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

Dia Agrawal and Robert Lainez Torres

Objective

The goal of this lab was to design and implement a digital system on the Nexys A7 board that integrates several functional components. These include two independent BCD up/down counters for the units and tens digits, each controlled by separate direction bits, and a 4-bit ALU capable of performing addition and subtraction operations on the BCD values. The system also features a control digit display that reflects the current switch settings and a 3-digit 7-segment display used to present the ALU results and control nibble. Additionally, LED outputs were utilized to provide real-time debugging information by displaying the raw BCD values from each counter.

Design Description

Clock Divider:

```
`timescale 1ns / 1ps


module clock_divider(
    input clk,                  // 100 MHz
    input [4:0] sw,            // SW[4:0]
    output clk_div              // selected bit
    );

    reg [31:0] counter ;

    always @(posedge clk)
        counter <= counter + 1;

    assign clk_div = counter[sw];  // Select 1 of 32 bits

endmodule
```

A 32-bit free-running counter was implemented using the 100 MHz system clock to generate a slower, observable timing signal. A 32-to-1 multiplexer is used to select one of the counter's bits based on the value of the switches SW[4:0], effectively allowing the user to choose the division factor and adjust the clock speed. The selected bit output, referred to as clk_div , serves as the input clock for the BCD counters, enabling visible counting rates suitable for real-time observation and debugging.

BCD counters:

```
`timescale 1ns / 1ps


module bcd_counter(
    input clk_div,
    input btn0,
    input dir_bit,              // 1 = up, 0 = down
    output reg [3:0] led
    );

    always @(posedge clk_div or negedge btn0)
        if (!btn0)
            led <= 4'b0000;
        else begin
            if (dir_bit==1'b1) begin
                if (led == 4'd9)
                    led <= 4'd0;
                else
                    led <= led + 1;
            end else begin
                if (led == 4'd0)
                    led <= 4'd9;
                else
                    led <= led - 1;
            end
        end

endmodule
```

Two instances of the BCD counter module were implemented—one for the units digit and one for the tens digit. Each counter receives the divided clock signal (clk_div) from the clock divider, an active-low reset signal (reset_n) from BTN0, and a direction control input (dir) from either SW7 (for the units counter) or SW8 (for the tens counter). On each rising edge of clk_div , the counter increments if the direction bit is set to 1, or decrements if the bit is set to 0. The counters are designed to roll over from 9 to 0 when incrementing, and from 0 to 9 when decrementing. The output of each counter is a 4-bit BCD value, with the units counter output mapped to LED[3:0] and the tens counter output mapped to LED[7:4] for debugging and visualization purposes.

ALU:

```
`timescale 1ns / 1ps
//////////////////////////////////////


module alu(
    input [3:0] A,
    input [3:0] 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'h00;
        endcase
    end
endmodule
```

The ALU module receives two 4-bit BCD inputs: A from the units counter and B from the tens counter. It also takes a 2-bit control signal (ctrl) from switches SW[6:5], which determines the operation to be performed. When the control input is 00, the ALU adds inputs A and B; when it is 01, the ALU subtracts B from A. If the subtraction results in an underflow, the output wraps around to maintain an 8-bit result. For any other control values, the output is set to zero. The result of the ALU operation is an 8-bit value, which is split into high and low 4-bit nibbles and displayed on the 7-segment display across two digits.

Control Decoder:

```
`timescale 1ns / 1ps


module control_decoder(
    input [3:0] sw,       // {SW8, SW7, SW6, SW5}
    output [3:0] ctrl_nibble
    );

    assign ctrl_nibble = sw;
endmodule
```

The control decoder module takes a 4-bit input formed by concatenating the switch values {SW8, SW7, SW6, SW5}, which represent the current settings for the counter directions and ALU operation. This input is passed directly as a 4-bit output called ctrl_nibble The ctrl_nibble is then displayed on digit AN2 of the 3-digit 7-segment display, allowing the user to visually confirm the current control settings in real time.

