3300 Lab 6
Dual BCD Up/Down Counters, ALU, and Control Display on 7-Segment

Group C
Rohan Walia
Parsa Ghasemi
7/28/25

Code:

```
module top_lab6(
    input CLK,
    input BTN0,
    input [8:0] SW,
    output [7:0] LED,
    output [7:0] AN,
    output [6:0] SEG
);
    wire clk_div;
    wire [31:0] cnt;
    wire [3:0] unit_bcd, tens_bcd, ctrl_nibble;
    wire [7:0] alu_result;

    // Instantiate clock divider (slows 100 MHz clk)
    clock_divider u_div(
        .clk(CLK),
        .BTN0(!BTN0),
        .sel(SW[4:0]),
        .cnt(cnt),
        .clk_div(clk_div)
    );

    // Units BCD counter with direction control
    bcd_counter bcd_ones(
        .clk_div(clk_div),
        .BTN0(!BTN0),
        .dir_bit(SW[7]),
        .BCD(unit_bcd)
    );

    // Tens BCD counter with direction control
    bcd_counter bcd_tens(
        .clk_div(clk_div),
        .BTN0(!BTN0),
        .dir_bit(SW[8]),
        .BCD(tens_bcd)
    );

    // 4-bit ALU: add or subtract units and tens BCD
    alu alu1(
        .A(unit_bcd),
        .B(tens_bcd),
        .ctrl(SW[6:5]),
```

```
        .result(alu_result)
    );

    // Control nibble register for display
    control_decoder dec(
        .clk_div(clk_div),
        .BTN0(!BTN0),
        .SW(SW[8:5]),
        .ctrl_nibble(ctrl_nibble)
    );

    // Convert ALU 8-bit result to unsigned decimal digits for 7-seg display
    wire [7:0] abs_result;
    wire [3:0] alu_units;
    wire [3:0] alu_tens;

    assign abs_result = (alu_result[7]) ? (~alu_result + 1) : alu_result;

    assign alu_tens  = (abs_result / 10) % 10;
    assign alu_units = abs_result % 10;

    seg7_scan u_scan(
        .clk(CLK),
        .BTN0(!BTN0),
        .digit0(alu_units),    // units digit decimal
        .digit1(alu_tens),     // tens digit decimal
        .digit2(ctrl_nibble),  // control nibble hex
        .SEG(SEG),
        .AN(AN)
    );

    assign LED = {tens_bcd, unit_bcd};

endmodule
```

```
module clock_divider(
    input clk,
    input BTN0,
    input [4:0] sel,
    output reg [31:0] cnt,
    output clk_div
);
    always @(posedge clk or negedge BTN0) begin
        if (!BTN0) cnt <= 32'd0;
        else cnt <= cnt + 1;
    end

    assign clk_div = cnt[sel];
endmodule
```

```
module bcd_counter(
    input clk_div,
    input BTN0,
    input dir_bit,
    output reg [3:0] BCD
);
    always @(posedge clk_div or negedge BTN0) begin
        if (!BTN0) BCD <= 4'd0;
        else if (dir_bit) BCD <= (BCD == 4'd9) ? 4'd0 : BCD + 4'd1;
        else BCD <= (BCD == 4'd0) ? 4'd9 : BCD - 4'd1;
    end
endmodule
```

```
module alu(
    input [3:0] A, B,
    input [1:0] ctrl,
    output reg [7:0] result
);
    always @(*) begin
        case (ctrl)
            2'b00: result = A + B;       // ADD
            2'b01: result = A - B;       // SUB
            2'b10: result = B - A;       // Reverse SUB
            2'b11: result = A + B + 4'd1;
            default: result = 8'd0;
        endcase
    end
endmodule
```

```
module control_decoder(
    input clk_div,
    input BTN0,
    input [3:0] SW,
    output reg [3:0] ctrl_nibble
);
    always @(posedge clk_div or negedge BTN0) begin
        if (!BTN0) ctrl_nibble <= 4'd0;
        else ctrl_nibble <= SW;
    end
endmodule
```

