ECE3300L Lab 6

Group B

By Faris Khan (ID #: 012621102)

AND

Nicholas Williams (ID#:016556982)

28 July 2025

Introduction

The goal of this lab was to design and implement a digital system on the Nexys A7 FPGA board that combines two independent BCD up/down counters, a 2-operation ALU (add/subtract), and a 3-digit 7-segment display system. The units and tens counters operate separately based on individual direction switches and are clocked by a speed-adjustable clock divider. Their BCD values are passed into a simple ALU, which performs either addition or subtraction based on control inputs. The output of the ALU is then split into decimal digits and displayed using a time-multiplexed 7-segment display, along with the control nibble for visual feedback. LEDs provide additional real-time binary debug output for both BCD counters.

Code

clock_divider.v

```
module clock_divider(
    input clk,                  // 100 MHz input clock
    input rst_n,                // active-low reset
    input [4:0] sel,            // speed-select from SW[4:0]
    output reg [31:0] count,    // counter
    output clk_div              // clock output
);
    //counter
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            count <= 32'd0;
        else
            count <= count + 1;
    end

    //select one of the bits from counter
    assign clk_div = count[sel];
endmodule
```

This module implements a 32-bit free-running counter using the system clock (100 MHz), and it includes an internal 32-to-1 multiplexer. The sel input (from SW[4:0]) chooses which bit of the counter to use as the divided clock signal (clk_div). Higher sel values result in slower output clock frequencies, allowing the user to control how fast the BCD counters update.

bcd_counter.v

```
module bcd_counter(
    input clk,
    input rst_n,
    input dir,
    output reg [3:0] bcd
);
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n)
            bcd <= 4'd0;
        else if (dir)
            bcd <= (bcd == 4'd9) ? 4'd0 : bcd + 1;
        else
            bcd <= (bcd == 4'd0) ? 4'd9 : bcd - 1;
    end
endmodule
```

This module is a 4-bit BCD (0–9) up/down counter. It increments or decrements based on a direction input (dir) and wraps around when it reaches the limits. Each counter can be independently controlled for counting direction and resets to 0 when BTN0 is pressed.

alu.v

```
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;
            2'b01: result = A - B;
            default: result = 8'd0;
        endcase
    end
endmodule
```

This module performs a basic 4-bit arithmetic operation based on a control input. It takes two 4-bit BCD values as inputs (from the units and tens counters), and performs either addition (A + B) or subtraction (A - B) depending on the value of ctrl. When subtracting, underflow is allowed to wrap naturally due to unsigned arithmetic, which can cause the result to roll over (e.g., 3 - 5 = 254). This behavior is intentional for the lab and demonstrates how digital subtraction behaves

control_decoder.v

```
module control_decoder(
    input [3:0] sw,
    output [3:0] ctrl_nibble
);
    assign ctrl_nibble = sw;
endmodule
```

This module takes 4 bits of control input from the switches (SW[8:5]) and outputs them as a single 4-bit nibble to be displayed on the 7-segment display. It helps verify that the control settings are active and being read correctly.

Seg7_scan.v

```
module seg7_scan(
    input clk,
    input rst_n,
    input [3:0] digit0, digit1, digit2,
    output reg [7:0] an,
    output reg [6:0] seg
);
    reg [19:0] scan_count;
    wire [1:0] sel;
    reg [3:0] current_digit;

    assign sel = scan_count[19:18];

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

    always @(*) begin
        case (sel)
            2'b00: begin
                an = 8'b1111_1110;    // AN0 active
                current_digit = digit0;
            end
            2'b01: begin
                an = 8'b1111_1101;    // AN1 active
                current_digit = digit1;
            end
            2'b10: begin
                an = 8'b1111_1011;    // AN2 active
                current_digit = digit2;
            end
            default: begin
                an = 8'b1111_1111;    // Disable all
                current_digit = 4'b0000;
            end
        endcase
    end

    always @(*) begin
        case (current_digit)
            4'd0:  seg = 7'b1000000; // 0
            4'd1:  seg = 7'b1111001; // 1
            4'd2:  seg = 7'b0100100; // 2
            4'd3:  seg = 7'b0110000; // 3
            4'd4:  seg = 7'b0011001; // 4
            4'd5:  seg = 7'b0010010; // 5
            4'd6:  seg = 7'b0000010; // 6
            4'd7:  seg = 7'b1111000; // 7
            4'd8:  seg = 7'b0000000; // 8
            4'd9:  seg = 7'b0010000; // 9
            4'd10: seg = 7'b0001000; // A
            4'd11: seg = 7'b0000011; // b
            4'd12: seg = 7'b1000110; // C
            4'd13: seg = 7'b0100001; // d
            4'd14: seg = 7'b0000110; // E
            4'd15: seg = 7'b0001110; // F
            default: seg = 7'b1111111; // Blank
        endcase
```

This module controls the 3-digit 7-segment display using time-multiplexing. It cycles quickly through the three digits — result units, result tens, and control nibble — and shows the correct value on each. Only one digit is lit at a time, but the cycling is fast enough that all digits appear lit.

