California Polytechnic State University Pomona

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Digital Circuit Design Verilog

ECE 3300L

Report #6

Prepared by

Heba Hafez 017353323

Sean Wygant 017376658

Group Y

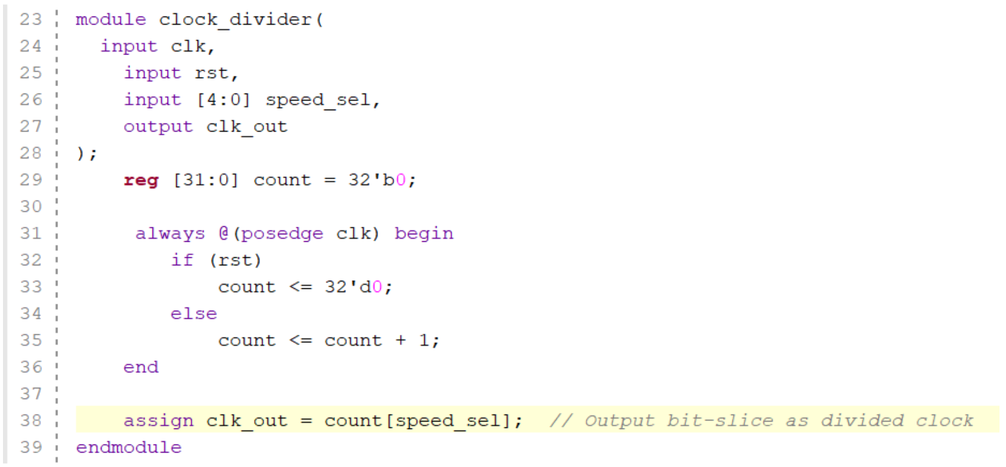
Mohamed Aly

July 28, 2025

Objective: Design two BCD counters, one for the one's place, and one for the tens place, to count up and down based on the switch input. Connect the counters to an ALU to complete addition and subtraction. Results will display on a three-digit 7-segment display, with raw binary output displayed on the LEDs.

Code and Explanation:

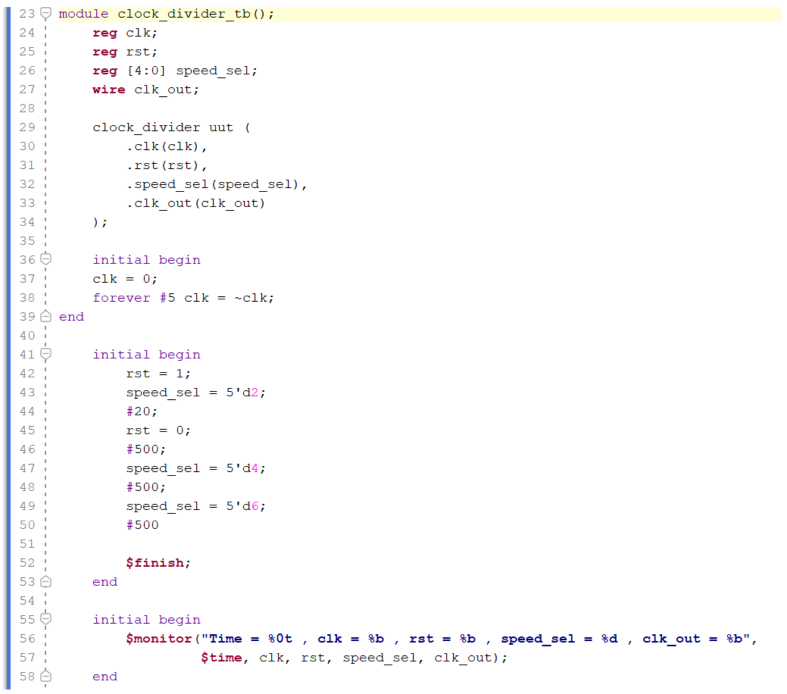
Clock_divider.v:



```
module clock_divider(
  input clk,
    input rst,
    input [4:0] speed_sel,
    output clk_out
);
    reg [31:0] count = 32'b0;

     always @(posedge clk) begin
        if (rst)
            count <= 32'd0;
        else
            count <= count + 1;
    end

    assign clk_out = count[speed_sel];  // Output bit-slice as divided clock
endmodule
```

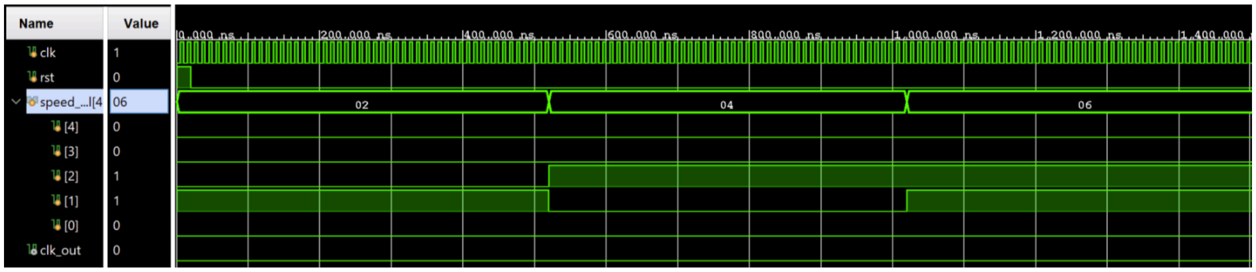
A clock is divided based on the inputted frequency working off a 32 bit counter that increments every clock cycle and resets (rst). Clk_out outputs a specific bit from the counter. The higher the bit, the slower the cycle.

Clock_divider_Tb:



```
module clock_divider_tb();
    reg clk;
    reg rst;
    reg [4:0] speed_sel;
    wire clk_out;

    clock_divider uut (
        .clk(clk),
        .rst(rst),
        .speed_sel(speed_sel),
        .clk_out(clk_out)
    );

    initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

    initial begin
        rst = 1;
        speed_sel = 5'd2;
        #20;
        rst = 0;
        #500;
        speed_sel = 5'd4;
        #500;
        speed_sel = 5'd6;
        #500

        $finish;
    end

    initial begin
        $monitor("Time = %0t , clk = %b , rst = %b , speed_sel = %d , clk_out = %b",
                 $time, clk, rst, speed_sel, clk_out);
    end
```

The Tesbench ensures different clock speeds are running by generating a toggling clock, then applying a rest and changing the speeds.



