CALIFORNIA STATE POLYTECHNIC
UNIVERSITY, POMONA
COLLEGE OF ENGINEERING

LAB 6

Two Independent BCD Up/Down Counters

ECE 3300L Summer 2025

Digital Circuit Design using Verilog

Professor Mohamed Aly

By: Clay Kim and Changwe Musonda

Objective

This lab teaches you how to create a digital project with two independent BCD up/down counters, a 4-bit ALU, and a 7-segment display decoder.. The goal is to reinforce understanding of sequential logic, arithmetic logic, and display interfacing by creating a system responsive to switch inputs and pushbuttons on the Nexys A7 FPGA board. This project equals an assessment of logical understanding of digital components and physical work with chips on a board.

Hardware Pin Mapping

- **CLK:** 100MHz On-board clock
- **SW0 –** SW4: SW[4:0] Speed select (0 = slowest ... 31 = fastest)
- **SW5-SW6:** SW[6:5] ALU control bits(00 = add, 01 = sub)
- **Signal:** Port/Pin
- **SW7:** SW7 Units counter direction (1 = up, 0 = down)
- **SW8:** SW8 Tens counter direction (1 = up, 0 = down)
- **BTN0:** PushBtn0 Reset both counters (active low)
- **AN[2:0]:** AN0 – AN2 Digit enables for 3-digit scan
- **SEG[6:0]:** SEG A-G Seven-segment
- **LED[3:0]:** LED0 - LED3 Units counter BCD
- **LED[7:4]:** LED4 – LED7 Tens counter BCD

Verilog Code:

alu.v

```
module alu(
   input wire [3:0] A,
   input wire [3:0] B,
   input wire [6:5] 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
```

bcd_counter.v

```
module bcd_counter(
```

```
    input wire clk_div,
    input wire BTN0,
    input wire dir_bit,
    output reg [3:0] LED
    );

    always@(posedge clk_div or negedge BTN0)
        if (~BTN0)
            LED <= 4'b0000;
        else if (dir_bit == 1'b1) begin // Increment mode
                if (LED == 4'd9) // If current value is 9
                    LED <= 4'd0;    // Wrap around to 0
                else
                    LED <= LED + 1; // Increment by 1
            end else begin // Decrement mode (dir_bit == 0)
                if (LED == 4'd0)  // If current value is 0
                    LED <= 4'd9;    // Wrap around to 9
                else
                    LED <= LED - 1; // Decrement by 1
                end
endmodule
```

clock_divider.v

```
module clock_divider(
    input wire clk,
    input wire [4:0] SW,
    output wire clk_div
    );
    reg [31:0] counter;

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

    assign clk_div = counter[SW];

endmodule
```

control_decoder.v

```
module control_decoder(
    input wire [8:5] SW,
    output wire [3:0] ctrl_nibble
    );

    assign ctrl_nibble = SW;

endmodule
```

seg7_scan.v

```
module seg7_scan(
    input wire clk,
    input wire BTN0,
    input wire [7:0] result,
    input wire [3:0] ctrl_nibble,
    output reg [2:0] AN,
    output reg [6:0] SEG
    );

    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
```

top_lab6.v

```
module top_lab6(
  input wire clk,
  input wire BTN0,
  input wire [8:0] SW,
  output wire [2:0] AN,
  output wire [6:0] SEG
  );

  wire one_hz_clk;            // 1Hz clock from the divider
  wire [3:0] units_BCD;       // Units digit from the BCD counter
  wire [3:0] tens_BCD;        // Tens digit from the BCD counter
  wire [7:0] alu_result;      // 8-bit result from the ALU
  wire [3:0] ctrl_nibble_out; // 4-bit output from the control decoder

  clock_divider clk_div_inst (
    .clk(clk),
    .BTN0(BTN0),
    .clk_out(one_hz_clk)
  );

  bcd_counter bcd_count_inst (
    .clk(clk),
    .BTN0(BTN0),
    .enable(one_hz_clk),
```

```
        .units(units_BCD),
        .tens(tens_BCD)
    );

    alu alu_inst (
        .A(units_BCD),
        .B(tens_BCD),
        .ctrl(SW[6:5]),
        .result(alu_result)
    );

    control_decoder ctrl_dec_inst (
        .nibble(SW[8:5]),
        .ctrl_nibble(ctrl_nibble_out)
    );

    seg7_scan seg7_scan_inst (
        .clk(clk),
        .BTN0(BTN0),
        .result(alu_result),
        .ctrl_nibble(ctrl_nibble_out),
        .AN(AN),
        .SEG(SEG)
    );
endmodule
```

