Cal Poly Pomona

College of Engineering

California Polytechnic State University, Pomona

ECE3300L

Experiment #5

GROUP K
Hoang, Dalton - 016062800
Siu, Andy - 016205137

July 21st, 2025

Introduction:

In this lab, a two-digit Binary-Coded Decimal up/down counter was designed, implemented, and tested on the Nexys A7 FPGA board using Verilog-HDL. This experiment involved building a modular digital system that counts between 00 and 99 in either direction, based on user input, and displays the result on a dual-digit 7-segment display through time multiplexing. Directional control of the counter is managed using BTN1, while BTN0 serves as an active-low reset. A cascaded counter architecture was implemented, where the unit digit triggers the tens digit upon overflow or underflow, ensuring proper carry and borrow behavior during counting.

Design:

Bcd_counter.v: This module implements a tewo diit bcd counter by instantiating two of the dig_select modules or one and tens place. Each tick increments or decrements and generates a unit_roll signal for overflow and another for the tens digit only active when unit_roll is active.

```
module bcd_counter (
    input  wire clk,
    input  wire rst_n,
    input  wire tick,
    input  wire dir,          // 1 = up, 0 = down
    output wire [3:0] units,
    output wire [3:0] tens
);
    wire unit_roll; //carry signal from units to tens

     // generates 'unit_roll' when units digit overflows (up) or underflows (down)
    dig_select u_units (
        .clk(clk), .enable(tick), .dir(dir), .rst_n(rst_n),
        .value(units), .roll_out(unit_roll)
    );

    //counts when 'unit_roll' is active
    dig_select u_tens (
        .clk(clk), .enable(unit_roll), .dir(dir), .rst_n(rst_n),
        .value(tens), .roll_out()
    );
endmodule
```

Seg7_driver.v: This module displays tens and units bcd digits on a 7 segment display using a 16 bit counter to control which digit is shown and its corresponding segment patterns.

```
module seg7_driver (
    input  wire       clk,
    input  wire       rst_n,
    input  wire [3:0] units,
    input  wire [3:0] tens,
    output reg  [6:0] SEG,
    output reg  [1:0] AN
);

    //// 16-bit scan counter for display multiplexing
    reg [15:0] scan_ctr;
    always @(posedge clk or negedge rst_n)
        scan_ctr <= !rst_n ? 16'd0 : scan_ctr + 16'd1;

    // Use the MSB of scan_ctr to toggle between 'tens' and 'units' digits
    wire sel = scan_ctr[15];

     // 7-segment lookup table
    function [6:0] seg7(input [3:0] n);
        case (n)
            4'd0: seg7 = 7'b0000001;
            4'd1: seg7 = 7'b1001111;
            4'd2: seg7 = 7'b0010010;
            4'd3: seg7 = 7'b0000110;
            4'd4: seg7 = 7'b1001100;
            4'd5: seg7 = 7'b0100100;
            4'd6: seg7 = 7'b0100000;
            4'd7: seg7 = 7'b0001111;
            4'd8: seg7 = 7'b0000000;
            4'd9: seg7 = 7'b0001100;
            default: seg7 = 7'b1111111;
        endcase
    endfunction

    // // Combinational logic for digit selection and segment output
    always @(*) begin
        if (sel) begin
            SEG = seg7(tens);
            AN  = 2'b01;
        end else begin
            SEG = seg7(units);
            AN  = 2'b10;
        end
    end
endmodule
```

Clk_divider.v: This module generates a slower clock. It uses a 32 bit counter and a 5 bit select input to create a variable clock.

```
module clk_divider(
    input  wire clk,
    input  wire rst_n,
    input  wire [4:0] sw,
    output wire tick
);
    //32 bit counter, reset to 0 on rst_n low
    reg [31:0] ctr;
    always @(posedge clk or negedge rst_n)
        ctr <= !rst_n ? 32'd0 : ctr + 32'd1;

    wire [4:0] tap = 5'd31 - sw; //higher 'sw' values = faster tick, lower 'sw' values = slower tick

    //store previous value of ctr(tap)
    reg tap_d1;
    always @(posedge clk) tap_d1 <= ctr[tap];

    assign tick =  ctr[tap] & ~tap_d1;
endmodule
```