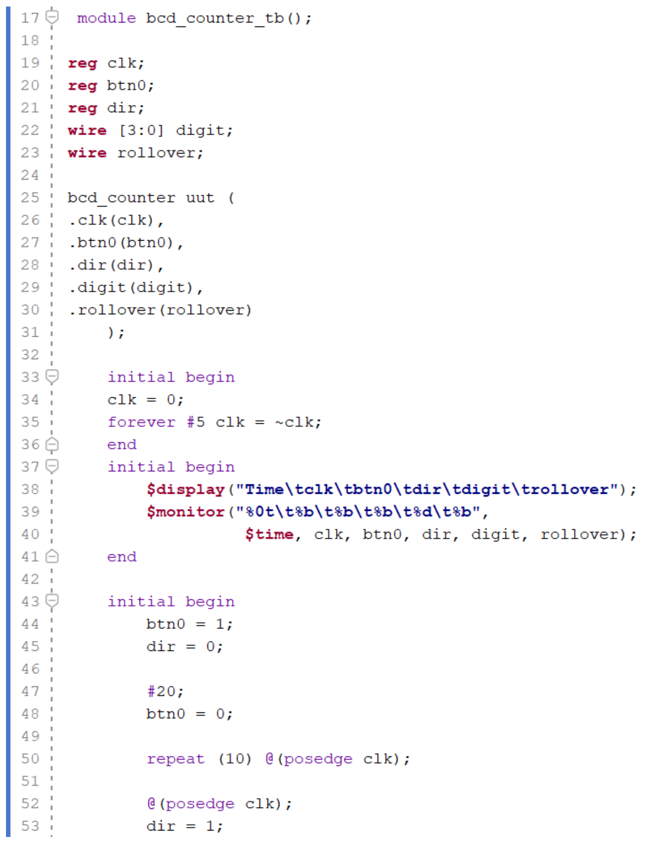
```
Time resolution is 1 ps
Time = 0 , ctrl_in = 0000 , ctrl_out = 0000
Time = 10000 , ctrl_in = 0001 , ctrl_out = 0001
Time = 20000 , ctrl_in = 0010 , ctrl_out = 0010
Time = 30000 , ctrl_in = 0011 , ctrl_out = 0011
Time = 40000 , ctrl_in = 0100 , ctrl_out = 0100
Time = 50000 , ctrl_in = 0101 , ctrl_out = 0101
Time = 60000 , ctrl_in = 0110 , ctrl_out = 0110
Time = 70000 , ctrl_in = 0111 , ctrl_out = 0111
Time = 80000 , ctrl_in = 1000 , ctrl_out = 1000
Time = 90000 , ctrl_in = 1001 , ctrl_out = 1001
$finish called at time : 100 ns : File "C:/Users/Sean/Documents/Summer 25 CPP/ECE3300L/Lab 6 - 1/Lab 6 - 1.srcs/sim_1/new/control_decoder_tb.v" Line 59
```

BCD_counter.v:

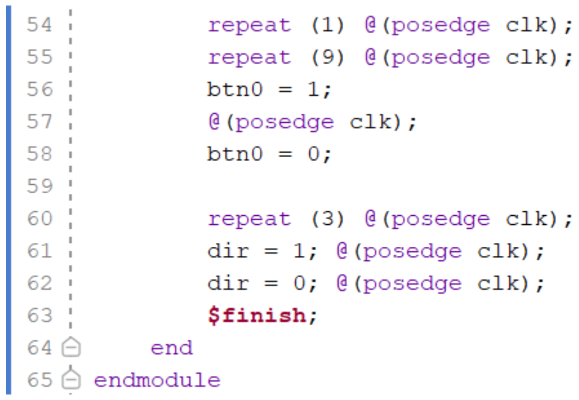
```
module bcd_counter(
    input clk,
    input btn0,
    input dir,
    output reg [3:0] digit,
    output reg rollover
);
    always @(posedge clk or posedge btn0) begin
        if (btn0) begin
            digit <= 4'd0;
            rollover <= 0;
        end else begin
            if (!dir) begin // up
                if (digit == 9) begin
                    digit <= 0;
                    rollover <= 1;
                end else begin
                    digit <= digit + 1;
                    rollover <= 0;
                end
            end
            else begin // down
                if (digit == 0) begin
                    digit <= 9;
                    rollover <= 1;
                end
                else begin
                    digit <= digit - 1;
                    rollover <= 0;
                end
            end
        end
    end
endmodule
```

Based on the dir signal, the positive edge btn0 and clk resets if btn0 is high and if dir=0 it counts up . It increments to 9 then rolls over. If dir=1, it decrements and wraps around 9 with rollover=1.

BCD_Counter_Tb:

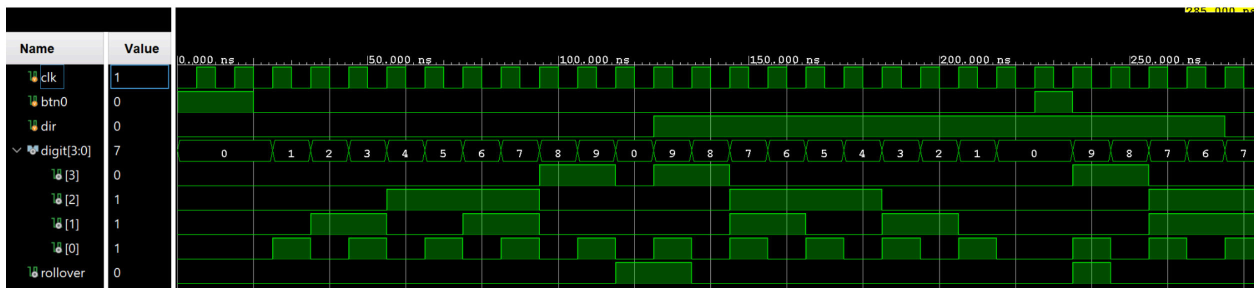


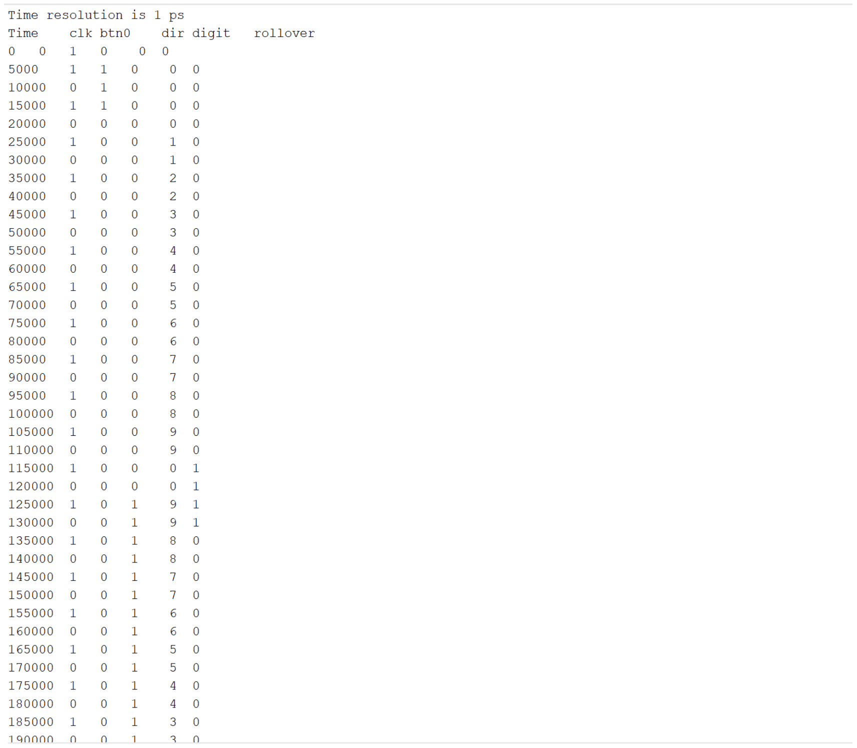
```
 module bcd_counter_tb();

reg clk;
reg btn0;
reg dir;
wire [3:0] digit;
wire rollover;

bcd_counter uut (
.clk(clk),
.btn0(btn0),
.dir(dir),
.digit(digit),
.rollover(rollover)
    );

    initial begin
    clk = 0;
    forever #5 clk = ~clk;
    end
    initial begin
        $display("Time\tclk\tbtn0\tdir\tdigit\trollover");
        $monitor("%0t\t%b\t%b\t%b\t%d\t%b",
                  $time, clk, btn0, dir, digit, rollover);
    end

    initial begin
        btn0 = 1;
        dir = 0;

        #20;
        btn0 = 0;

        repeat (10) @(posedge clk);

        @(posedge clk);
        dir = 1;
```



```
        repeat (1) @(posedge clk);
        repeat (9) @(posedge clk);
        btn0 = 1;
        @(posedge clk);
        btn0 = 0;

        repeat (3) @(posedge clk);
        dir = 1; @(posedge clk);
        dir = 0; @(posedge clk);
        $finish;
    end
endmodule
```

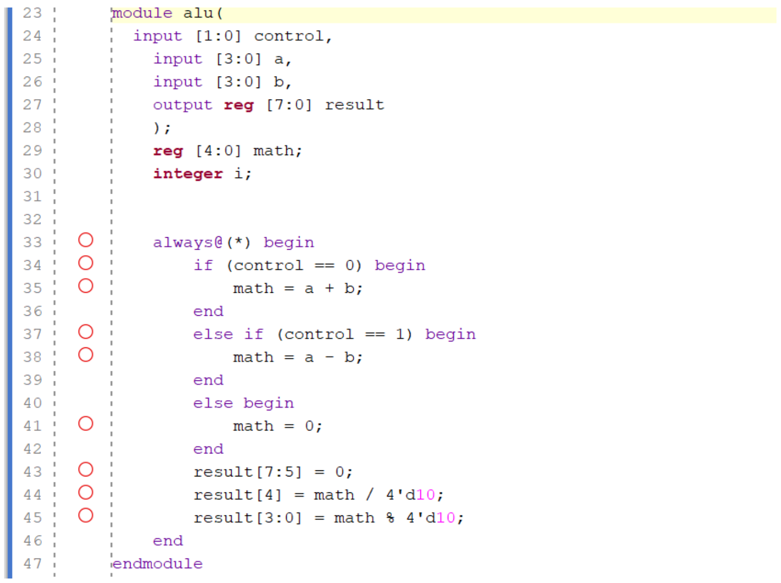
A 10ns clock period is created to mimic counting up 0 to 9 and then 9 to 0. There is an initial reset with btn0 then resets again at 3.





```
Time resolution is 1 ps
Time     clk btn0    dir digit    rollover
0   0    1   0    0  0
5000     1   1   0    0  0
10000    0   1   0    0  0
15000    1   1   0    0  0
20000    0   0   0    0  0
25000    1   0   0    1  0
30000    0   0   0    1  0
35000    1   0   0    2  0
40000    0   0   0    2  0
45000    1   0   0    3  0
50000    0   0   0    3  0
55000    1   0   0    4  0
60000    0   0   0    4  0
65000    1   0   0    5  0
70000    0   0   0    5  0
75000    1   0   0    6  0
80000    0   0   0    6  0
85000    1   0   0    7  0
90000    0   0   0    7  0
95000    1   0   0    8  0
100000   0   0   0    8  0
105000   1   0   0    9  0
110000   0   0   0    9  0
115000   1   0   0    0  1
120000   0   0   0    0  1
125000   1   0   1    9  1
130000   0   0   1    9  1
135000   1   0   1    8  0
140000   0   0   1    8  0
145000   1   0   1    7  0
150000   0   0   1    7  0
155000   1   0   1    6  0
160000   0   0   1    6  0
165000   1   0   1    5  0
170000   0   0   1    5  0
175000   1   0   1    4  0
180000   0   0   1    4  0
185000   1   0   1    3  0
190000   0   0   1    3  0
```