top_lab6.v

```
module top_lab6(
    input CLK,
    input [8:0] SW,       // SW[8]: tens dir, SW[7]: units dir, SW[6:5]: ALU, SW[4:0]: speed
    input BTN0,
    output [6:0] SEG,
    output [7:0] AN,
    output [7:0] LED
);
    wire rst_n = ~BTN0;
    wire [31:0] count;
    wire clk_div;
    wire [3:0] units, tens;
    wire [7:0] result;
    wire [3:0] ctrl_nibble;

    // ALU display result
    reg [3:0] result_units, result_tens;

    // Clock divider with mux
    clock_divider u_clk (
        .clk(CLK),
        .rst_n(rst_n),
        .sel(SW[4:0]),
        .count(count),
        .clk_div(clk_div)
    );

    // Counters
    bcd_counter u_counter (.clk(clk_div), .rst_n(rst_n), .dir(SW[7]), .bcd(units));
    bcd_counter t_counter (.clk(clk_div), .rst_n(rst_n), .dir(SW[8]), .bcd(tens));

    // ALU logic
    alu u_alu (.A(tens), .B(units), .ctrl(SW[6:5]), .result(result));

    // split into tens and units
    always @(*) begin
        result_units = result % 10;
        result_tens  = result / 10;
    end

    // Control nibble display
    control_decoder u_ctrl (.sw(SW[8:5]), .ctrl_nibble(ctrl_nibble));

    // 7-segment scanner
    seg7_scan u_seg (
        .clk(CLK), .rst_n(rst_n),
        .digit0(result_units),
        .digit1(result_tens),
        .digit2(ctrl_nibble),
        .an(AN), .seg(SEG)
    );

    // LED debug
    assign LED[3:0] = units;
    assign LED[7:4] = tens;
endmodule
```

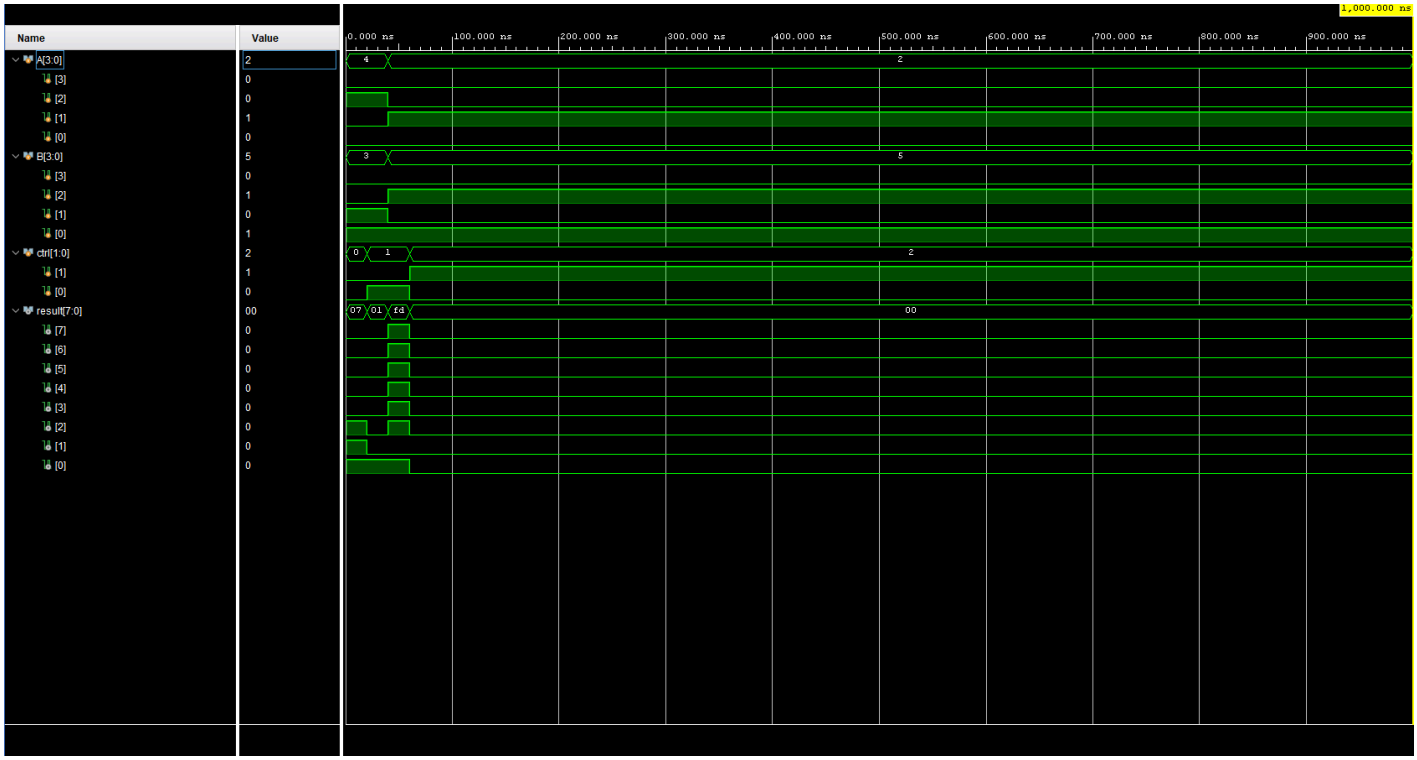
This is the top-level module that connects all components. It uses clock_divider to produce a slower clock signal based on the switch inputs. Two independent BCD counters count up or down based on their direction bits, and their values are passed to an ALU that performs addition or subtraction. The ALU output is converted to decimal digits for display on the 7-segment display. A third digit shows the control nibble. The LED outputs reflect the live values of the counters for real-time debugging.

