Lab Report 6

Dr. Aly

Hector Garibay,

Justin Wong

Due Date: July 28,

2025

Introduction:

In this lab, we built a simple FPGA system that reads two decimal digits from slide switches, counts each digit up or down based on switch settings, performs an addition or subtraction on

those digits, and displays both the chosen operation and the result on LEDs and a multiplexed seven-segment display. We divided the board clock to a human-visible rate, implemented BCD counters for units and tens, created an ALU for basic arithmetic, decoded control signals, and scanned the display which we tied together in the top module.

Code and Explanation:

```
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'b0;
else
cnt <= cnt + 1;
end
assign clk_div = cnt[sel];

endmodule
```

The clock divider module uses a 32-bit register that increments on each rising edge of the 100 MHz board clock. By selecting one bit of this counter with a five-bit input, we generate a slower clock signal for downstream logic. Pushing the reset button asynchronously clears the counter to zero.

```
module bcd_counter(
    input clk_div,//divided clock
    input BTN0,// active low reset
    input dir_bit, // direction bit
    output reg [3:0] BCD // 4 bit BCD
    );
    always @(posedge clk_div or negedge BTN0) begin
        if (!BTN0) begin
        //resetbcd value
            BCD <= 4'd0;
```

```
        end else begin
            if (dir_bit) begin // count up
                if (BCD == 4'd9) begin // set up case for rollover
                    BCD <= 4'd0;
                end else begin
                    BCD <= BCD + 4'd1;
                end
            end else begin // count down
                if (BCD == 4'd0) begin // setup case for rollover counting down
                    BCD <= 4'd9;
                end else begin
                    BCD <= BCD - 4'd1;
                end
            end
        end
    end
endmodule
```

In bcd_counter, we take that divided clock and count a single decimal digit up or down. When the reset button is pressed, the counter snaps back to zero. On each rising edge of the divided clock, if the direction bit is high we add one rolling over from 9 back to 0, if it's low we subtract one, rolling under from 0 to 9.

```
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'b0;
endcase
end
endmodule
```

The ALU module accepts two four-bit inputs representing the ones and tens digits produced by our counters, plus a two-bit control code. When the code is 00 it adds the inputs; when it's 01 it

subtracts them; otherwise it drives the result to zero. We make the result eight bits wide so it can hold values from –15 up to +18 without overflow.

```
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'b0;
    else
      ctrl_nibble <= SW;
  end
endmodule
```

To display which operation is active, the control_decoder module samples the same switch bits used by the ALU on the divided clock. It latches those four bits into a register on every clock tick or resets them to zero on reset, and sends them out as the control nibble that we'll show on the third digit of the display.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
/
// Company:
// Engineer:
//
// Create Date: 07/26/2025 10:31:49 PM
// Design Name:
// Module Name: seg7_scan
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
```

```
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
/



module seg7_scan(
    input clk,                      // 100 MHz board clock
    input BTN0,                    // Active-low reset
    input [3:0] digit0,            // lower nibble result
    input [3:0] digit1,             //upper nibble result
    input [3:0] digit2,             // control nibble
    output reg [6:0] SEG,           // Segment lines a-g
    output reg [3:0] AN             // Anode lines (active low)
    );
    reg [15:0] refresh_counter = 0;
    always @(posedge clk or negedge BTN0) begin
    if (!BTN0)
    refresh_counter <= 16'd0;
    else
    refresh_counter <= refresh_counter + 1;
    end

    wire [1:0] sel = refresh_counter[15:14];

   reg [3:0] current_digit;
   always @(*) begin
   case (sel)
   2'd0: current_digit = digit0;
   2'd1: current_digit = digit1;
   2'd2: current_digit = digit2;
   default: current_digit = 4'd0;
   endcase
   end
       always @(*) begin
        case (sel)
            2'd0: AN = 4'b1110;  // enable digit0
            2'd1: AN = 4'b1101;  // enable digit1
            2'd2: AN = 4'b1011;
            default: AN = 4'b1111;
        endcase
    end
```

```
    // Decode the digit to 7-segment display pattern
    always @(*) begin
        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; // 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; // blank
        endcase
    end
endmodule
```

The seg7_scan module handles multiplexing three separate four-bit values onto a single four-digit seven-segment display. Inside, a 16-bit refresh counter runs on the full board clock; its two high bits select which of the three digits is active at any moment. A combinational block chooses which of the three inputs to send to the segment decoder, and another block drives the appropriate anode lines so that only one digit lights at a time. Cycling rapidly through all three makes them appear lit simultaneously.

```
module top_lab6(
input CLK,
input BTN0,
input  [8:0]  SW,         // SW[0-4]: clk_sel, SW[5-6]: ALU ctrl, SW[7]: units
dir, SW[8]: tens dir
output [7:0]  LED,
output [7:0] AN,
output [6:0] SEG
);
wire [31:0] cnt;
wire clk_out;
```

```
clock_divider u_div(
.clk(CLK),
.BTN0(!BTN0),
.sel(SW[4:0]),
.clk_div(clk_out),
.cnt(cnt)
);
 wire [3:0] unit_bcd, tens_bcd, control_display;
 wire roll;
  bcd_counter bcd_count_ones(
  .clk_div(clk_out),
  .BTN0(!BTN0),
  .dir_bit(SW[7]),
  .BCD(unit_bcd)
  );
  bcd_counter bcd_count_tens(
  .clk_div(clk_out),
  .BTN0(!BTN0),
  .dir_bit(SW[8]),
  .BCD(tens_bcd)
  );
wire [7:0] alu_result;
wire [3:0] ctrl_nibble = SW[8:5];
alu alu1(
.A(unit_bcd),
.B(tens_bcd),
.ctrl({SW[6], SW[5]}),
.result(alu_result)
);
control_decoder dec(
 .clk_div(clk_out),
 .BTN0(!BTN0),
.SW(SW[8:5]),
.ctrl_nibble(control_nibble)
);
wire [3:0] scan_AN;
    seg7_scan u_scan (
    .clk(CLK),
    .BTN0(!BTN0),
    .digit0(alu_result[3:0]),
    .digit1(alu_result[7:4]),
    .digit2(ctrl_nibble),
    .SEG(SEG),
    .AN(scan_AN)
```

```
    );
    // LEDs
  assign LED[3:0] = unit_bcd;
  assign LED[7:4] = tens_bcd;
  assign AN = { 5'b11111, scan_AN[2:0] };
endmodule
```

In the top_lab6 module, we tie everything together. Nine slide switches choose the clock-divider bit, the ALU operation code, and the up/down directions for units and tens counters. A single push-button serves as an active-low reset. We instantiate each submodule, wire their inputs and outputs, and finally route the two BCD values to LEDs and the three nibbles to the seven-segment scanner.

Utilizations:

Resource	Utilization	Available	Utilization %
LUT	32	63400	0.05
FF	56	126800	0.04
IO	34	210	16.19

Contributions:

Hector Garibay: Code/Report 50%

Justin Wong: Code/Report 50%