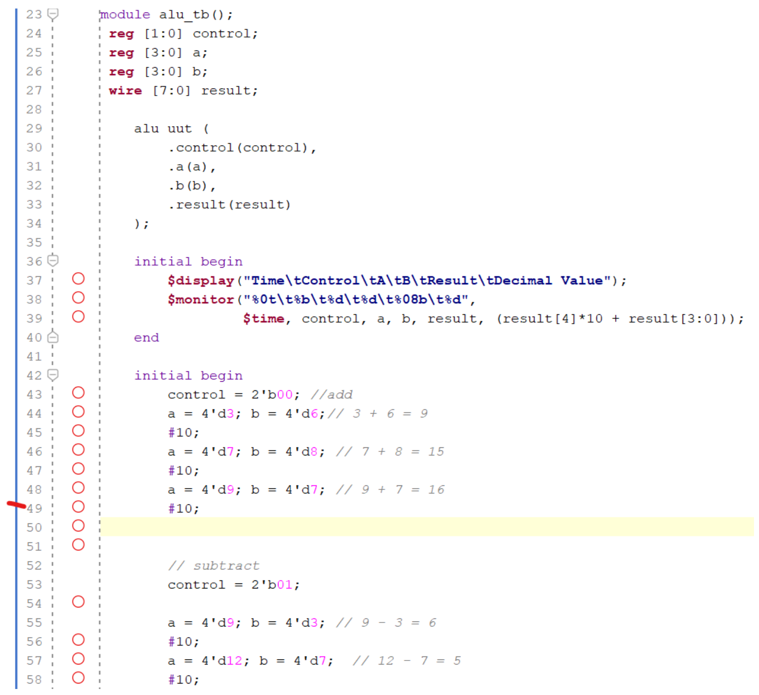
<u>Alu.v:</u>



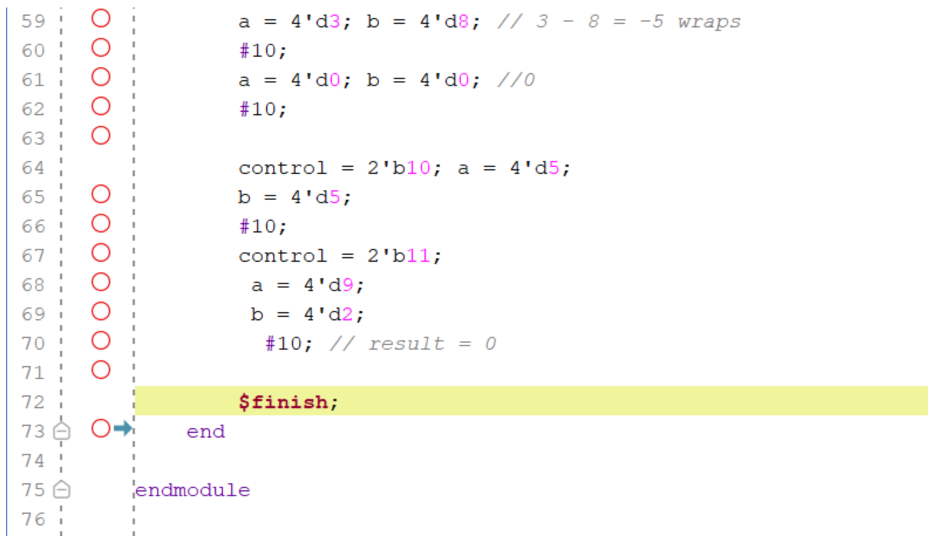
```
module alu(
 input [1:0] control,
   input [3:0] a,
   input [3:0] b,
   output reg [7:0] result
   );
   reg [4:0] math;
   integer i;


   always@(*) begin
       if (control == 0) begin
           math = a + b;
       end
       else if (control == 1) begin
           math = a - b;
       end
       else begin
           math = 0;
       end
       result[7:5] = 0;
       result[4] = math / 4'd10;
       result[3:0] = math % 4'd10;
   end
endmodule
```

The Alu performs the addition and subtraction based on the 2 bit control. It takes in two 4bit numbers for a and b to add or subtract, then stores the answer in math. The result is 8 bits and outputs the 2-bit encoding for the BCD.

<u>Alu_Tb:</u>

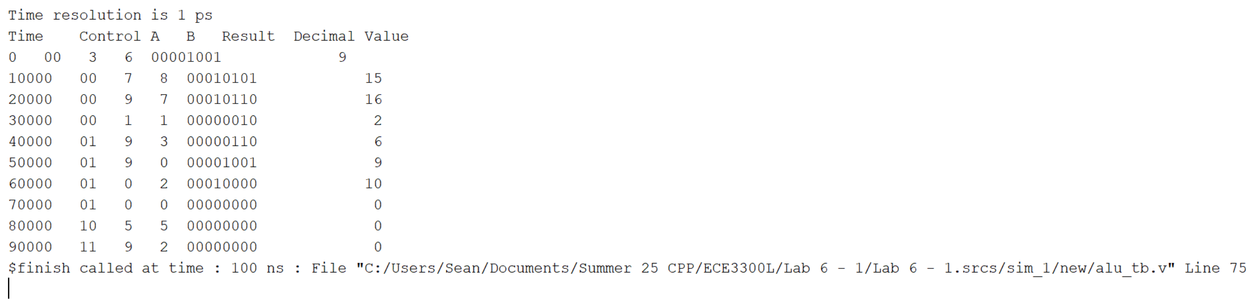


```verilog
module alu_tb();
 reg [1:0] control;
 reg [3:0] a;
 reg [3:0] b;
 wire [7:0] result;

    alu uut (
        .control(control),
        .a(a),
        .b(b),
        .result(result)
    );

    initial begin
        $display("Time\tControl\tA\tB\tResult\tDecimal Value");
        $monitor("%0t\t%b\t%d\t%d\t%08b\t%d",
                 $time, control, a, b, result, (result[4]*10 + result[3:0]));
    end

    initial begin
        control = 2'b00; //add
        a = 4'd3; b = 4'd6;// 3 + 6 = 9
        #10;
        a = 4'd7; b = 4'd8; // 7 + 8 = 15
        #10;
        a = 4'd9; b = 4'd7; // 9 + 7 = 16
        #10;


        // subtract
        control = 2'b01;

        a = 4'd9; b = 4'd3; // 9 - 3 = 6
        #10;
        a = 4'd12; b = 4'd7;  // 12 - 7 = 5
        #10;
```



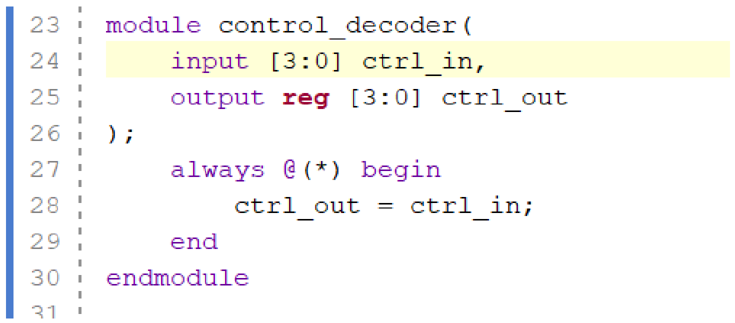
```verilog
        a = 4'd3; b = 4'd8; // 3 - 8 = -5 wraps
        #10;
        a = 4'd0; b = 4'd0; //0
        #10;

        control = 2'b10; a = 4'd5;
        b = 4'd5;
        #10;
        control = 2'b11;
         a = 4'd9;
         b = 4'd2;
          #10; // result = 0

        $finish;
    end

endmodule
```

The test bench runs through different addition and subtraction combinations to ensure the proper answers are being outputted.



```
Time resolution is 1 ps
Time     Control A    B    Result   Decimal Value
0    00    3    6   00001001          9
10000    00    7    8   00010101          15
20000    00    9    7   00010110          16
30000    00    1    1   00000010           2
40000    01    9    3   00000110           6
50000    01    9    0   00001001           9
60000    01    0    2   00010000          10
70000    01    0    0   00000000           0
80000    10    5    5   00000000           0
90000    11    9    2   00000000           0
$finish called at time : 100 ns : File "C:/Users/Sean/Documents/Summer 25 CPP/ECE3300L/Lab 6 - 1/Lab 6 - 1.srcs/sim_1/new/alu_tb.v" Line 75
```