Testbenches

Alu_tb.v

```
`timescale 1ns / 1ps
module alu_tb;
    reg [3:0] A, B;
    reg [1:0] ctrl;
    wire [7:0] result;

    alu uut (
        .A(A),
        .B(B),
        .ctrl(ctrl),
        .result(result)
    );

    initial begin
        $display("Starting alu_tb");

        A = 4'd4; B = 4'd3; ctrl = 2'b00; #20;
        $display("A = %d, B = %d, ctrl = %b -> result = %d", A, B, ctrl, result);

        ctrl = 2'b01; #20;
        $display("A = %d, B = %d, ctrl = %b -> result = %d", A, B, ctrl, result);

        A = 4'd2; B = 4'd5; ctrl = 2'b01; #20;
        $display("A = %d, B = %d, ctrl = %b -> result = %d (wrap from underflow)", A, B, ctrl, result);

        ctrl = 2'b10; #20;
        $display("ctrl = %b -> result = %d (default case)", ctrl, result);
    end
endmodule
```

```
Starting alu_tb
A =  4, B =  3, ctrl = 00 -> result =   7
A =  4, B =  3, ctrl = 01 -> result =   1
A =  2, B =  5, ctrl = 01 -> result = 253 (wrap from underflow)
ctrl = 10 -> result =   0 (default case)
```

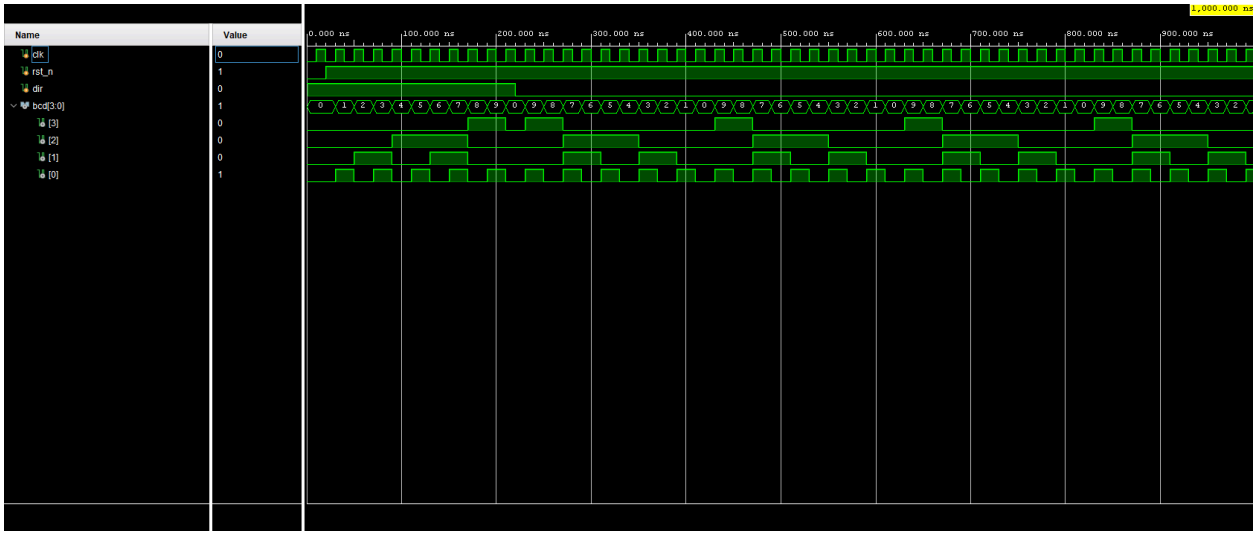

Name
Value
A[3:0]
2
[3]
0
[2]
0
[1]
1
[0]
0
B[3:0]
5
[3]
0
[2]
1
[1]
0
[0]
1
ctrl[1:0]
2
[1]
1
[0]
0
result[7:0]
00
[7]
0
[6]
0
[5]
0
[4]
0
[3]
0
[2]
0
[1]
0
[0]
0
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
600.000 ns
700.000 ns
800.000 ns
900.000 ns
1,000.000 ns
4
2
3
5
0
1
2
07
01
fd
00

Bcd_counter_tb.v

```
`timescale 1ns / 1ps
module bcd_counter_tb;
    reg clk = 0, rst_n = 0, dir = 1;
    wire [3:0] bcd;

    bcd_counter uut (
        .clk(clk),
        .rst_n(rst_n),
        .dir(dir),
        .bcd(bcd)
    );

    always #10 clk = ~clk;

    initial begin
        $display("Starting bcd_counter_tb");

        rst_n = 0; #20;
        rst_n = 1;
        dir = 1;
        $monitor("Time = %0t | dir = %b | bcd = %d", $time, dir, bcd);

        #200 dir = 0;
    end
endmodule
```

```
Starting bcd_counter_tb
Time = 20000 | dir = 1 | bcd =  0
Time = 30000 | dir = 1 | bcd =  1
Time = 50000 | dir = 1 | bcd =  2
Time = 70000 | dir = 1 | bcd =  3
Time = 90000 | dir = 1 | bcd =  4
Time = 110000 | dir = 1 | bcd =  5
Time = 130000 | dir = 1 | bcd =  6
Time = 150000 | dir = 1 | bcd =  7
Time = 170000 | dir = 1 | bcd =  8
Time = 190000 | dir = 1 | bcd =  9
Time = 210000 | dir = 1 | bcd =  0
Time = 220000 | dir = 0 | bcd =  0
Time = 230000 | dir = 0 | bcd =  9
Time = 250000 | dir = 0 | bcd =  8
Time = 270000 | dir = 0 | bcd =  7
Time = 290000 | dir = 0 | bcd =  6
Time = 310000 | dir = 0 | bcd =  5
Time = 330000 | dir = 0 | bcd =  4
Time = 350000 | dir = 0 | bcd =  3
Time = 370000 | dir = 0 | bcd =  2
Time = 390000 | dir = 0 | bcd =  1
```

