ECE 3300 – Lab 6 Report
Dual BCD Up/Down Counters, ALU, and Control Display on 7-Segment
Team Name: Group V
Date: July 28, 2025

Group Members

Nathan Marlow, Khristian Chan

1. Objective

The objective of this lab is to design and implement a dual BCD up/down counter system on the Nexys A7 FPGA using Verilog. Each counter (units and tens) is independently controlled by direction switches, with a reset button and adjustable clock speed via a 32-bit clock divider and multiplexer. The counters' output values are used as inputs to a 4-bit ALU that performs addition or subtraction, controlled by dedicated switches. The final output, including the ALU result and a control nibble, is visualized through a 3-digit 7-segment display, while raw BCD values are mirrored on the LEDs for debugging. This lab combines sequential logic (counters), arithmetic logic (ALU), control decoding, and real-time digital display.

2. Verilog Code

Top Module:

The top_lab6 module serves as the main integration point. It wires together the clock divider, two BCD counters, ALU, control nibble decoder, and 7-segment display scanner. It reads the 100 MHz system clock and controls the speed through a 5-bit switch-controlled clock divider. It also assigns reset (BTN0), direction switches (SW7, SW8), ALU operation mode (SW6–SW5), and output indicators.

```
module top_lab6(
input clk,
input rst_n,
input [8:0]sw, //4:0 Clock Control, 6:5 alu control, 8:7 up/down
output[7:0]led,//7:4 tens debug, 3:0 units debug
output[7:0]an,
output [6:0]seg,
output DP
);
wire clk_out;
wire[3:0]units_bcd, tens_bcd;
wire[7:0]result;
wire[3:0]ctrl_nibble;
wire [3:0]tens = (result/10) % 10;
wire [3:0]units = result % 10;


//BCD result
assign DP = 1'b1;

clock_divider u_clock_divider(
.clk(clk),
.rst_n(rst_n),
.sel(sw[4:0]),
.clk_out(clk_out)
);

bcd_counter u_counter_units(
    .clk_div(clk_out), .BTN0(rst_n), .dir_bit(sw[7]), .BCD(units_bcd)
    );

bcd_counter u_counter_tens(
    .clk_div(clk_out), .BTN0(rst_n), .dir_bit(sw[8]), .BCD(tens_bcd)
);

alu u_alu(
    .A(units_bcd), .B(tens_bcd), .CTRL(sw[6:5]), .RESULT(result)
);

    control_decoder u_ctrl(
        .nibble({sw[8:5]}), .ctrl_nibble(ctrl_nibble)
    );


seg7_scan u_seg(
    .clk(clk), .rst_n(rst_n),
    .dig0(units), .dig1(tens), .dig2(ctrl_nibble),
    .an(an), .seg(seg)
);

assign led[3:0] = units_bcd;
assign led[7:4] = tens_bcd;
```

7Seg Driver:

```
    input clk,
    input rst_n,
    input [3:0] dig0,
    input [3:0] dig1,
    input [3:0] dig2,
    output reg [7:0] an,
    output reg [6:0] seg
);
//instantiate wires/registers for secondary clock divider
    reg[16:0] secondary_divider;
    wire divbit = secondary_divider[15];
    reg [1:0] sel;
    reg [3:0] digit;
//clock divider circuit
    always @(posedge clk or negedge rst_n)
        if(!rst_n) secondary_divider <= 0;
        else secondary_divider <= secondary_divider + 1;
//select which anode to drive
    always @(posedge clk or negedge rst_n)
        if (!rst_n) sel <= 0;
        else if(divbit)
        sel <= sel + 1;
//case statement to drive anode
    always @(*) begin
        case (sel)
            2'b00: begin an = 8'b11111110; digit = dig0; end
            2'b01: begin an = 8'b11111101; digit = dig1; end
            2'b10: begin an = 8'b11111011; digit = dig2; end
            default: an = 8'b11111111; //digit = 4'd0; end
        endcase
    end
//drive this set of bits on the 7seg
    always @(*) begin
        case (digit)
            4'd0: seg = 7'b1000000;
            4'd1: seg = 7'b1111001;
            4'd2: seg = 7'b0100100;
            4'd3: seg = 7'b0110000;
            4'd4: seg = 7'b0011001;
            4'd5: seg = 7'b0010010;
            4'd6: seg = 7'b0000010;
            4'd7: seg = 7'b1111000;
            4'd8: seg = 7'b0000000;
            4'd9: seg = 7'b0010000;
            4'd10: seg = 7'b0001000;
            4'd11: seg = 7'b0000011;
            4'd12: seg = 7'b1000110;
            4'd13: seg = 7'b0100001;
            4'd14: seg = 7'b0000110;
            4'd15: seg = 7'b0001110;
            default: seg = 7'b1111111;
        endcase
```