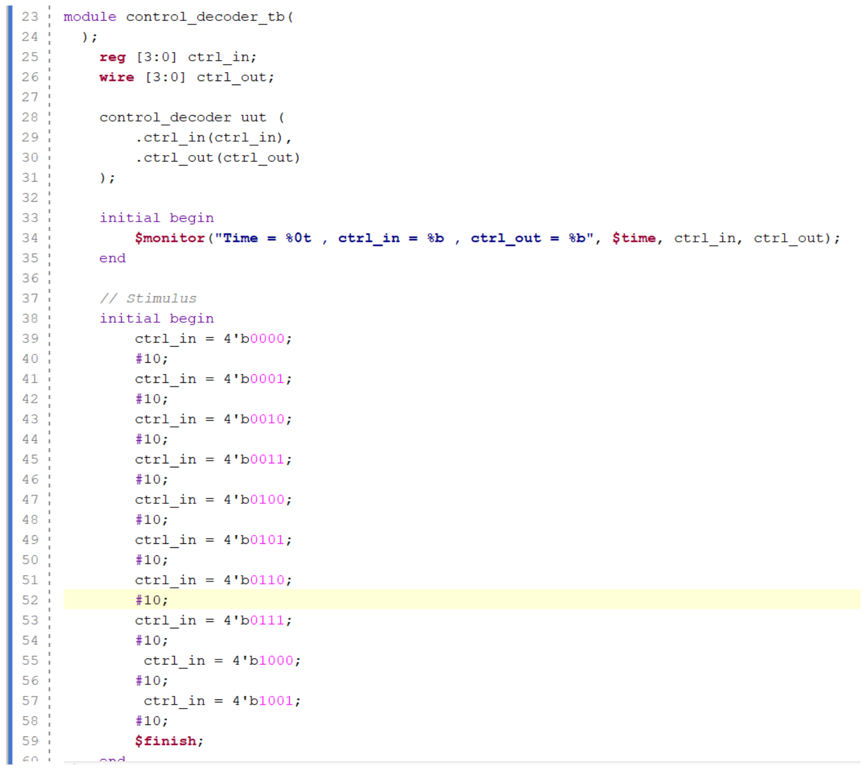
Control_decoder.v:



```verilog
module control_decoder(
    input [3:0] ctrl_in,
    output reg [3:0] ctrl_out
);
    always @(*) begin
        ctrl_out = ctrl_in;
    end
endmodule
```

This directly copies the 4-bit input to the output to demonstrate the switch inputs.

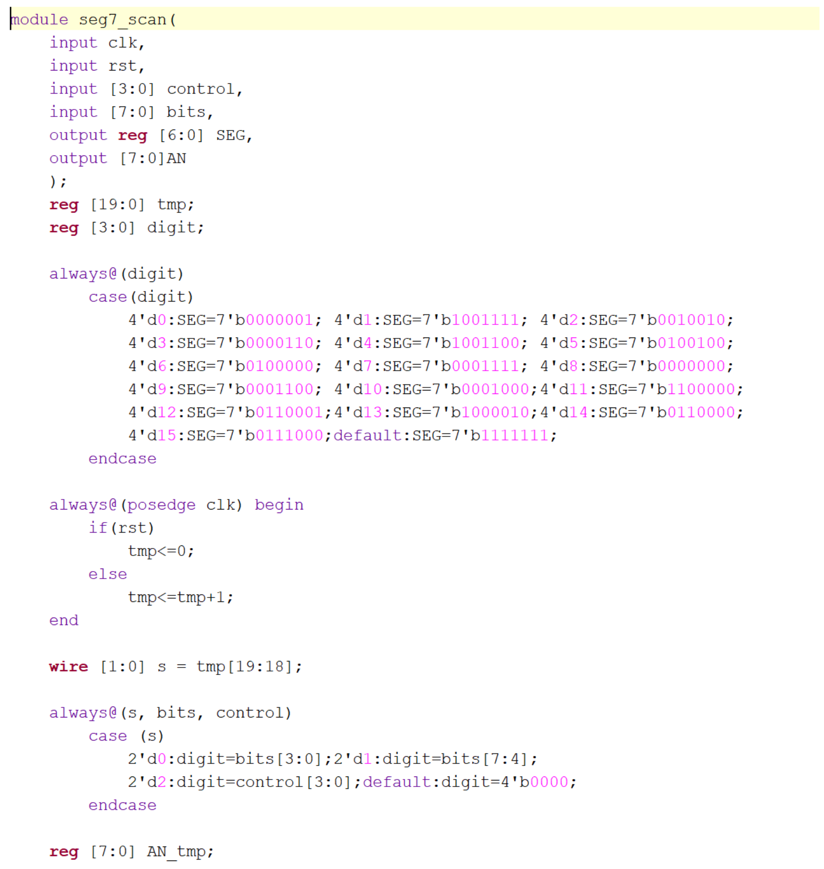
<u>Control_decoder.v:</u>



```verilog
module control_decoder_tb(
  );
    reg [3:0] ctrl_in;
    wire [3:0] ctrl_out;

    control_decoder uut (
        .ctrl_in(ctrl_in),
        .ctrl_out(ctrl_out)
    );

    initial begin
        $monitor("Time = %0t , ctrl_in = %b , ctrl_out = %b", $time, ctrl_in, ctrl_out);
    end

    // Stimulus
    initial begin
        ctrl_in = 4'b0000;
        #10;
        ctrl_in = 4'b0001;
        #10;
        ctrl_in = 4'b0010;
        #10;
        ctrl_in = 4'b0011;
        #10;
        ctrl_in = 4'b0100;
        #10;
        ctrl_in = 4'b0101;
        #10;
        ctrl_in = 4'b0110;
        #10;
        ctrl_in = 4'b0111;
        #10;
         ctrl_in = 4'b1000;
        #10;
         ctrl_in = 4'b1001;
        #10;
        $finish;
    end
```

4-bit binary sequences are used to confirm that the output is proper and the same as the input.