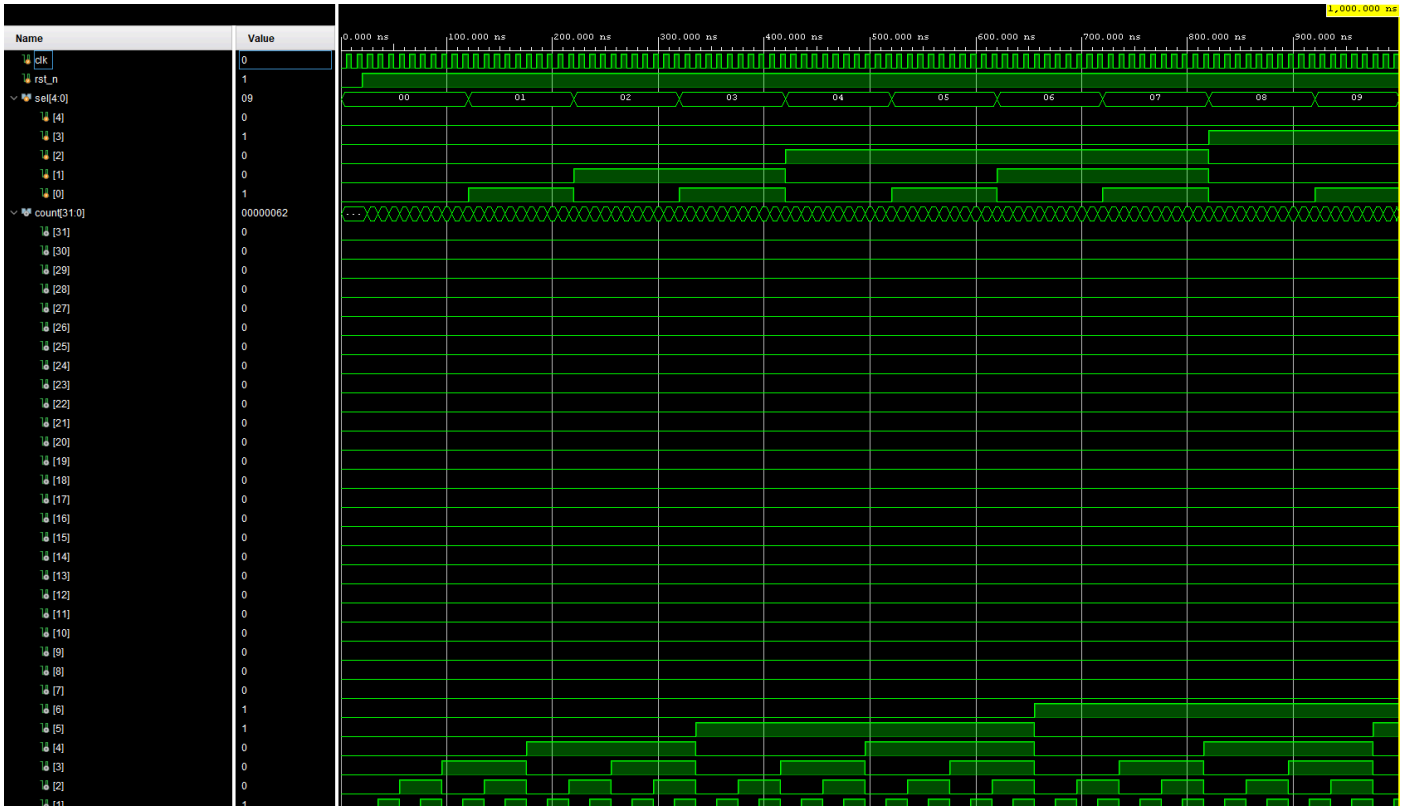
```
Time = 410000 | dir = 0 | bcd =  0
Time = 430000 | dir = 0 | bcd =  9
Time = 450000 | dir = 0 | bcd =  8
Time = 470000 | dir = 0 | bcd =  7
Time = 490000 | dir = 0 | bcd =  6
Time = 510000 | dir = 0 | bcd =  5
```



Clock_divider_tb.v

```
`timescale 1ns / 1ps
module clock_divider_tb;
    reg clk = 0;
    reg rst_n = 0;
    reg [4:0] sel = 5'd0;
    wire [31:0] count;
    wire clk_div;

    clock_divider uut (
        .clk(clk),
        .rst_n(rst_n),
        .sel(sel),
        .count(count),
        .clk_div(clk_div)
    );

    always #5 clk = ~clk;

    initial begin
        $display("Starting clock_divider_tb");
        rst_n = 0; #20;
        rst_n = 1;

        sel = 0;
        repeat (10) begin
            #100 $display("sel = %0d, clk_div = %b, count = %h", sel, clk_div, count);
            sel = sel + 1;
        end
    end
endmodule
```

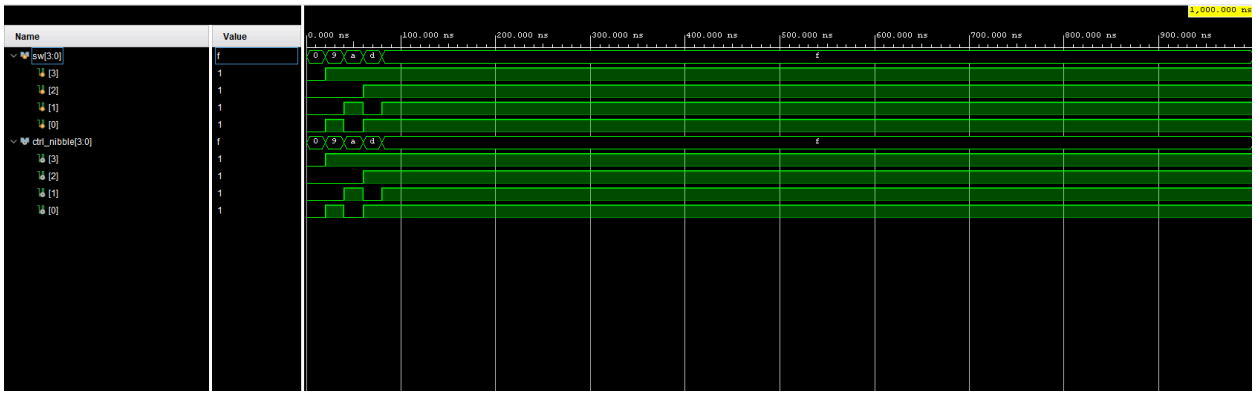
```
Starting clock_divider_tb
sel = 0, clk_div = 0, count = 0000000a
sel = 1, clk_div = 0, count = 00000014
sel = 2, clk_div = 1, count = 0000001e
sel = 3, clk_div = 1, count = 00000028
sel = 4, clk_div = 1, count = 00000032
sel = 5, clk_div = 1, count = 0000003c
sel = 6, clk_div = 1, count = 00000046
sel = 7, clk_div = 0, count = 00000050
sel = 8, clk_div = 0, count = 0000005a
```