Top.v: This module integrates the other Verilog source files to create an up/down BCD counter where the switches control the counting speed, the buttons control the reset and counting direction, and the current count is shown on a seven-segment display and LEDs.

```
module top (
    input  wire        CLK, //sys clock input
    input  wire [4:0]  SW, //5 switches for clock divider control
    input  wire        BTN1, //counting rising edge
    input  wire        BTN0, //system reset
    output wire [7:0]  AN,  //anode control for 7 seg
    output wire [6:0]  SEG,
    output wire [12:0] LED //13 leds for BCD counter
);
    wire rst_n = ~BTN0;
    reg btn1_d; //button debouncing
    always @(posedge CLK) btn1_d <= BTN1;

    reg dir;
    always @(posedge CLK or negedge rst_n) begin
        if (!rst_n) dir <= 1'b1;                        //sets default counting direction to up
        else if ( BTN1 & ~btn1_d ) dir <= ~dir;    //toggle directino with BTN1
    end

    //generates tick input
    wire tick;
    clk_divider u_tick (
        .clk (CLK), .rst_n(rst_n), .sw(SW), .tick(tick)
    );

    wire [3:0] units, tens;
    bcd_counter u_counter (
        .clk(CLK), .rst_n(rst_n), .tick(tick), .dir(dir),
        .units(units), .tens(tens)
    );
```

```
    //calls units and tens digit onto 7 seg
    seg7_driver u_sevenseg (
        .clk(CLK), .rst_n(rst_n),
        .units(units), .tens(tens),
        .SEG(SEG), .AN(AN[1:0])
    );
    assign AN[7:2] = 6'b111111;

    //led outputs
    assign LED[3:0]  = units;
    assign LED[7:4]  = tens;
    assign LED[12:8] = {3'b000, SW};
endmodule
```

Dig_select.v: This module defines a single digit bcd counter that increments or decrements its value based on the clock pulse, enable state, and inc or dec input. It also rollovers when reaching 0 or 9.

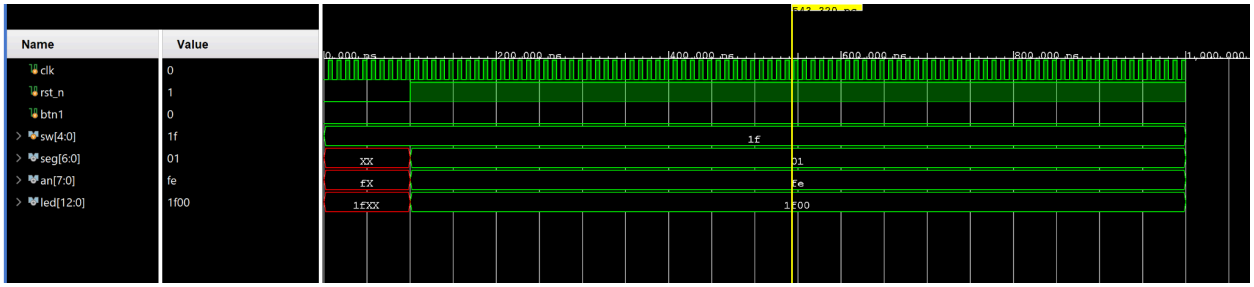
```
module dig_select (
    input  wire clk,
    input  wire enable,
    input  wire dir,
    input  wire rst_n,
    output reg  [3:0] value,
    output reg        roll_out
);

    // Detect if current value is at maximum (9) or minimum (0)
    wire at_max = value == 4'd9;
    wire at_min = value == 4'd0;

      // Sequential logic: counter operation
    always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin //set value to 0 and clear roll_out
            value    <= 4'd0;
            roll_out <= 1'b0;
        end else begin //no rollover unless counting triggers it
            roll_out <= 1'b0;
            if (enable) begin
                if (dir) begin  // Counting up
                    value    <= at_max ? 4'd0   : value + 4'd1;
                    roll_out <=  at_max;
                end else begin   // Counting down
                    value    <= at_min ? 4'd9   : value - 4'd1;
                    roll_out <=  at_min;
                end
            end
        end
    end
endmodule
```