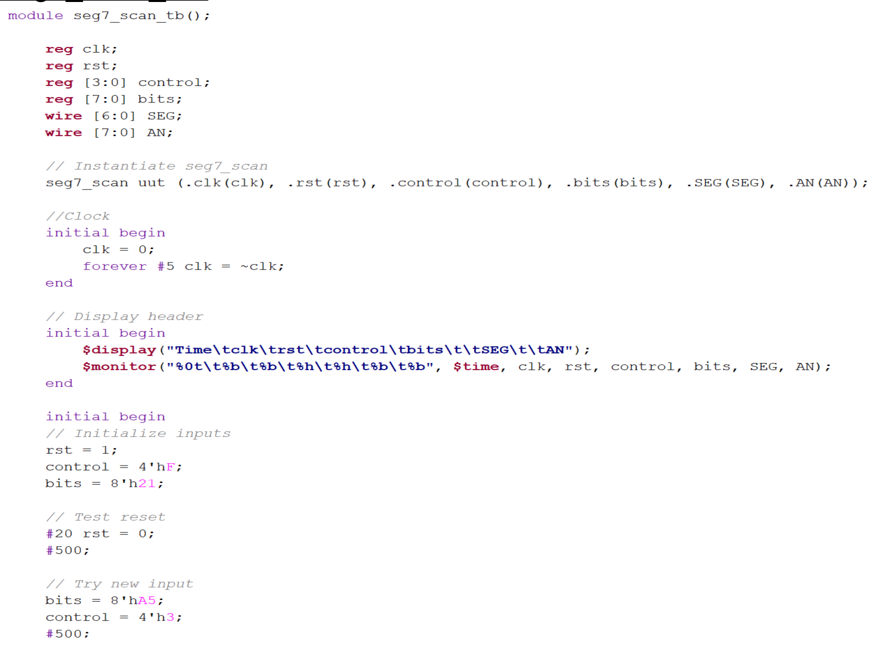
<u>Seg7_Scan.v:</u>



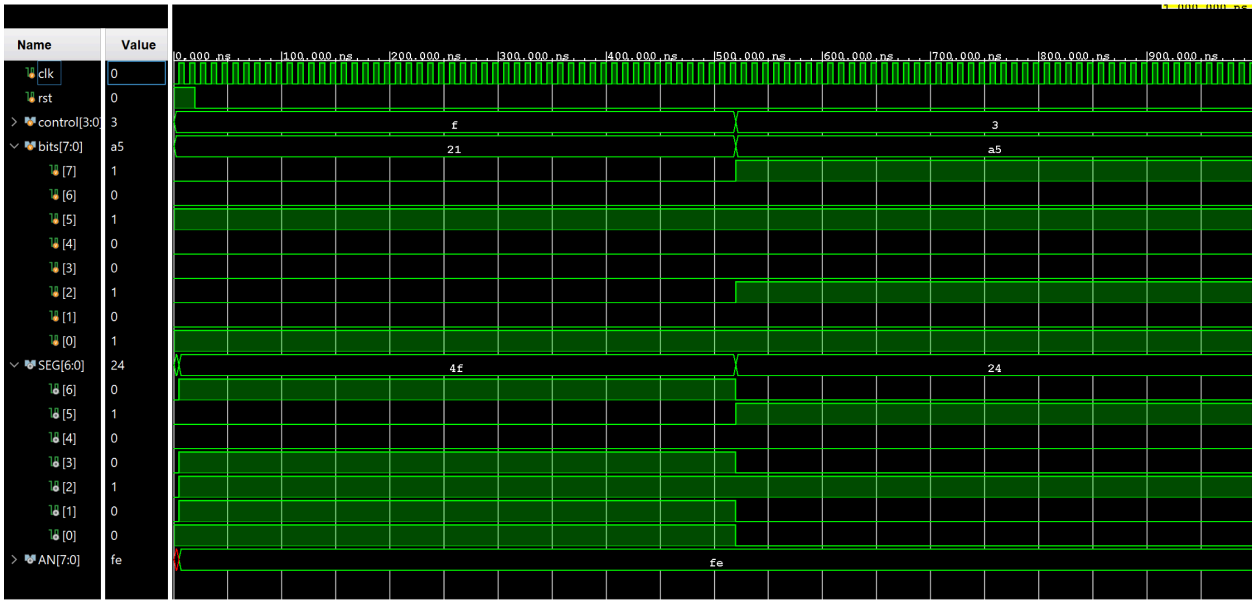
```
module seg7_scan(
    input clk,
    input rst,
    input [3:0] control,
    input [7:0] bits,
    output reg [6:0] SEG,
    output [7:0]AN
    );
    reg [19:0] tmp;
    reg [3:0] digit;

    always@(digit)
        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; 4'd10:SEG=7'b0001000;4'd11:SEG=7'b1100000;
            4'd12:SEG=7'b0110001;4'd13:SEG=7'b1000010;4'd14:SEG=7'b0110000;
            4'd15:SEG=7'b0111000;default:SEG=7'b1111111;
        endcase

    always@(posedge clk) begin
        if(rst)
            tmp<=0;
        else
            tmp<=tmp+1;
    end

    wire [1:0] s = tmp[19:18];

    always@(s, bits, control)
        case (s)
            2'd0:digit=bits[3:0];2'd1:digit=bits[7:4];
            2'd2:digit=control[3:0];default:digit=4'b0000;
        endcase

    reg [7:0] AN_tmp;
```

The scan has all the case statements for the 7 segment displays to properly light all numbers. Based on the edge of the run the tmp would be set to 0 or increment to determine whether to apply case s or not.

Seg7_Scan_Tb:



```
module seg7_scan_tb();

    reg clk;
    reg rst;
    reg [3:0] control;
    reg [7:0] bits;
    wire [6:0] SEG;
    wire [7:0] AN;

    // Instantiate seg7_scan
    seg7_scan uut (.clk(clk), .rst(rst), .control(control), .bits(bits), .SEG(SEG), .AN(AN));

    //Clock
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Display header
    initial begin
        $display("Time\tclk\trst\tcontrol\tbits\t\tSEG\t\tAN");
        $monitor("%0t\t%b\t%b\t%h\t%h\t%b\t%b", $time, clk, rst, control, bits, SEG, AN);
    end

    initial begin
    // Initialize inputs
    rst = 1;
    control = 4'hF;
    bits = 8'h21;

    // Test reset
    #20 rst = 0;
    #500;

    // Try new input
    bits = 8'hA5;
    control = 4'h3;
    #500;
```



```
Time resolution is 1 ps
Time     clk  rst  control  bits         SEG      AN
0    0    1    f    21   0000001  xxxxxxxx
5000     1    1    f    21   1001111  11111110
10000    0    1    f    21   1001111  11111110
15000    1    1    f    21   1001111  11111110
20000    0    0    f    21   1001111  11111110
25000    1    0    f    21   1001111  11111110
30000    0    0    f    21   1001111  11111110
35000    1    0    f    21   1001111  11111110
40000    0    0    f    21   1001111  11111110
45000    1    0    f    21   1001111  11111110
50000    0    0    f    21   1001111  11111110
55000    1    0    f    21   1001111  11111110
60000    0    0    f    21   1001111  11111110
65000    1    0    f    21   1001111  11111110
70000    0    0    f    21   1001111  11111110
75000    1    0    f    21   1001111  11111110
80000    0    0    f    21   1001111  11111110
85000    1    0    f    21   1001111  11111110
90000    0    0    f    21   1001111  11111110
95000    1    0    f    21   1001111  11111110
100000   0    0    f    21   1001111  11111110
105000   1    0    f    21   1001111  11111110
110000   0    0    f    21   1001111  11111110
115000   1    0    f    21   1001111  11111110
120000   0    0    f    21   1001111  11111110
125000   1    0    f    21   1001111  11111110
130000   0    0    f    21   1001111  11111110
135000   1    0    f    21   1001111  11111110
```