Name
Value
clk
0
rst_n
1
sel[4:0]
09
[4]
0
[3]
1
[2]
0
[1]
0
[0]
1
count[31:0]
00000062
[31]
0
[30]
0
[29]
0
[28]
0
[27]
0
[26]
0
[25]
0
[24]
0
[23]
0
[22]
0
[21]
0
[20]
0
[19]
0
[18]
0
[17]
0
[16]
0
[15]
0
[14]
0
[13]
0
[12]
0
[11]
0
[10]
0
[9]
0
[8]
0
[7]
0
[6]
1
[5]
1
[4]
0
[3]
0
[2]
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
1,000.000 ns
00
01
02
03
04
05
06
07
08
09

control_decoder_tb.v

```
`timescale 1ns / 1ps
module control_decoder_tb;
    reg [3:0] sw;
    wire [3:0] ctrl_nibble;

    control_decoder uut (
        .sw(sw),
        .ctrl_nibble(ctrl_nibble)
    );

    initial begin
        $display("Starting control_decoder_tb");
        $monitor("SW = %b (%1X) -> ctrl_nibble = %b (%1X)", sw, sw, ctrl_nibble, ctrl_nibble);

        sw = 4'b0000; #20;  // Expect 0
        sw = 4'b1001; #20;  // Expect 9
        sw = 4'b1010; #20;  // Expect A
        sw = 4'b1101; #20;  // Expect D
        sw = 4'b1111; #20;  // Expect F
    end
endmodule
```

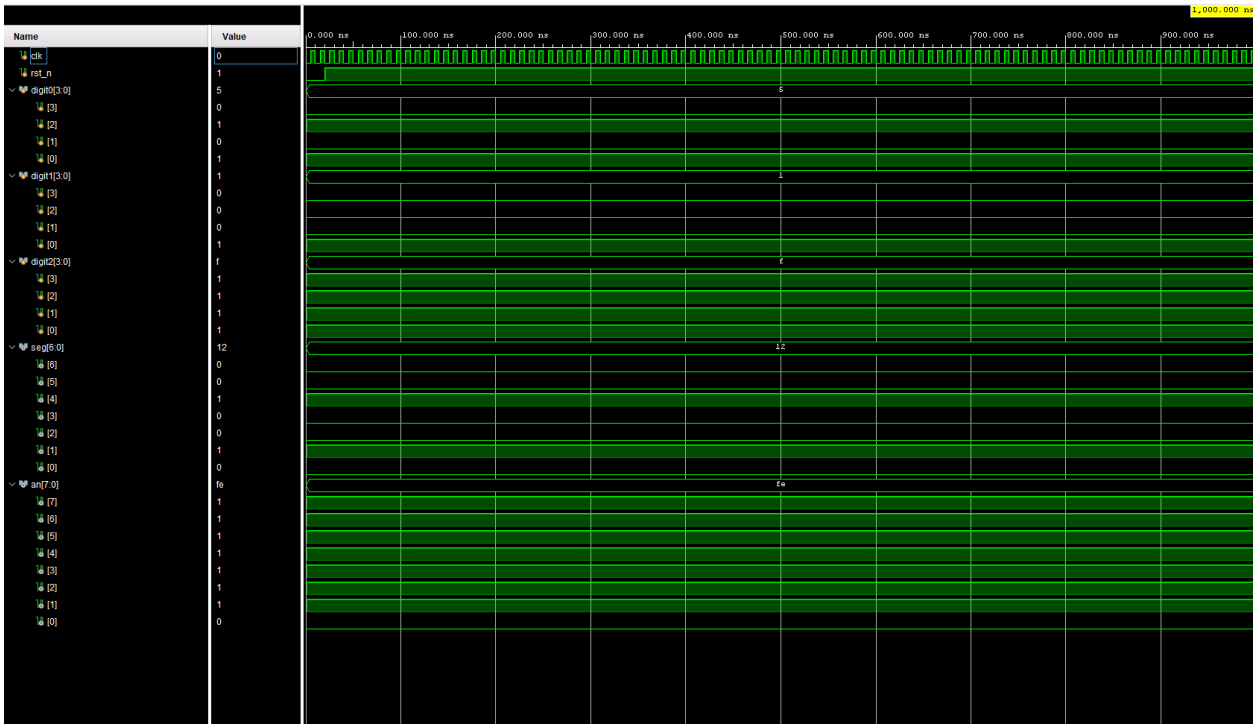
```
Starting control_decoder_tb
SW = 0000 (0) -> ctrl_nibble = 0000 (0)
SW = 1001 (9) -> ctrl_nibble = 1001 (9)
SW = 1010 (a) -> ctrl_nibble = 1010 (a)
SW = 1101 (d) -> ctrl_nibble = 1101 (d)
SW = 1111 (f) -> ctrl_nibble = 1111 (f)
```