Testbench:

alu_tb.v

```
module alu_tb;
   // Declare inputs as regs and outputs as wires
   reg [3:0] A, B;
   reg [2:0] opcode;
   wire [3:0] Y;
   wire cout, zero;

   // Instantiate the ALU
   alu uut (
      .A(A),
      .B(B),
      .opcode(opcode),
      .Y(Y),
      .cout(cout),
      .zero(zero)
   );

   initial begin
      // Monitor values
      $monitor("Time=%0t A=%h B=%h opcode=%b | Y=%h cout=%b zero=%b", $time, A, B, opcode, Y, cout,
zero);

      // Test case 1: Addition
      A = 4'h3; B = 4'h5; opcode = 3'b000; #10;
      // Test case 2: Subtraction
      A = 4'h7; B = 4'h2; opcode = 3'b001; #10;
      // Test case 3: AND
      A = 4'hF; B = 4'hA; opcode = 3'b010; #10;
      // Test case 4: OR
      A = 4'h6; B = 4'h9; opcode = 3'b011; #10;
      // Test case 5: XOR
      A = 4'hC; B = 4'h7; opcode = 3'b100; #10;
      // Test case 6: Zero output
      A = 4'h0; B = 4'h0; opcode = 3'b000; #10;

      // Add more tests as needed for your ALU functions

      $finish;
   end
endmodule
```

bcd_counter_tb.v

```
module bcd_counter_tb(

   );
endmodule
```

clock_divider_tb.v

```
module clock_divider_tb;
    reg clk, reset;
    wire clk_out;

    clock_divider uut (
        .clk(clk),
        .reset(reset),
        .clk_out(clk_out)
    );

    initial clk = 0;
    always #2 clk = ~clk; // Fast clock for sim

    initial begin
        reset = 1; #10;
        reset = 0; #10;
        #200; // Wait for clock_out toggles
        $finish;
    end

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

control_decoder_tb.v

```
module control_decoder_tb;
    reg [2:0] opcode;
    wire [3:0] ctrl;

    control_decoder uut (
        .opcode(opcode),
        .ctrl(ctrl)
    );

    initial begin
        $monitor("Time=%0t opcode=%b ctrl=%b", $time, opcode, ctrl);

        opcode = 3'b000; #10;
        opcode = 3'b001; #10;
        opcode = 3'b010; #10;
        opcode = 3'b011; #10;
        opcode = 3'b100; #10;
        opcode = 3'b101; #10;
        opcode = 3'b110; #10;
        opcode = 3'b111; #10;
        $finish;
```

```
    end
endmodule
```

seg7_scan_tb.v

```
module seg7_scan_tb;
    reg clk, reset;
    reg [15:0] data_in;
    wire [3:0] an;
    wire [6:0] seg;

    seg7_scan uut (
        .clk(clk),
        .reset(reset),
        .data_in(data_in),
        .an(an),
        .seg(seg)
    );

    initial clk = 0;
    always #5 clk = ~clk;

    initial begin
        reset = 1; data_in = 16'h1234; #10;
        reset = 0; #10;
        data_in = 16'hABCD; #20;
        data_in = 16'h0000; #20;
        data_in = 16'hFFFF; #20;
        $finish;
    end