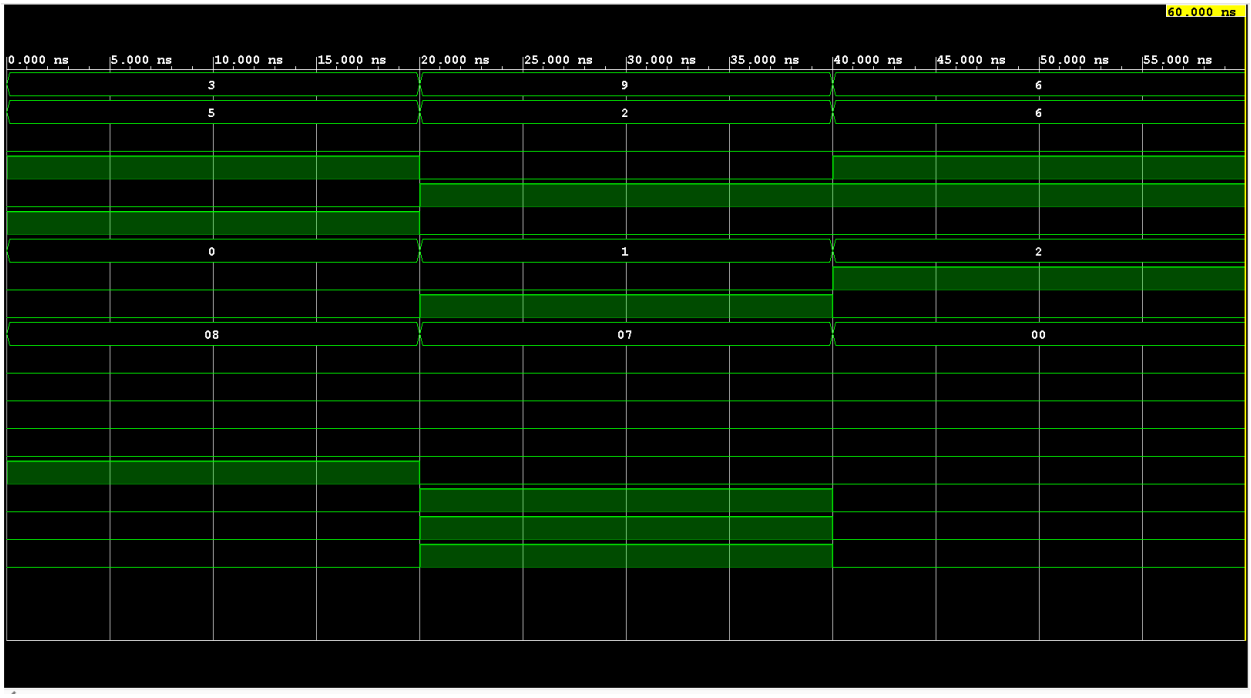
```
module seg7_scan(
    input clk,
    input BTN0,
    input [3:0] digit0, digit1, digit2,
    output reg [6:0] SEG,
    output reg [7:0] AN
);
    reg [19:0] refresh_counter;

    always @(posedge clk or negedge BTN0) begin
        if (!BTN0) refresh_counter <= 20'd0;
        else refresh_counter <= refresh_counter + 1;
    end

    wire [1:0] sel = refresh_counter[19:18];
    reg [3:0] current_digit;

    always @(*) begin
        case (sel)
            2'b00: begin
                AN = 8'b11111110;
                current_digit = digit0;
            end
            2'b01: begin
                AN = 8'b11111101;
                current_digit = digit1;
            end
            2'b10: begin
                AN = 8'b11111011;
                current_digit = digit2;
            end
            default: begin
                AN = 8'b11111111;
                current_digit = 4'd0;
            end
        endcase

        case (current_digit)
            4'h0: SEG = 7'b1000000;
            4'h1: SEG = 7'b1111001;
            4'h2: SEG = 7'b0100100;
            4'h3: SEG = 7'b0110000;
```

```
            4'h4: SEG = 7'b0011001;
            4'h5: SEG = 7'b0010010;
            4'h6: SEG = 7'b0000010;
            4'h7: SEG = 7'b1111000;
            4'h8: SEG = 7'b0000000;
            4'h9: SEG = 7'b0010000;
            4'hA: SEG = 7'b0001000;
            4'hB: SEG = 7'b0000011;
            4'hC: SEG = 7'b1000110;
            4'hD: SEG = 7'b0100001;
            4'hE: SEG = 7'b0000110;
            4'hF: SEG = 7'b0001110;
            default: SEG = 7'b1111111;
        endcase
    end
endmodule
```