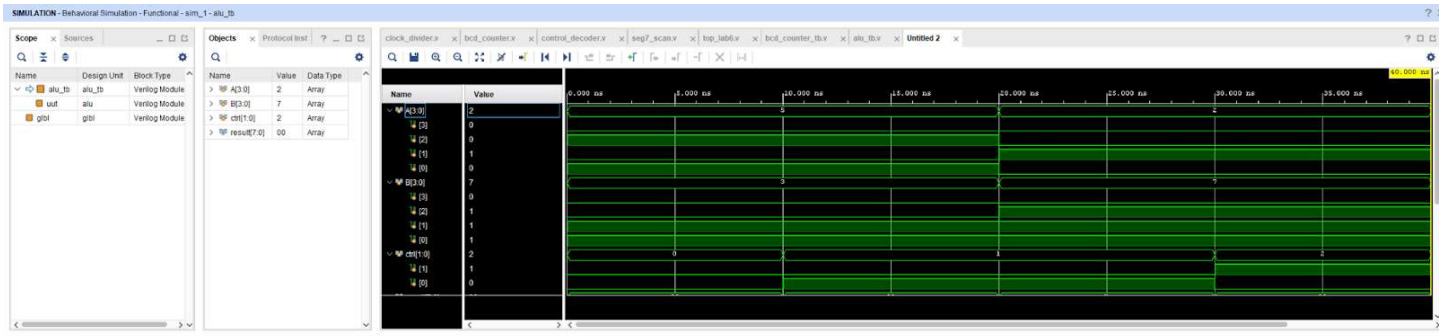
7 Segment Display:

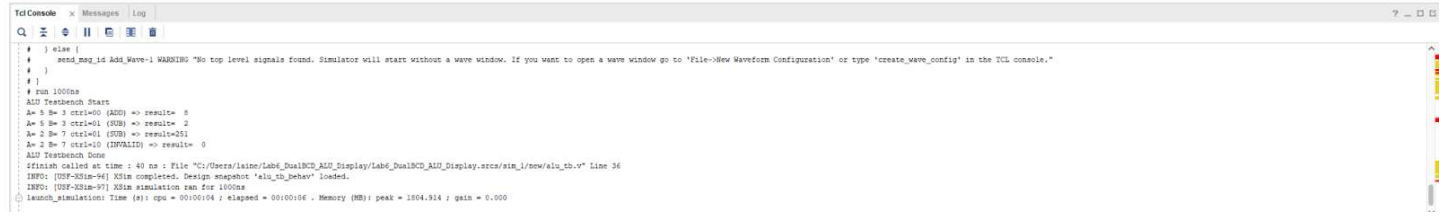
timescale 1ns / 1ps

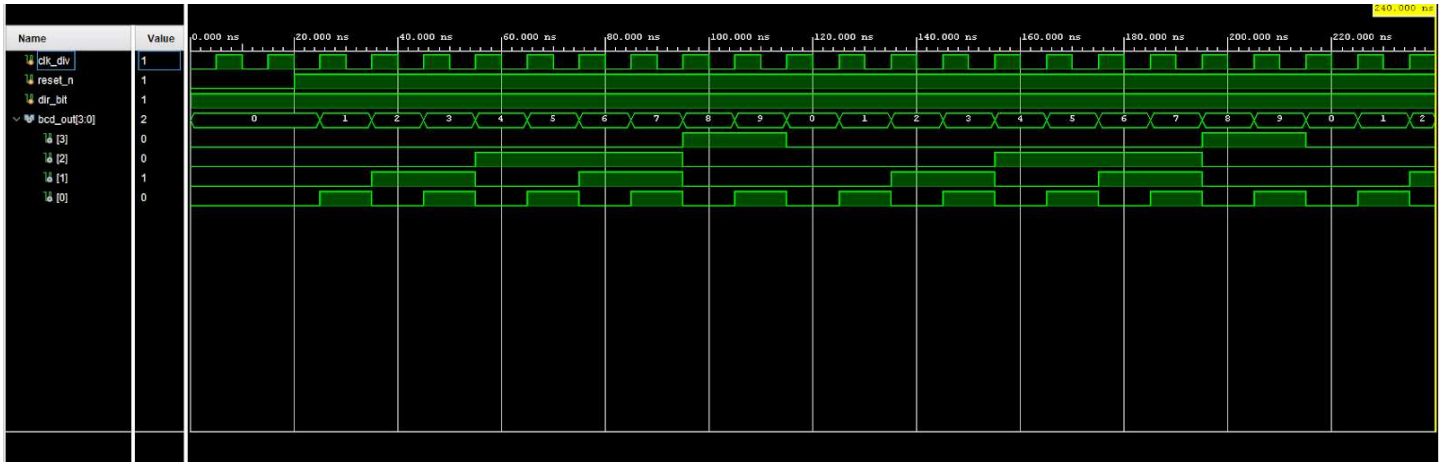
```
module seg7_scan(
    input wire clk,
    input wire btn0,
    input wire [7:0] result,
    input wire [3:0] ctrl_nibble,
    output reg [6:0] seg,
    output reg [2:0] an
    );

    reg [15:0] clk_divider_count = 0;
   wire scan_clk;
    reg [1:0] data_to_display;

    always @(posedge clk or posedge btn0) begin
        if (~btn0)
            clk_divider_count <= 0;
        else
            clk_divider_count <= clk_divider_count + 1;
    end

    always @(posedge clk or posedge btn0) begin
        if(~btn0)
             clk_divider_count <= 0;
        else if (scan_clk);
            data_to_display <= data_to_display + 1;
    end


    always @(*) begin
        case (data_to_display)
            4'h0: seg = 7'b1000000; // 0
            4'h1: seg = 7'b1111001; // 1
            4'h2: seg = 7'b0100100; // 2
            4'h3: seg = 7'b0110000; // 3
            4'h4: seg = 7'b0011001; // 4
            4'h5: seg = 7'b0010010; // 5
            4'h6: seg = 7'b0000010; // 6
            4'h7: seg = 7'b1111000; // 7
```

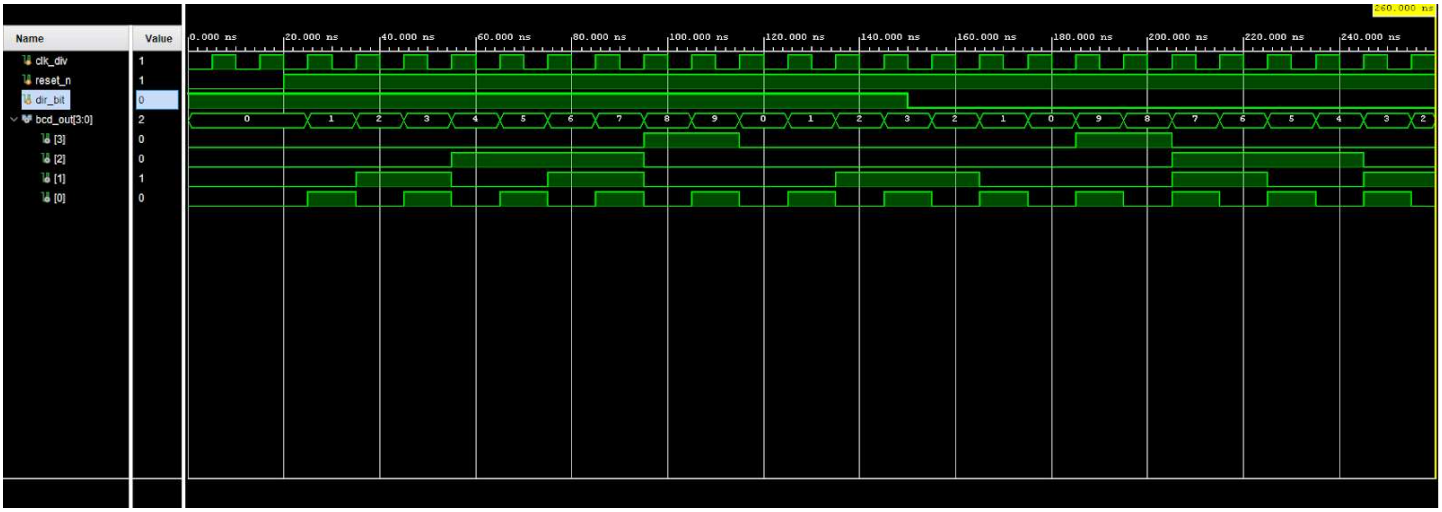
```
            4'h8: seg = 7'b0000000; // 8
            4'h9: seg = 7'b0010000; // 9
            4'hA: seg = 7'b0001000; // A
            4'hB: seg = 7'b0000011; // b
            4'hC: seg = 7'b1000110; // C
            4'hD: seg = 7'b0100001; // d
            4'hE: seg = 7'b0000110; // E
            4'hF: seg = 7'b0001110; // F
            default: seg = 7'b1111111; // Off
        endcase
    end
endmodule
```