    initial begin
        $monitor("Time=%0t data_in=%h an=%b seg=%b", $time, data_in, an, seg);
    end
endmodule
```

top_lab6_tb.v

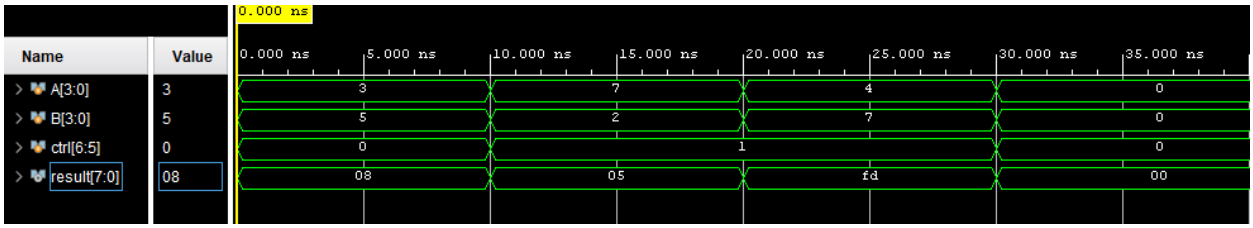
```
module top_lab6_tb;
    reg clk, reset, up, down;
    wire [6:0] seg;
    wire [3:0] an;

    top_lab6 uut (
        .clk(clk),
        .reset(reset),
        .up(up),
        .down(down),
        .seg(seg),
        .an(an)
```

```
    );

    initial clk = 0;
    always #5 clk = ~clk;

    initial begin
        reset = 1; up = 0; down = 0; #20;
        reset = 0; #10;
        // Test counting up
        up = 1; down = 0; #50;
        // Test counting down
        up = 0; down = 1; #50;
        // Both up and down inactive
        up = 0; down = 0; #20;
        $finish;
    end