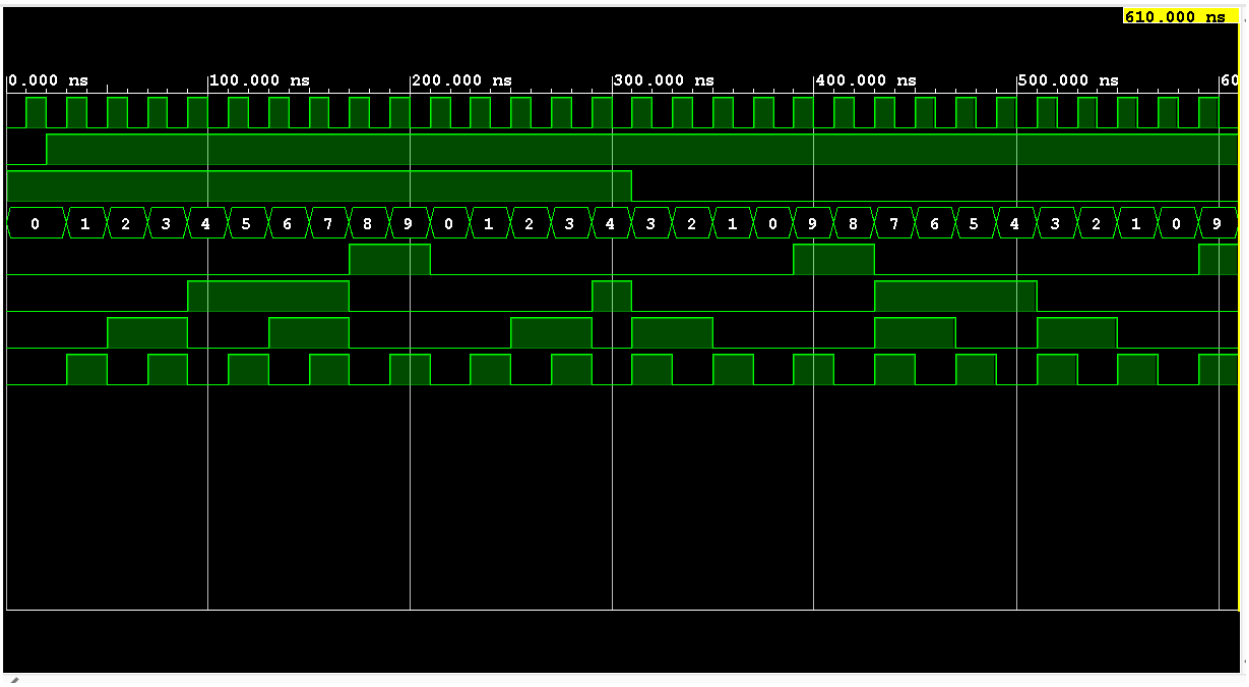
This project has two counters that go up or down based on switches SW7 and SW8, and their speed is controlled by switches SW[4:0]. The ALU takes the two counter values and either adds or subtracts them, depending on switches SW6 and SW5. The result from the ALU is split into digits and shown as a decimal number on the first two digits of the 7-segment display. The third digit shows the switch settings from SW[8:5] as a hex number, mainly for debugging. The LEDs light up to show the raw counter values so you can check they're working.