tb_seg7_scan.v

```
`timescale 1ns / 1ps
module seg7_scan_tb;
    reg clk = 0, rst_n = 0;
    reg [3:0] digit0 = 4'd5, digit1 = 4'd1, digit2 = 4'd15;
    wire [6:0] seg;
    wire [7:0] an;

    seg7_scan uut (
        .clk(clk),
        .rst_n(rst_n),
        .digit0(digit0),
        .digit1(digit1),
        .digit2(digit2),
        .an(an),
        .seg(seg)
    );


    always #5 clk = ~clk;

    initial begin
        $display("Starting seg7_scan_tb");
        rst_n = 0; #20;
        rst_n = 1;
    end

    // Monitor output values during simulation
    initial begin
        $monitor("Time = %0t | AN = %b | SEG = %b | digit0 = %1X | digit1 = %1X | digit2 = %1X",
                 $time, an, seg, digit0, digit1, digit2);
    end
endmodule
```

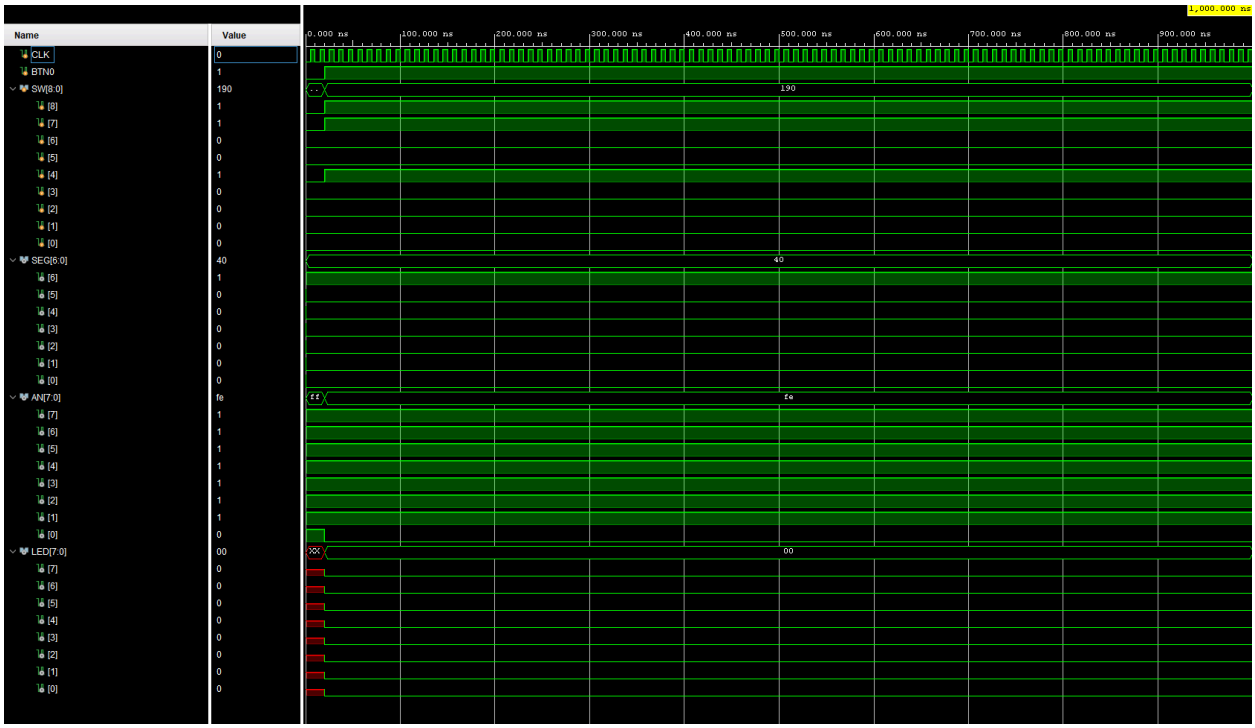
```
Starting seg7_scan_tb
Time = 0 | AN = 11111110 | SEG = 0010010 | digit0 = 5 | digit1 = 1 | digit2 = f
```


Name
Value
clk
0
rst_n
1
digit0[3:0]
5
[3]
0
[2]
1
[1]
0
[0]
1
digit1[3:0]
1
[3]
0
[2]
0
[1]
0
[0]
1
digit2[3:0]
f
[3]
1
[2]
1
[1]
1
[0]
1
seg[6:0]
12
[6]
0
[5]
0
[4]
1
[3]
0
[2]
0
[1]
1
[0]
0
an[7:0]
fe
[7]
1
[6]
1
[5]
1
[4]
1
[3]
1
[2]
1
[1]
1
[0]
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
1,000.000 ns