    initial begin
        $monitor("Time=%0t seg=%b an=%b", $time, seg, an);
    end
endmodule
```

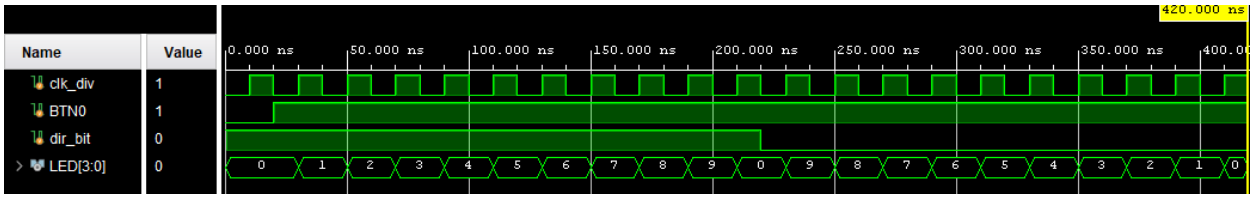
Synthesis and Implementation

Screenshots:

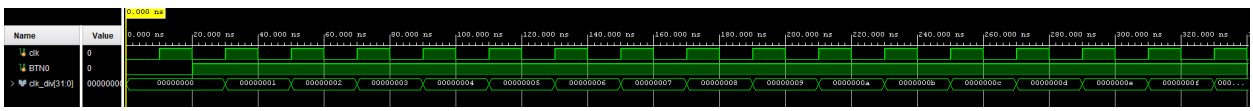
Alu



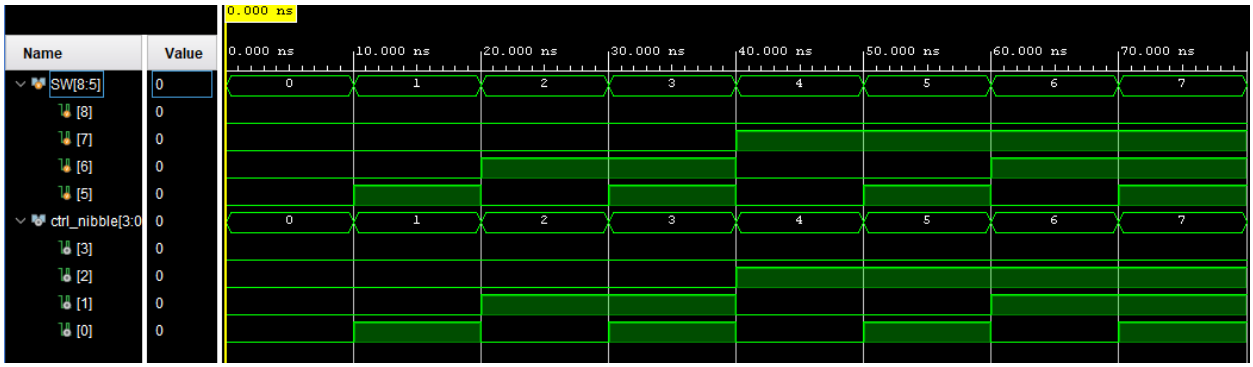
BCD_counter



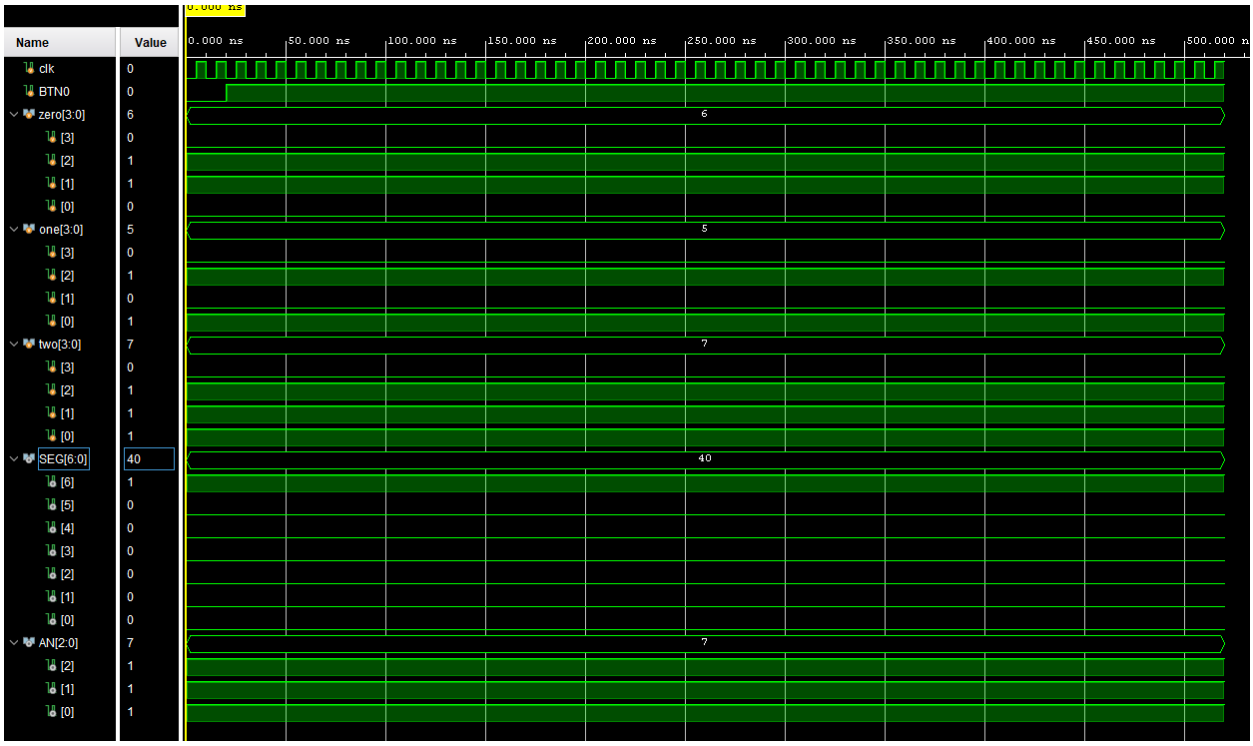
Clock_divider



Control_decoder



Seg7_scan



Reflections

The successful completion of this lab demonstrated the integration of counters, ALU operations, and display decoding in a cohesive FPGA-based system. By using switches to control counter direction and ALU operations, we were able to observe the effects in real-time through LEDs and the 7-segment display. By implementing testbenches and simulations we were able to verify correctness before programming the hardware. The project emphasized the importance of modular design and signal synchronization in digital systems. Overall, this lab enhanced practical skills in Verilog design and FPGA implementation.