Waveform screenshot

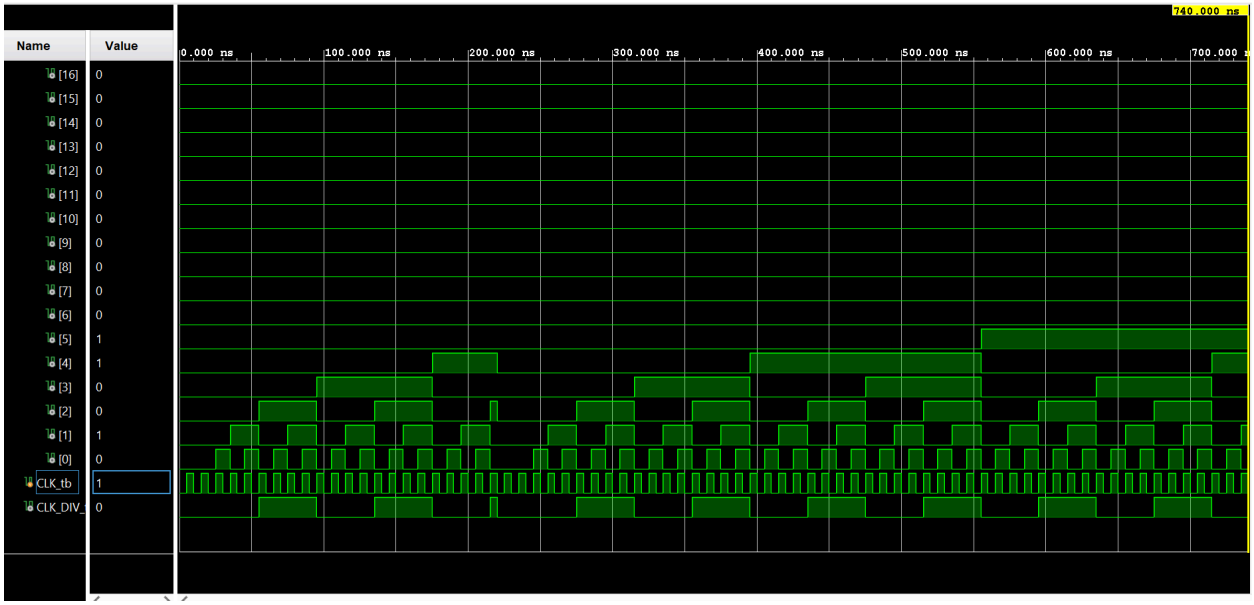
Alu



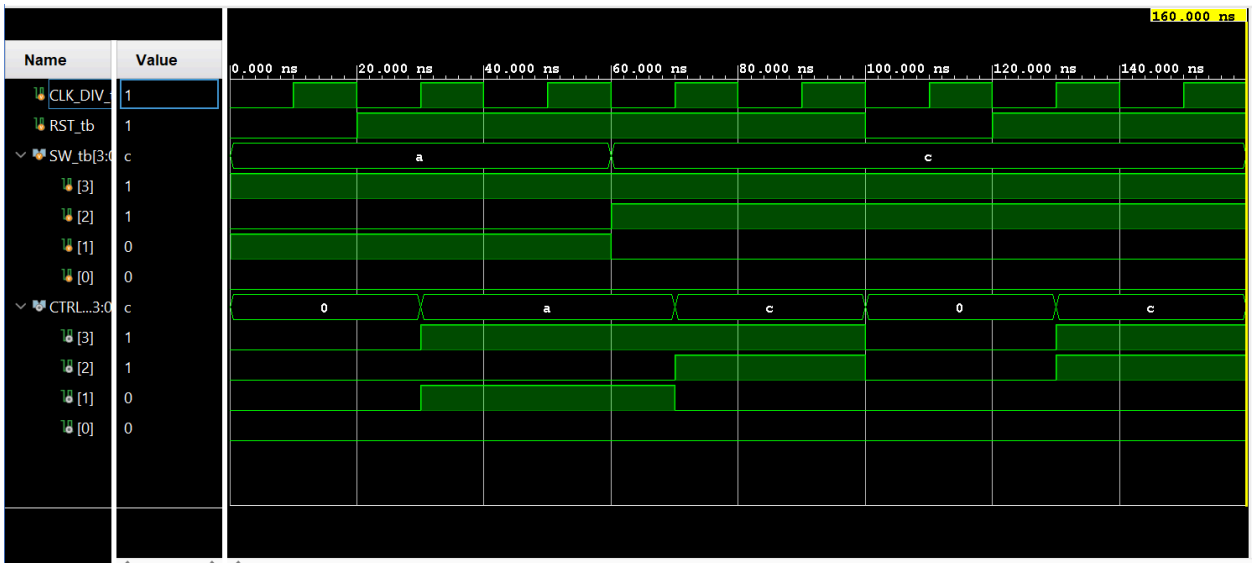
Bcd_counter



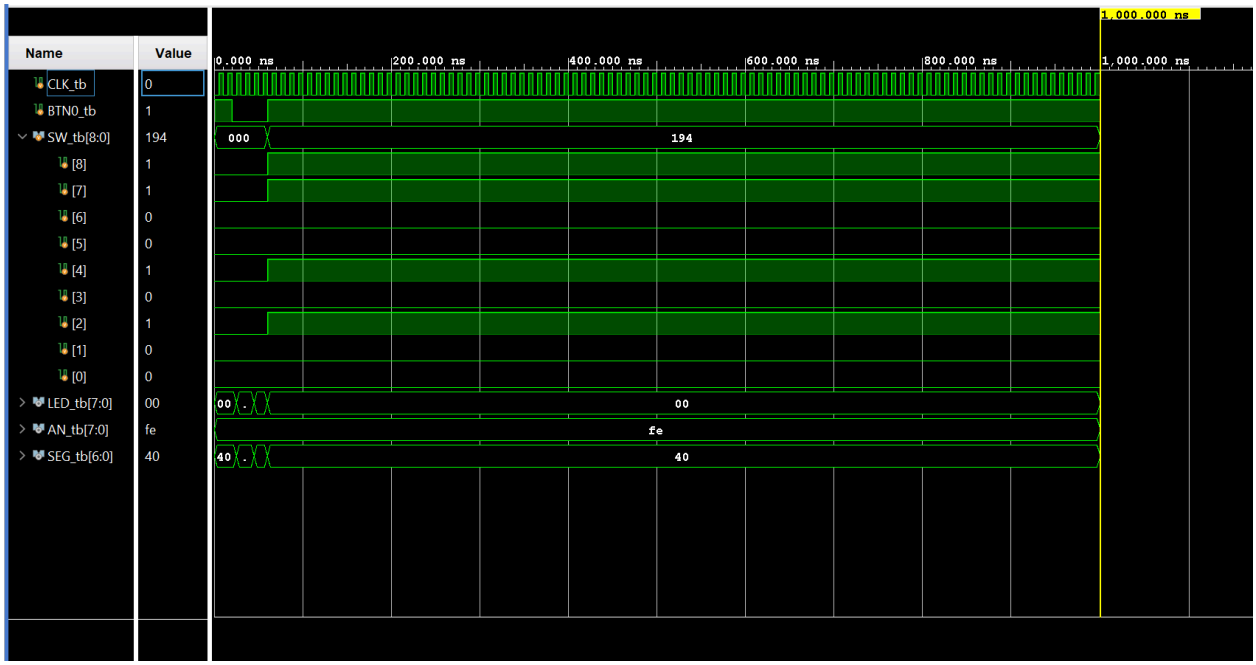
Clock_divider



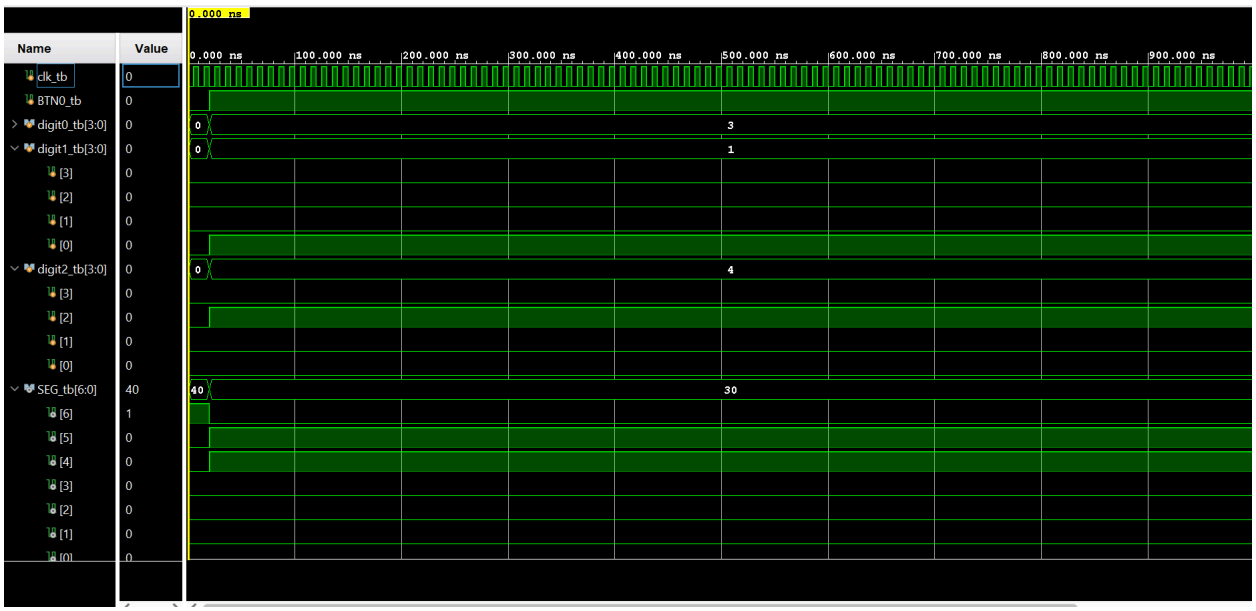
Control_decoder



Top

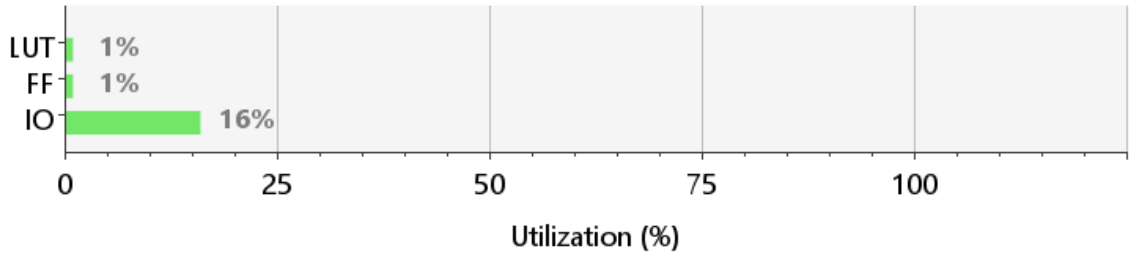

Name
Value
CLK_tb
0
BTN0_tb
1
SW_tb[8:0]
194
[8]
1
[7]
1
[6]
0
[5]
0
[4]
1
[3]
0
[2]
1
[1]
0
[0]
0
LED_tb[7:0]
00
AN_tb[7:0]
fe
SEG_tb[6:0]
40
0.000 ns
200.000 ns
400.000 ns
600.000 ns
800.000 ns
1,000.000 ns
000
194
00
fe
40

7seg


Name
Value
clk_tb
0
BTN0_tb
0
digit0_tb[3:0]
0
digit1_tb[3:0]
0
[3]
0
[2]
0
[1]
0
[0]
0
digit2_tb[3:0]
0
[3]
0
[2]
0
[1]
0
[0]
0
SEG_tb[6:0]
40
[6]
1
[5]
0
[4]
0
[3]
0
[2]
0
[1]
0
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
600.000 ns
700.000 ns
800.000 ns
900.000 ns
3
1
4
30

LUTs and FF screenshot:

Resource	Utilization	Available	Utilization %
LUT	34	63400	0.05
FF	64	126800	0.05
IO	34	210	16.19



Timing screenshot:

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	8.044 ns	Worst Hold Slack (WHS):	0.250 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	52	Total Number of Endpoints:	52	Total Number of Endpoints:	53

All user specified timing constraints are met.

Contribution:
Rohan Walia (50%): Implementation, demo, verilog, report
Parsa Ghasemi (50%): testbench code, testing, report

Reflections: This lab gave us hands-on experience designing and integrating BCD counters, an ALU, and a 7-segment display driver into a complete digital system. We learned the importance of clock division for timing control and how switch-based direction and operation inputs affect counter behavior and arithmetic processing.