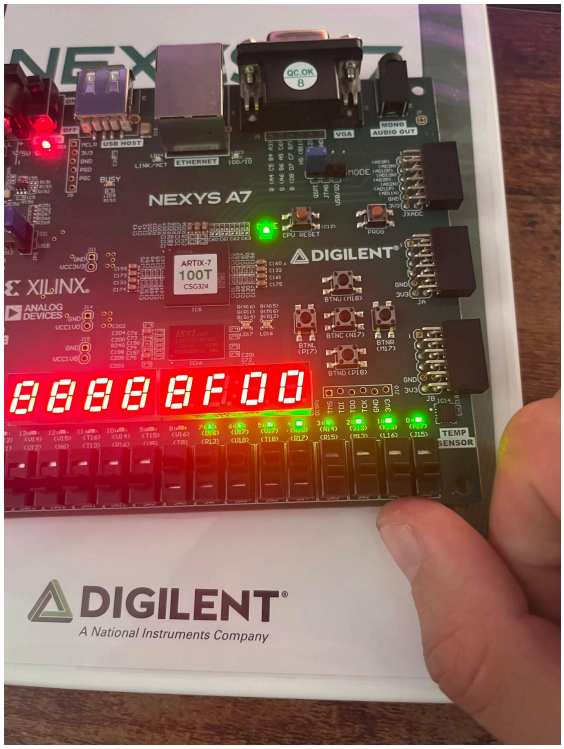
top_lab6_tb.v

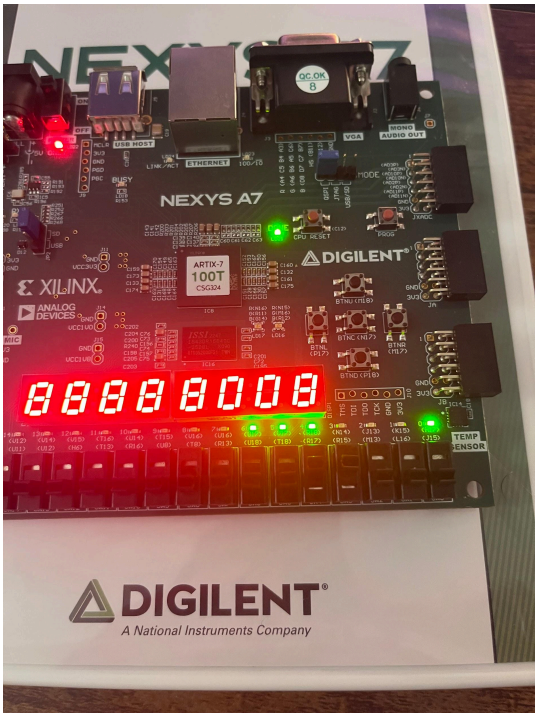
```
`timescale 1ns / 1ps
module top_lab6_tb;

    reg CLK = 0;
    reg BTN0 = 0;
    reg [8:0] SW = 9'b000000000;

    wire [6:0] SEG;
    wire [7:0] AN;
    wire [7:0] LED;

    top_lab6 uut (
        .CLK(CLK),
        .SW(SW),
        .BTN0(BTN0),
        .SEG(SEG),
        .AN(AN),
        .LED(LED)
    );

    always #5 CLK = ~CLK;

    initial begin
        $display("Starting top_lab6_tb");
        $monitor("Time = %0t | LED = %b | SEG = %b | AN = %b", $time, LED, SEG, AN);

        // Initial reset
        BTN0 = 0; #20;
        BTN0 = 1;

        SW[4:0] = 5'd16;

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

        // Count up on both counters
        SW[7] = 1; // units
        SW[8] = 1; // tens

        // Simulate longer to see counting
        #50000;


    end

endmodule
```

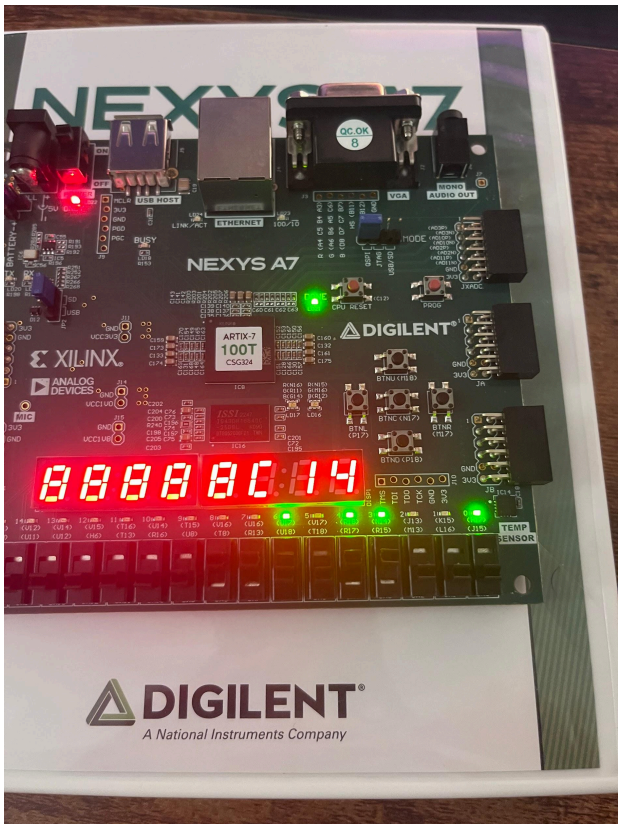
```
Starting top_lab6_tb
Time = 0 | LED = xxxxxxxx | SEG = 1000000 | AN = 11111111
Time = 20000 | LED = 00000000 | SEG = 1000000 | AN = 11111110
```