Clock Divider:

```
`timescale 1ns / 1ps

module clock_divider(
input clk,
input rst_n,
input [4:0]sel,
output clk_out
);
reg [31:0]cnt;
always @(posedge clk or negedge rst_n) begin
    if(!rst_n)
        cnt <= 0;//if reset is low, count is 0
    else
        cnt <= cnt + 1;//if reset is high count can increment
end

assign clk_out = cnt[sel]; //we assign our clock signal to a single cnt index, since a single index is proportional to clock period

endmodule
```

Control Decoder:

```
`timescale 1ns / 1ps

module control_decoder(
input [3:0]nibble,
output reg[3:0]ctrl_nibble
);
always @ (*)begin
ctrl_nibble = nibble;//decoder where input=output, no change at all
end
endmodule
```

ALU:

```
module alu(
input [3:0]A,
input [3:0]B,
input [1:0]CTRL,
output reg[7:0]RESULT
    );
//mux to choose which math to do
always @(*) begin
    case (CTRL)
        2'b00: begin
            RESULT = A + B; //normal addition
        end

        2'b01: begin
            if (A > B)
                RESULT = A - B; //normal subtraction
            else
                RESULT = A + 18 - B; //rollover, if A-B is less than zero, it will rollover by adding 18 as a "reset"
        end

        default: begin
            RESULT = 0;
        end
    endcase
end

endmodule
```

BCD Counter:

```
module bcd_counter(
input clk_div,
input BTN0,//rst_n,
input dir_bit,//sw7 or sw8 (instantiated twice),
output reg[4:0]BCD
);
//rst_n low output = 0
//each rising edge of clk_div, dir = 1, inc, else de

always @ (posedge clk_div or negedge BTN0) begin

    if (!BTN0)begin
        BCD <= 0;
    end else begin
        if (dir_bit) begin // counting up
            if (BCD == 9)
                BCD <= 0;
            else
                BCD <= BCD + 1;
        end else begin
            if (BCD == 0)
                BCD <= 9;
            else
                BCD <= BCD - 1;
            end
        end
    end

endmodule
```

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	65	0	0	63400	0.10
LUT as Logic	65	0	0	63400	0.10
LUT as Memory	0	0	0	19000	0.00
Slice Registers	64	0	0	126800	0.05
Register as Flip Flop	60	0	0	126800	0.05
Register as Latch	4	0	0	126800	<0.01
F7 Muxes	8	0	0	31700	0.03
F8 Muxes	0	0	0	15850	0.00

* Warning! LUT value is adjusted to account for LUT combining

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

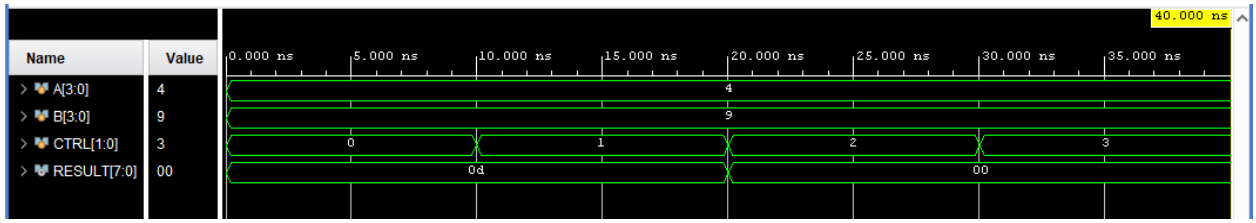
* Specify Design Power Budget using, set_operating_conditions -design_power_budget <value in Watts>