Top_lab6:

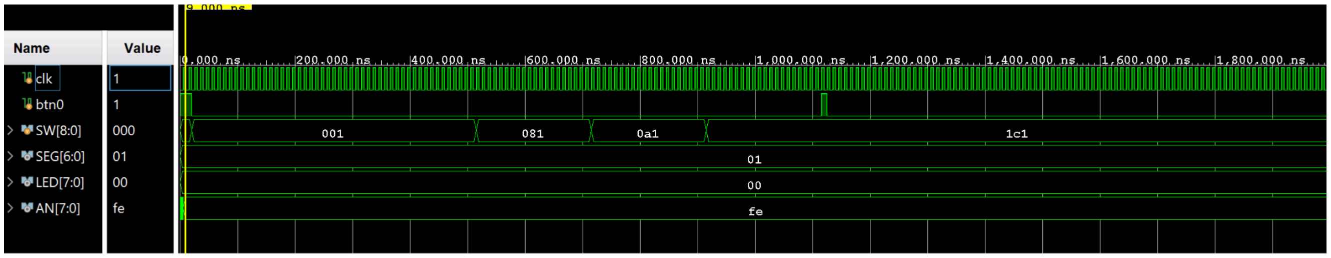
```
module top_lab6(
    input [8:0] SW,
    input clk,
    input btn0,
    output [6:0] SEG,
    output [7:0] LED,
    output [7:0] AN
    );

    wire clk_div;
    wire [3:0] ones_digit, tens_digit, ctrl_nibble;
    wire rollover_ones, rollover_tens;
    wire [7:0] result;

    //instantiates clock_divder which contains both the clock divison and mux
    clock_divider clock(.clk(clk), .rst(btn0), .speed_sel(SW[4:0]), .clk_out(clk_div));

    //instatiates bcd_counter which are single digit binary coded decimal counters for each digit
    bcd_counter ones_counter(.clk(clk_div), .btn0(btn0), .dir(SW[7]), .digit(ones_digit), .rollover(rollover_ones));
    bcd_counter tens_counter(.clk(clk_div), .btn0(btn0), .dir(SW[8]), .digit(tens_digit), .rollover(rollover_tens));

    //instantiates the arithmetic logic unit which handles addition and subtraction
    //control as well as result mapping and underflow (0 to 18)
    alu alu_unit(.a(tens_digit), .b(ones_digit), .control(SW[6:5]), .result(result));

    //instantiates control_decoder to display the control settings on the left most 7-segment display
    control_decoder ctrl_dec(.ctrl_in({SW[8], SW[7], SW[6], SW[5]}), .ctrl_out(ctrl_nibble));

    //instantiates seg7_scan display the result of the ALU operation on the 7-segment displays
    seg7_scan scan_inst(.clk(clk), .rst(btn0), .bits(result), .control(ctrl_nibble), .SEG(SEG), .AN(AN));

    //binds the value of the LEDs to each value imported to the ALU for debugging
    assign LED = {tens_digit[3:0], ones_digit[3:0]};

endmodule
```

The top lab as always wraps together all of our files to integrate them together.

Top_lab6_tb:

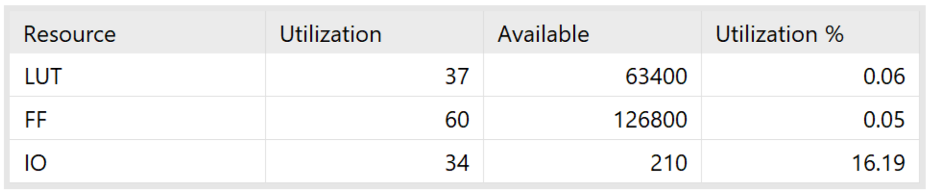
```
    // Clock 100MHz
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    // Monitoring
    initial begin
        $display("Time\tclk\tbtn0\tSW[8:0]\t\tLED\t\tSEG\t\tAN");
        $monitor("%0t\t%b\t%b\t%b\t%b\t%b\t%b", $time, clk, btn0, SW, LED, SEG, AN);
    end

        initial begin
        // Initial values
        btn0 = 1;           // Reset
        SW = 9'b000000000;  // All inputs zero

        #20 btn0 = 0;        // Reset off

        // Set a speed select
        SW[4:0] = 5'b00001; // Slowest speed

        // Count up on both counters
        SW[7] = 0; // dir for ones_counter = up
        SW[8] = 0; // dir for tens_counter = up

        // ALU mode
        SW[6:5] = 2'b00;

        // Counters increment
        repeat (50) @(posedge clk);

        // Change direction for ones_counter
        SW[7] = 1;
        repeat (20) @(posedge clk);
```

This test bench was designed to test the function of the main inputs and outputs of the build. The top level module is initialized then some test settings are input along with a clock. These settings allow us to see what the total output of the program will look like based on the input. This simple test is supplemented by the other test benches we built for each module.

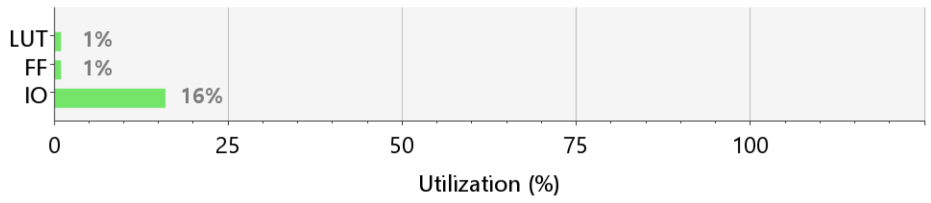


```
Time resolution is 1 ps
Time     clk btn0    SW[8:0]      LED      SEG     AN
0    0    1    000000000    00000000    0000001 xxxxxxxx
5000     1    1    000000000    00000000    0000001 11111110
10000    0    1    000000000    00000000    0000001 11111110
15000    1    1    000000000    00000000    0000001 11111110
20000    0    0    000000001    00000000    0000001 11111110
25000    1    0    000000001    00000000    0000001 11111110
30000    0    0    000000001    00000000    0000001 11111110
35000    1    0    000000001    00000000    0000001 11111110
40000    0    0    000000001    00000000    0000001 11111110
45000    1    0    000000001    00000000    0000001 11111110
50000    0    0    000000001    00000000    0000001 11111110
55000    1    0    000000001    00000000    0000001 11111110
60000    0    0    000000001    00000000    0000001 11111110
65000    1    0    000000001    00000000    0000001 11111110
70000    0    0    000000001    00000000    0000001 11111110
75000    1    0    000000001    00000000    0000001 11111110
80000    0    0    000000001    00000000    0000001 11111110
85000    1    0    000000001    00000000    0000001 11111110
90000    0    0    000000001    00000000    0000001 11111110
95000    1    0    000000001    00000000    0000001 11111110
100000   0    0    000000001    00000000    0000001 11111110
105000   1    0    000000001    00000000    0000001 11111110
110000   0    0    000000001    00000000    0000001 11111110
115000   1    0    000000001    00000000    0000001 11111110
```