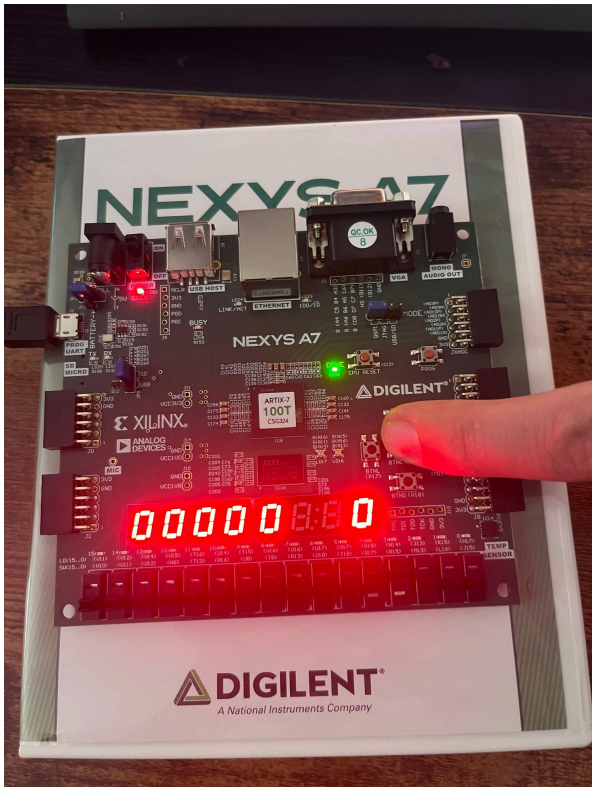


Hardware Results



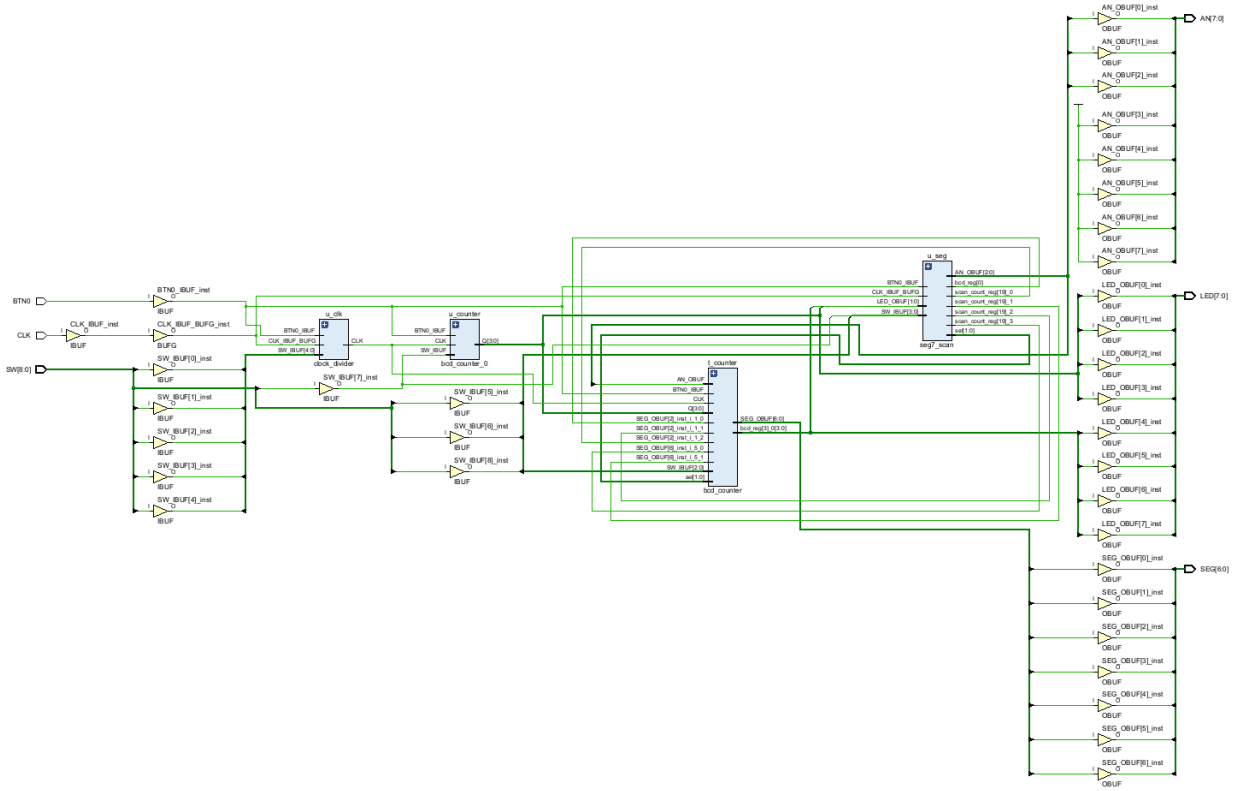






Screenshots

Schematic



LUT & FF

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
N top_lab6	40	60	4	26	40	29	1

Timing Summary

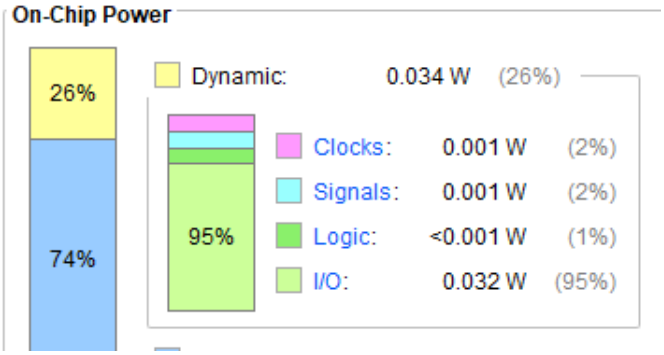
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.311 ns	Worst Hold Slack (WHS):	0.252 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing 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.

Power Summary

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**



Conclusion

This lab successfully demonstrated the use of multiple interacting hardware modules in Verilog to perform real-time arithmetic and display output on an FPGA. The design met all required functionality: two BCD counters independently counted up or down based on switch settings; the ALU performed addition and subtraction operations correctly, including edge cases like underflow; and the result was properly split and shown across two 7-segment digits. The control nibble display and LED outputs further supported debugging and confirmed correct behavior.

Contributions

Faris (50%) - Source code, helped with lab report

Nicholas (50%) - Testbench, handled simulations, helped with report, demo