Top_tb.v: This testbench initializes inputs for the top module, simulates a clock, uses a reset, sets the switches for a fast tick, counts up, then toggles the direction by simulating a button pres, and inputs a faster 7 segment display scan.

```
module top_tb();

    reg clk;
    reg rst_n;
    reg btn1;
    reg [4:0] sw;

    wire [6:0] seg;
    wire [7:0] an;
    wire [12:0] led;

    // Instantiate DUT
    top uut (
        .CLK(clk),
        .SW(sw),
        .BTN1(btn1),
        .BTN0(rst_n),
        .AN(an),
        .SEG(seg),
        .LED(led)
    );

    // Generate 100 MHz clock
    initial clk = 0;
    always #5 clk = ~clk;

    initial begin
        // Initialize
        rst_n = 0;
        btn1 = 0;
        sw = 5'b11111; // Fastest tick for testbench

        #100;
        rst_n = 1;

        // Run for some time counting up
        #50000;

        // Toggle direction (simulate BTN1 press)
        btn1 = 1; #10; btn1 = 0;

        #50000;

        $stop;
    end

    // Speed up display scan for simulation ONLY
    initial begin
        force uut.u_sevenseg.scan_ctr[15] = uut.u_sevenseg.scan_ctr[3];
    end

endmodule
```

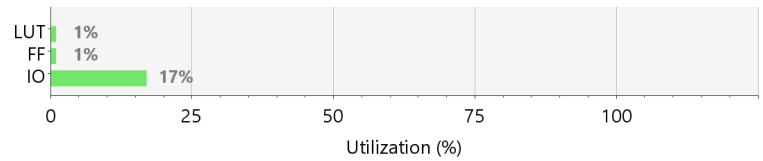
Simulation:

This waveform shows the behavior of the BCD up/down counter from an initial reset. When the reset is released and a fast SW input of all ones is applied, the counter starts. The outputs change in response to the counter's correct operation, confirming the intended inputs and outputs.



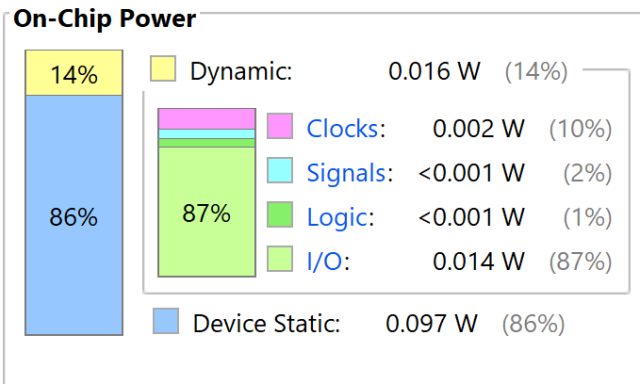
Implementation:

Resource	Utilization	Available	Utilization %
LUT	33	63400	0.05
FF	60	126800	0.05
IO	36	210	17.14



Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.113 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**25.5°C**



Report Cell Usage:

	Cell	Count
1	BUFG	1
2	CARRY4	8
3	LUT1	4
4	LUT3	3
5	LUT5	8
6	LUT6	16
7	MUXF7	4
8	MUXF8	2
9	FDRE	46
10	IBUF	8
11	OBUF	24

Video Link: https://www.youtube.com/shorts/78EFA4HC9S0

Contributions:

Andy Siu: 50% bcdcounter, clk_divider, top, implementation, report

Dalton Hoang: 50% dig_select, seg7_driver, testbench, simulation, demo, report