ECE3300L

Lab 5

By Justin Wong, Hector Garibay

Summer 2025

Report Date: July 21, 2025

Objective

The objective of this lab is to design and implement a two-digit BCD up/down counter on the Nexys A7 FPGA using Verilog-HDL. The system includes a clock divider with selectable speeds via 5 switches, direction control via BTN1, and reset via BTN0. The counter output is displayed on a multiplexed dual-digit 7-segment display.

Top Lab Code Snippet:

```
module top_lab5(
    input        CLK,
    input        BTN0,    // Reset
    input        BTN1,    // Dir
    input  [4:0] SW,
    output [6:0] SEG,     // Segments a-g
    output [7:0] AN,
    output [12:0] LED
    );
   // Instantiate clock divider
    wire [31:0] cnt;
    clock_divider u_div (
        .clk (CLK),
        .rst_n (!BTN0),   // Active-low reset
        .cnt (cnt)
    );

    // Instantiate 32x1 mux
    wire clk_out;
    mux32x1 u_mux (
        .cnt (cnt),
        .sel (SW),
        .clk_out (clk_out)
    );

 wire [3:0] unit_bcd, tens_bcd;
 wire [3:0] digit0, digit1;
assign digit0 = unit_bcd; // assign digit 0 to ones
assign digit1 = tens_bcd; // assign digit 1 to tens
  wire       unit_roll;
   // instantiate the units-digit counter
  bcd_up_down_counter u_unit (
    .clk_div   (clk_out),    // from mux32x1
    .rst_n     (!BTN0),      // reset button
    .dir       (!BTN1),        // up/down button
    .bcd_value (unit_bcd),   // output = units digit
    .roll      (unit_roll)   // pulse when units wraps 0 to 9
  );
```

7 Segment Scan Snippet:

```
    reg [15:0] refresh_counter = 0;
    wire sel;  // Which digit to display
    always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
    refresh_counter <= 16'd0;
    else
    refresh_counter <= refresh_counter + 1;
    end

    assign sel = refresh_counter[15];

   reg [3:0] current_digit;
   always @(*) begin
   case (sel)
   2'd0: current_digit = digit0;
   2'd1: current_digit = digit1;
   endcase
   end
       always @(*) begin
        case (sel)
            2'd0: an = 4'b10;  // enable digit0
            2'd1: an = 4'b01;  // enable digit1
            default: an = 4'b11;
        endcase
    end

    // Decode the digit to 7-segment display pattern
    always @(*) begin
        case (current_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;
            default: seg = 7'b1111111; // blank
        endcase
    end
endmodule
```

BCD_up_down Snippet:

```
module bcd_up_down_counter(
    input clk_div, //divided clock
    input rst_n, // reset
    input dir, // direction
    output reg [3:0] bcd_value, //4 bit bcd value
    output reg roll // check for rollover
    );
 always @(posedge clk_div or negedge rst_n) begin
        if (!rst_n) begin
        //reset roll and bcd value

            bcd_value <= 4'd0;
            roll      <= 1'b0;

        end else begin
            roll <= 1'b0;  // initialize roll to 0
            if (dir) begin // count up
                if (bcd_value == 4'd9) begin // set up case for rollover
                    bcd_value <= 4'd0;
                    roll      <= 1'b1;  // overflow
                end else begin
                    bcd_value <= bcd_value + 4'd1;
                end
            end else begin // count down
                if (bcd_value == 4'd0) begin // setup case for rollover counting down
                    bcd_value <= 4'd9;
                    roll      <= 1'b1;  // underflow
                end else begin
                    bcd_value <= bcd_value - 4'd1;
                end
            end
        end
    end
endmodule
```

Clock_divider and Mux Code:

```
module clock_divider(
    input clk,
    input rst_n,
    output reg [31:0] cnt // counter
    );
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        cnt <= 0; //reset count
    else
        cnt <= cnt + 1; // else increment count
    end
endmodule
```

```
module mux32x1 (
    input [31:0] cnt,
    input [4:0] sel,
    output clk_out
);
    assign clk_out = cnt[sel];  // Output the selected bit from D
endmodule
```