Vivado (LUTs, FFs, Power):

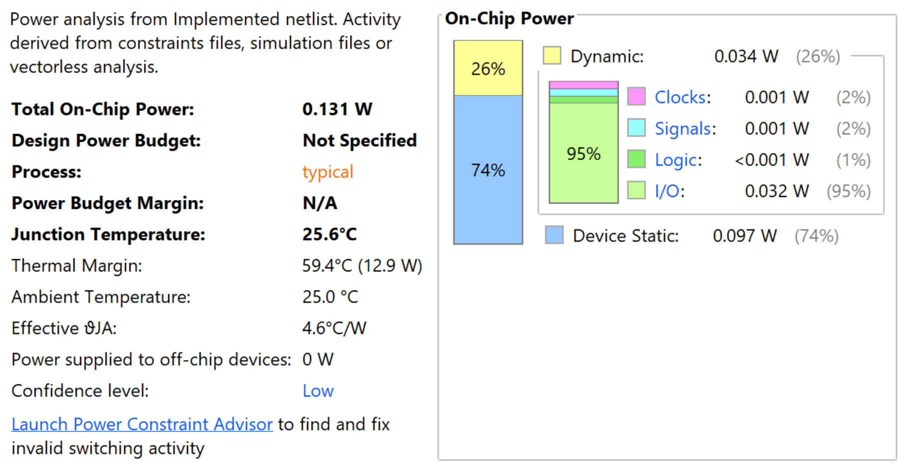


Resource	Utilization	Available	Utilization %
LUT	37	63400	0.06
FF	60	126800	0.05
IO	34	210	16.19



```
1. Slice Logic
--------------

+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs              |   36 |     0 |          0 |     63400 |  0.06 |
|   LUT as Logic          |   36 |     0 |          0 |     63400 |  0.06 |
|   LUT as Memory         |    0 |     0 |          0 |     19000 |  0.00 |
| Slice Registers         |   60 |     0 |          0 |    126800 |  0.05 |
|   Register as Flip Flop |   60 |     0 |          0 |    126800 |  0.05 |
|   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.
```

```
2. Slice Logic Distribution
---------------------------

+-------------------------------------------+------+-------+------------+-----------+-------+
|                 Site Type                 | Used | Fixed | Prohibited | Available | Util% |
+-------------------------------------------+------+-------+------------+-----------+-------+
| Slice                                     |   25 |     0 |          0 |     15850 |  0.16 |
|   SLICEL                                  |   17 |     0 |            |           |       |
|   SLICEM                                  |    8 |     0 |            |           |       |
| LUT as Logic                              |   36 |     0 |          0 |     63400 |  0.06 |
|   using O5 output only                    |    0 |       |            |           |       |
|   using O6 output only                    |   24 |       |            |           |       |
|   using O5 and O6                         |   12 |       |            |           |       |
| LUT as Memory                             |    0 |     0 |          0 |     19000 |  0.00 |
|   LUT as Distributed RAM                  |    0 |     0 |            |           |       |
|     using O5 output only                  |    0 |       |            |           |       |
|     using O6 output only                  |    0 |       |            |           |       |
|     using O5 and O6                       |    0 |       |            |           |       |
|   LUT as Shift Register                   |    0 |     0 |            |           |       |
|     using O5 output only                  |    0 |       |            |           |       |
|     using O6 output only                  |    0 |       |            |           |       |
|     using O5 and O6                       |    0 |       |            |           |       |
| Slice Registers                           |   60 |     0 |          0 |    126800 |  0.05 |
|   Register driven from within the Slice   |   60 |       |            |           |       |
|   Register driven from outside the Slice  |    0 |       |            |           |       |
| Unique Control Sets                       |    2 |       |          0 |     15850 |  0.01 |
+-------------------------------------------+------+-------+------------+-----------+-------+
* * Note: Available Control Sets calculated as Slice * 1, Review the Control Sets Report for more information regarding control sets.
```



Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.131 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.6°C**
Thermal Margin:	59.4°C (12.9 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	4.6°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

🟨	Dynamic:	0.034 W	(26%)
🟪	Clocks:	0.001 W	(2%)
🟦	Signals:	0.001 W	(2%)
🟩	Logic:	<0.001 W	(1%)
🟩	I/O:	0.032 W	(95%)
🟦	Device Static:	0.097 W	(74%)

```
1. Summary
----------

+--------------------------+--------------+
| Total On-Chip Power (W)  | 0.131        |
| Design Power Budget (W)  | Unspecified* |
| Power Budget Margin (W)  | NA           |
| Dynamic (W)              | 0.034        |
| 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 |      132 |       --- |             --- |
|   LUT as Logic  |    <0.001 |       36 |     63400 |            0.06 |
|   Register      |    <0.001 |       60 |    126800 |            0.05 |
|   CARRY4        |    <0.001 |       13 |     15850 |            0.08 |
|   F7/F8 Muxes   |    <0.001 |        4 |     63400 |           <0.01 |
|   Others        |     0.000 |        7 |       --- |             --- |
| Signals         |    <0.001 |      115 |       --- |             --- |
| I/O             |     0.032 |       34 |       210 |           16.19 |
| Static Power    |     0.097 |          |           |                 |
| Total           |     0.131 |          |           |                 |
+-----------------+-----------+----------+-----------+-----------------+
```

Video Link:
https://youtu.be/D2C_ej10In0?si=xFRM62qPlkh71soX

Reflections:
The ALU proved to be a struggle, especially when it came to the display. Converting the addition and subtraction into two separate displays became an issue, specifically at the number 18. At 18 we needed it to correctly display on the BCD and wrap around. The division and modulo had to break the numbers down carefully in order to properly display. Since we worked on the counter and divider last week, it was easier to build this week, along with the test benches for them. As the semester continues, the testbenches are progressively becoming easier to write as we sometimes struggle with them.

Partner Contributions:
Heba initially started the code, then Sean looked at her code and edited it to get the proper displays. The test benches were evenly split, each working back and forth with each other to get the right outputs on the FPGA and simulation. The lab report was collaboratively worked on, with each person working on different sections. The work was 50/50.