             | ---          |
| Simulation Activity File | ---          |
| Design Nets Matched      | NA           |
+--------------------------+--------------+
* Specify Design Power Budget using, set_operating_conditions -design_power_budget <value in Watts>


1.1 On-Chip Components
----------------------

+----------------+-----------+----------+-----------+-----------------+
| On-Chip        | Power (W) | Used     | Available | Utilization (%) |
+----------------+-----------+----------+-----------+-----------------+
| Clocks         |    <0.001 |        3 |       --- |             --- |
| Slice Logic    |    <0.001 |       93 |       --- |             --- |
|   LUT as Logic |    <0.001 |       28 |     63400 |            0.04 |
|   Register     |    <0.001 |       41 |    126800 |            0.03 |
|   CARRY4       |    <0.001 |        8 |     15850 |            0.05 |
|   F7/F8 Muxes  |    <0.001 |        4 |     63400 |           <0.01 |
|   Others       |     0.000 |        5 |       --- |             --- |
| Signals        |    <0.001 |       77 |       --- |             --- |
| I/O            |     0.043 |       31 |       210 |           14.76 |
| Static Power   |     0.097 |          |           |                 |
| Total          |     0.142 |          |           |                 |
```

Simulation Results:

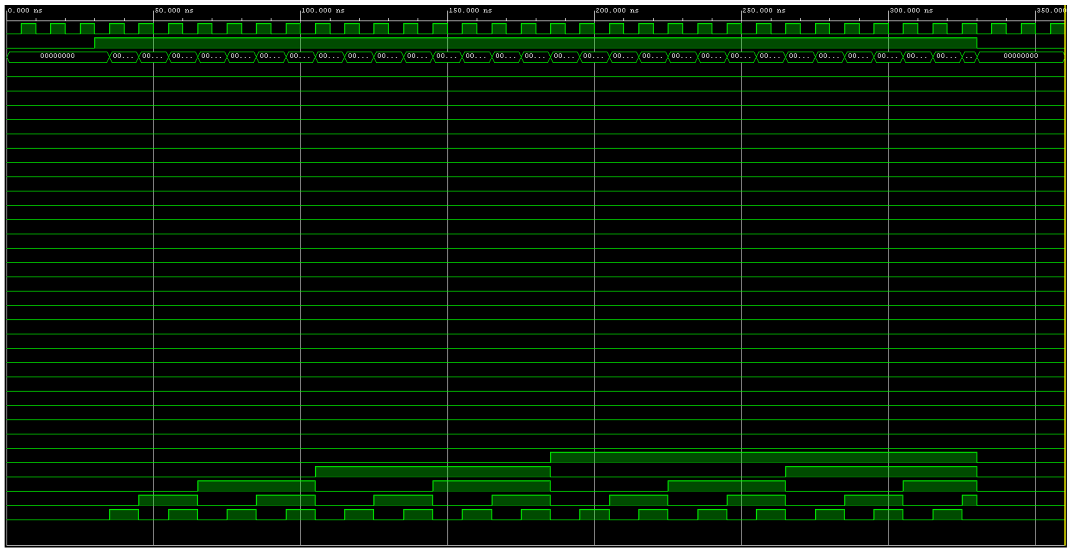
Since the simulation would take a very long time to go through all possible values, we decided to just simulate one switch input. The testbench is able to validate our fpga code.

mux32x1_tb.v



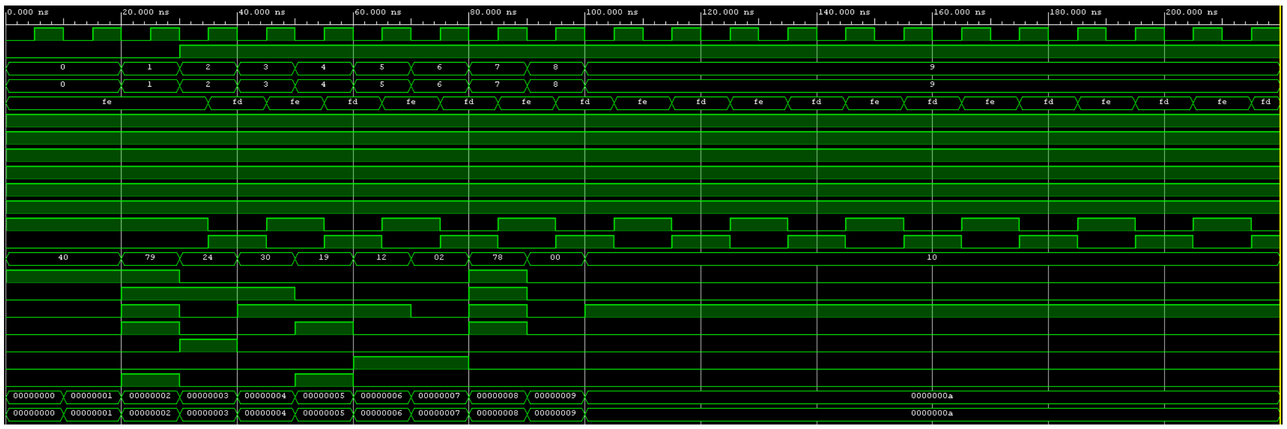
We see that our chosen input waveform (alternating 1’s and 0’s 11001100…) is reflected in our output waveform

clock_divider_tb.v



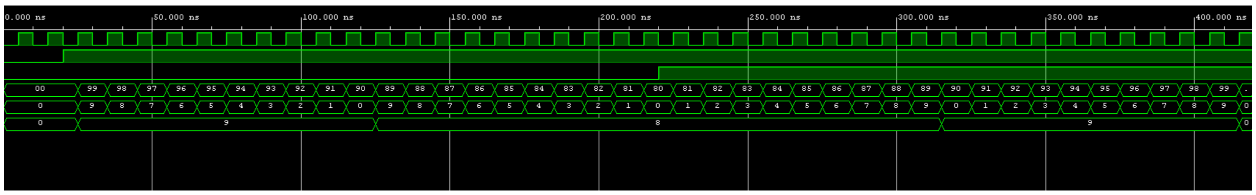
We see once our reset goes high, we start counting up, and once reset is pressed again our output goes low again.

seg7_scan_tb.v



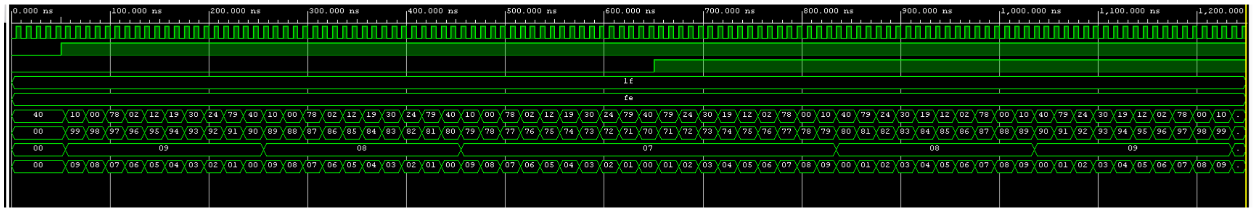
We see that our AN value for our unwanted bits is always high signifying that they are off. We see that our 7segment displays go through all values 0-9 on both displays.

bcd_updown_tb.v



We count down from 99, and we see that once dir goes high we count up.

top_lab5_tb.v



We see that we can count both down and up in this testbench, along with our reset working as intended.

6. Demonstration Video

https://youtu.be/_39TesYDkGE

Contributions:

Khristian Chan- 50%: Testbench, Simulation, Debugging, Report, Demo
Nathan Marlow- 50%: Implementation, Code, XDC, Report