1.1 On-Chip Components

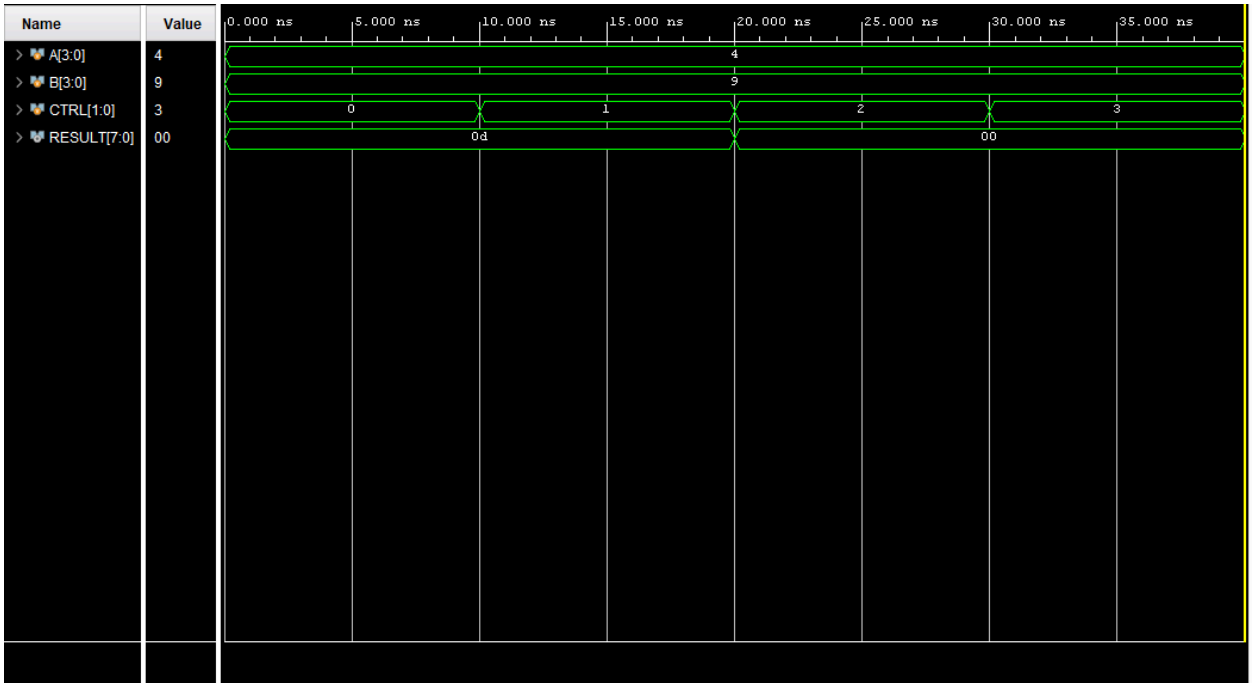
On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	<0.001	3	---	---
Slice Logic	<0.001	167	---	---
LUT as Logic	<0.001	65	63400	0.10
Register	<0.001	64	126800	0.05
CARRY4	<0.001	12	15850	0.08
F7/F8 Muxes	<0.001	8	63400	0.01
Others	0.000	9	---	---
Signals	<0.001	136	---	---
I/O	0.031	35	210	16.67
Static Power	0.097			
Total	0.131			

Simulation Results:

alu_tb.v

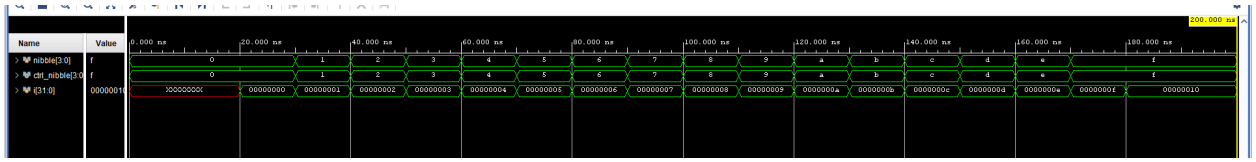


We see that our ALU works as intended, when switch 6 is high, the result is 0, or else the result is addition or subtraction with an underflow.