Test Bench Code Snippet

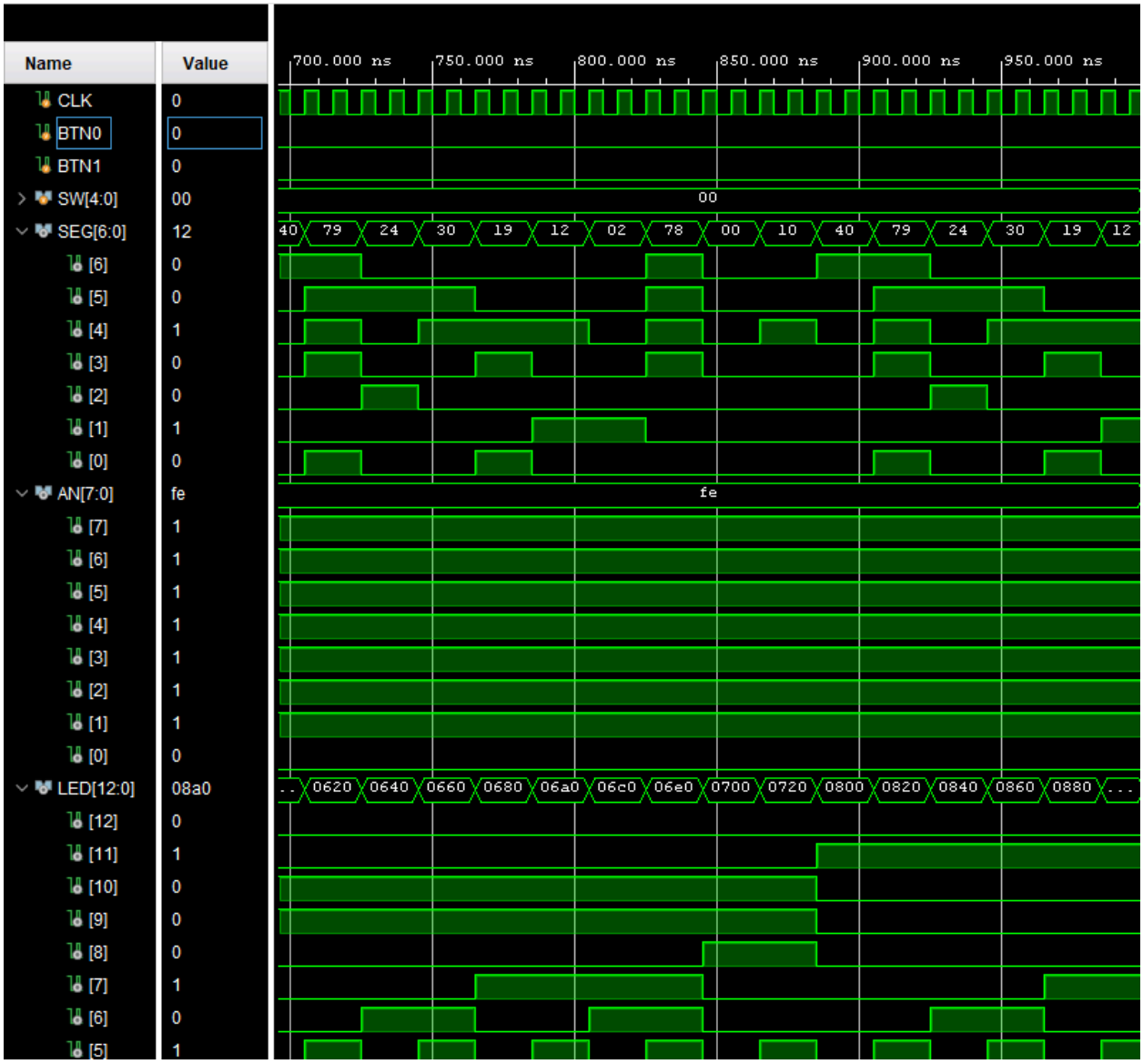
```
module top_lab5(
    input        CLK,
    input        BTN0,    // Reset
    input        BTN1,    // Dir
    input  [4:0] SW,
    output [6:0] SEG,     // Segments a-g
    output [7:0] AN,
    output [12:0] LED
    );
   // Instantiate clock divider
    wire [31:0] cnt;
    clock_divider u_div (
        .clk (CLK),
        .rst_n (!BTN0),   // Active-low reset
        .cnt (cnt)
    );

    // Instantiate 32x1 mux
    wire clk_out;
    mux32x1 u_mux (
        .cnt (cnt),
        .sel (SW),
        .clk_out (clk_out)
    );

 wire [3:0] unit_bcd, tens_bcd;
 wire [3:0] digit0, digit1;
assign digit0 = unit_bcd; // assign digit 0 to ones
assign digit1 = tens_bcd; // assign digit 1 to tens
  wire       unit_roll;
   // instantiate the units-digit counter
  bcd_up_down_counter u_unit (
    .clk_div   (clk_out),    // from mux32x1
    .rst_n     (!BTN0),      // reset button
    .dir       (!BTN1),        // up/down button
    .bcd_value (unit_bcd),   // output = units digit
    .roll      (unit_roll)   // pulse when units wraps 0 to 9
  );

  // instantiate the tens-digit counter
  bcd_up_down_counter u_tens (
    .clk_div   (unit_roll),  // advance on roll
    .rst_n     (!BTN0),
```

Simulation:


Name
Value
700.000 ns
750.000 ns
800.000 ns
850.000 ns
900.000 ns
950.000 ns
CLK 0
BTN0 0
BTN1 0
SW[4:0] 00
00
SEG[6:0] 12
40 79 24 30 19 12 02 78 00 10 40 79 24 30 19 12
[6] 0
[5] 0
[4] 1
[3] 0
[2] 0
[1] 1
[0] 0
AN[7:0] fe
fe
[7] 1
[6] 1
[5] 1
[4] 1
[3] 1
[2] 1
[1] 1
[0] 0
LED[12:0] 08a0
.. 0620 0640 0660 0680 06a0 06c0 06e0 0700 0720 0800 0820 0840 0860 0880 ...
[12] 0
[11] 1
[10] 0
[9] 0
[8] 0
[7] 1
[6] 0
[5] 1

Implementation:

Utilization Table:

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
top_lab5	26	57	4	22	26	36	1

Resource	Utilization	Available	Utilization %
LUT	26	63400	0.04
FF	57	126800	0.04
IO	36	210	17.14

Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.321 ns	Worst Hold Slack (WHS):	0.254 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:	48	Total Number of Endpoints:	48	Total Number of Endpoints:	49

All user specified timing constraints are met.

Video:

https://youtu.be/QSN8LI0lV3c

Contributions:

Justin Wong: XDC, Driver Code, testbench code, report. 50% for all.
Hector Garibay: XDC, Driver Code, testbench code, report. 50% for all.