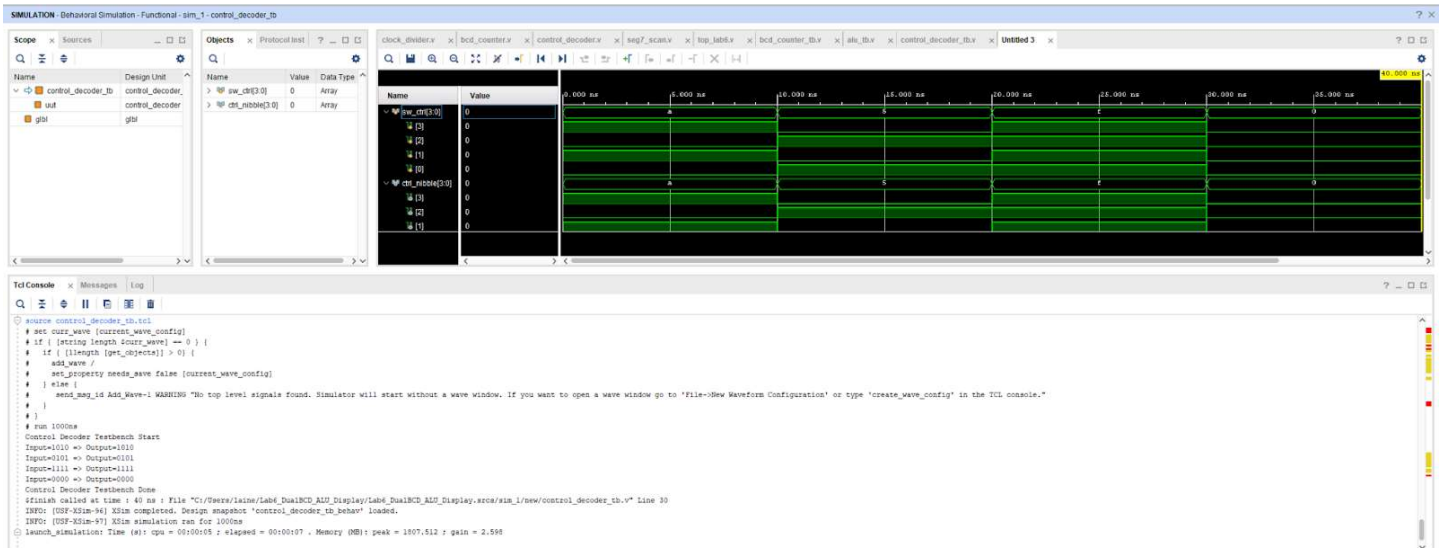
TEST BENCH:



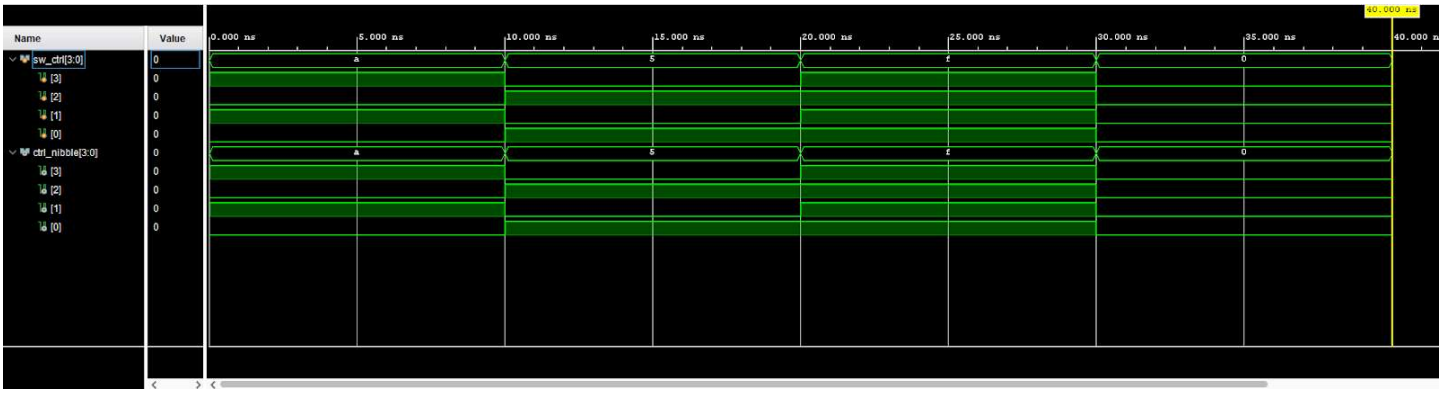


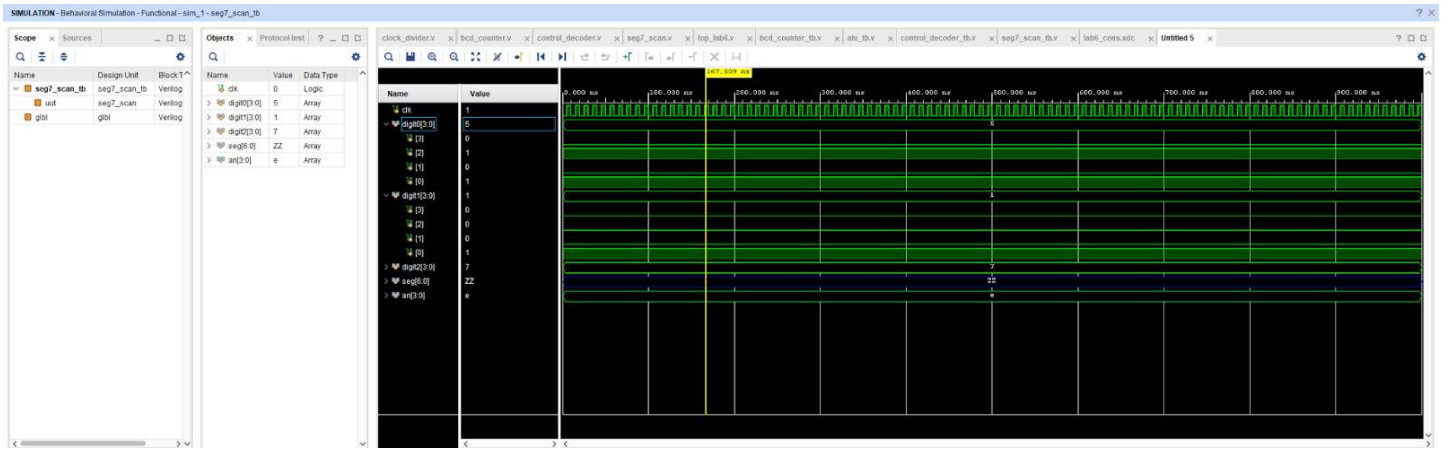


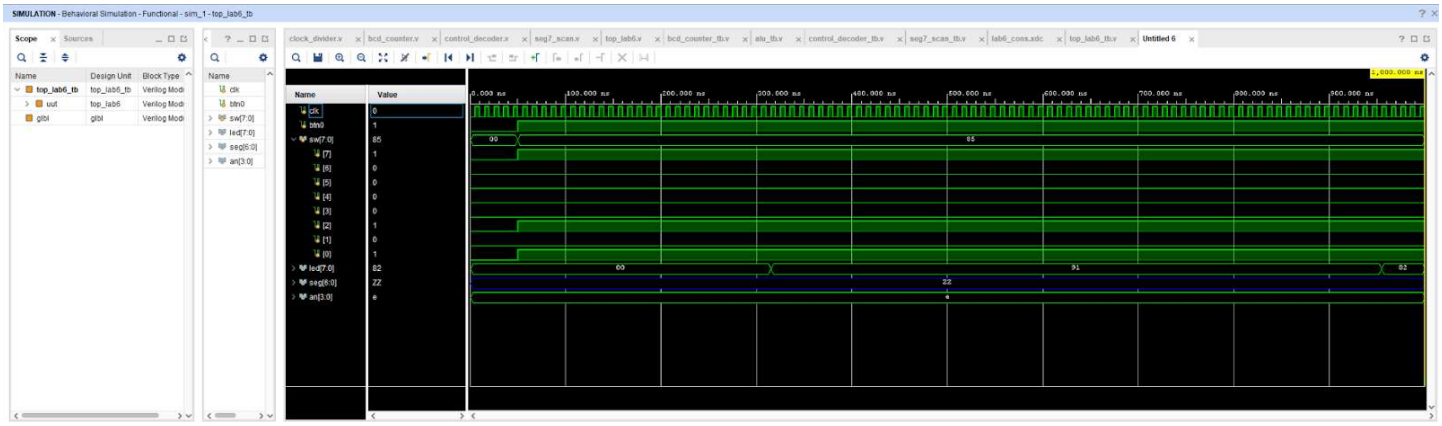





```
Control Decoder Testbench Start
Input=1010 => Output=1010
Input=0101 => Output=0101
Input=1111 => Output=1111
Input=0000 => Output=0000
Control Decoder Testbench Done
$finish called at time : 40 ns : File "C:/Users/laine/lab6/lab6.srcs/sim_1/new/control_decoder_tb.v" Line 29
INFO: [USF-XSim-96] XSim completed. Design snapshot 'control_decoder_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:01 ; elapsed = 00:00:06 . Memory (MB): peak = 1516.410 ; gain = 0.000
```







Utilization Report

Resource	Utilization	Available	Utilization %
LUT	37	63400	0.06
FF	58	126800	0.05
IO	29	210	13.81

Team Contributions

This lab involved more module interaction and timing sensitivity than previous ones, requiring careful debugging and multiple simulation stages. As usual, we both contributed throughout the design, testing, and documentation process, often working together to integrate and verify our modules.

Robert's Contributions:
I wrote initial versions of the controller and timing modules, and set up the first simulation environment we used for functional testing. I also helped debug issues with clock division and signal synchronization. For the report, I gathered simulation waveforms and wrote the explanation for the controller logic and the clocking scheme.

Dia's Contributions:
I developed the datapath and output modules, and worked on wiring everything together in the top-level design. I created the final simulation testbench and handled synthesis and implementation in Vivado. I also recorded the demo video and wrote the lab report sections covering the datapath behavior and overall module integration.

We both participated in debugging, testbench design, and reviewing each other's code to ensure correctness. Final documentation and polishing were done collaboratively.