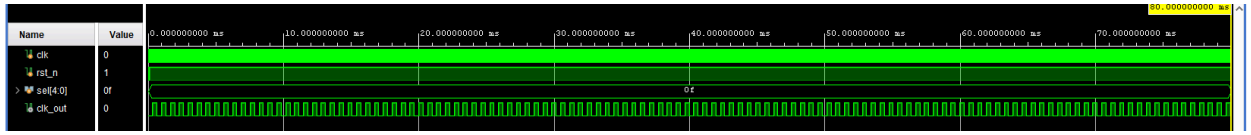
We see for our values 4 and 9, when we add 4 and 9 we get 13, 4-9 is negative 5, but underflow wraps it back to 13.

control_decoder_tb.v

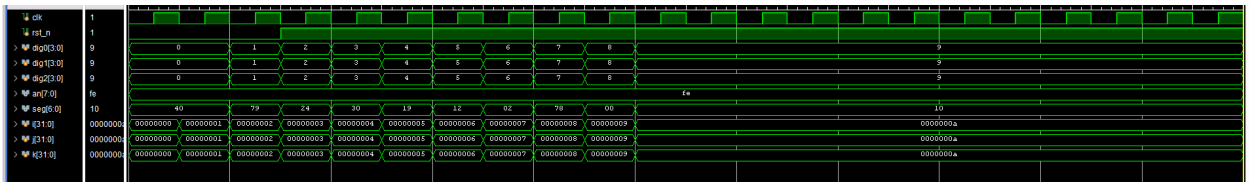


We see our decoder goes through every possible value.

clock_divider_tb.v

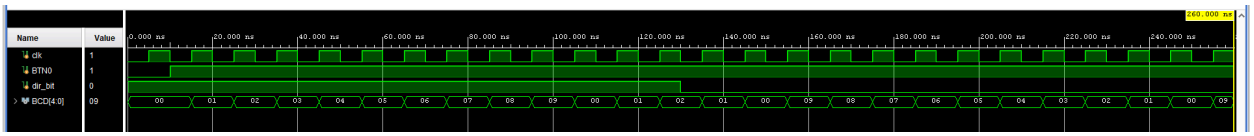


Seg7_scan_tb.v (set clock speed to fastest speed for this simulation)



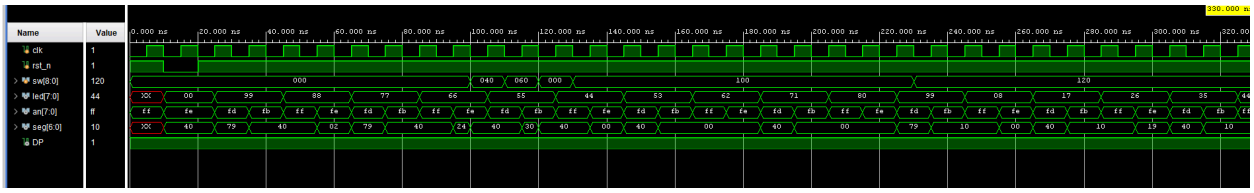
We see that our 7seg goes through all values correctly, and polls through each number correctly in our for loop

bcd_counter_tb.v



We count both up and down, and we see our overflow and underflow works as intended

Top_lab6_tb.v (we set our clock to fastest for this simulation)



We see that our module goes through every assigned value correctly.

6. Demonstration Video

https://youtu.be/AlvJzK5goLM?si=YMC3VkNM3EkGJOgh

Contributions:

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