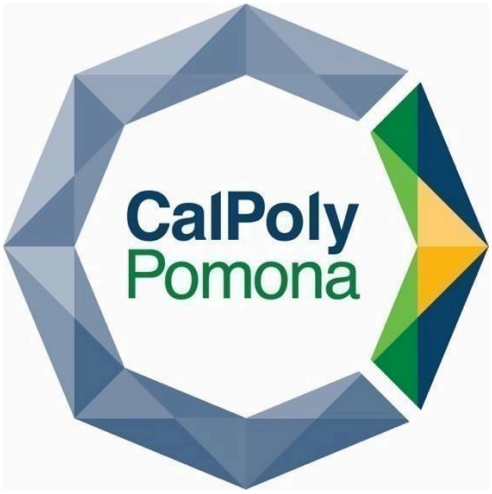


California Polytechnic State University Pomona

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

Digital Circuit Design Verilog

ECE 3300L

Report #5

Prepared by

Heba Hafez 017353323

Sean Wygant 017376658

Group Y

Mohamed Aly

July 15, 2025

Objective: Using a 32-bit counter clock divider, 32x1 Mux, and 5 SW speed selects, design a two digit BCD up/down counter in Verilog HDL. Have BTN0 as a reset and BTN1 as an up/down direction counter.

Code and Explanation:

Toggle Flip-Flop:

```
module toggle_switch(
input clk,
input rst,
input btn_raw,
output reg state
);
wire btn_clean;
reg btn_prev;
debounce db (.clk(clk), .btn_in(btn_raw), .btn_clean(btn_clean));
always @(posedge clk) begin
if (rst) begin
state <= 0;
btn_prev <= 0;
end else begin
if (btn_clean && !btn_prev)
state <= ~state;
btn_prev <= btn_clean;
end
end
endmodule
```

Similar to lab 3, since we were implementing multiplexers and buttons, we added the toggle flip flop code in to fix any hiccups.

Clock Divider:

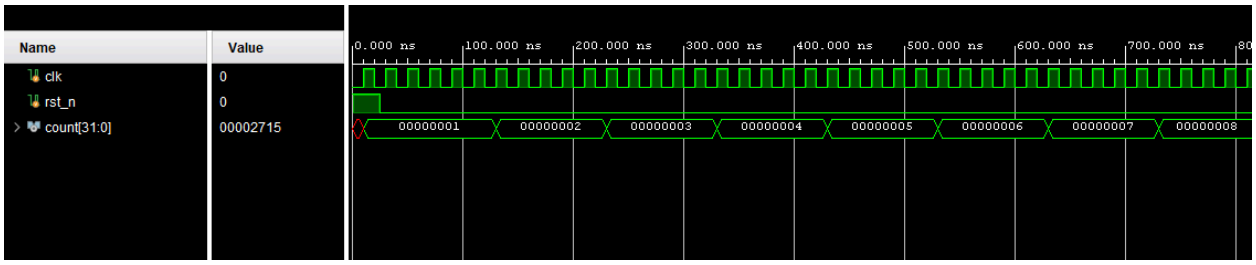
```
module clock_divider(
    input clk,
    input rst_n,
    output reg [31:0] count
    );
        always @ (posedge clk) begin
            if (rst_n) begin // if reset is high
                count = 32'b0; // count is 0
            end
            else
                #100;
                count = count + 1'b1; // increment counter
        end
endmodule
```

On the rising edge, the block triggers, and if the reset is high, the count is set to 0 and if the reset is low, the count is incremented by one . By doing so we create a 32 bit counter that increases per every clock cycle

Clock Divider Tb:

For the clock divider test bench, we ran a 100MHz clock cycle alternating the reset to display the simulation time, reset state, and counter value to verify that our clock divider is properly running.

```
Time resolution is 1 ps
Time=0 , rst_n=1 , count=xxxxxxxx
Time=10000 , rst_n=1 , count=00000001
Time=25000 , rst_n=0 , count=00000001
Time=130000 , rst_n=0 , count=00000002
Time=230000 , rst_n=0 , count=00000003
Time=330000 , rst_n=0 , count=00000004
Time=430000 , rst_n=0 , count=00000005
$finish called at time : 525 ns : File "C:/Users/Sean/Documents/Summer 25 CPP/ECE3300L/Lab 5/Lab 5.srcs/sim_1/new/bcd_tb.v" Line 41
```



Mux:

With the 32-bit count, one bit from the count is selected based on the 5-bit selector sel and outputs through clk_out. By using count[31-sel], the most significant bit is chosen when sel is -0 and count[31] and vice versa.

```
module mux32x1(
    input [31:0] count,
    input [4:0] sel,
    output clk_out
);
    assign clk_out = count[(31-sel)];
endmodule
```

Mux Tb:

In order to test our multiplexer we run a clock cycle to steadily increment the count and select in order to view the changes of every bit. As time passes, we can view the waveform to see the speed increasing as the bits change.

```
module mux32x1_tb;

    reg [31:0] count;
    reg [4:0] sel;
    reg clk;
    wire clk_out;
    integer i, j;

    mux32x1 uut (
        .count(count),
        .sel(sel),
        .clk_out(clk_out)
    );

initial begin
clk = 0;
count = 31'b0;
sel = 0;
forever #10
clk=~clk;
end

 always@(posedge clk)begin
      //  for (i = 0; i < 64; i = i + 1) begin
            count = count+1;
          //  sel = i[4:0];
            #5;
            for (j=0; j<32; j=j+1) begin
         //   count = 1'b1<<j;
         sel = sel+1;
            #100;
            $monitor("sel = %b , count[%b] = %b , clk_out = %b", sel, j, count, clk_out);
            #5;
        end

        $finish;
```

BCD Up Down Counter:

Using various if else statements we created different unit tests to determine whether the ones place or the tenths place will light which number. We also checked for when there was a carry in order to send it to the next place .

```
module bcd_up_down_counter(
    input wire clk_div,
    input wire rst_n,
    input dir, //1 up 0 down
    output reg [3:0] digit,
    output reg rollover
);

    always@(posedge clk_div or posedge rst_n) begin
        if ((dir == 0 && digit == 9) || (dir == 1 && digit == 0)) rollover = 1;
        #5;
        if (rst_n) begin
            #1;
            digit = 4'b0;
            rollover = 0;
            #1;
        end
        else begin
            if (dir == 0) begin
                if (digit == 9) begin
                digit = 0;
                #1;
                end
                else begin
                digit = digit + 1;
                rollover = 0;
                #1;
                end
            end
            if (dir == 1) begin
                if (digit == 0) begin
                digit = 9;
                #1;
                end
                else begin
                digit = digit - 1;
                rollover = 0;
```

BCD Up-Down TB:

The test bench ran through a clock cycle and various time frames to check the different values and outputs coming from the BCD. This ensured our if else statements were running through properly how we needed it to, to transition between the units.

```
module bcd_tb(
    );
reg clk;
reg rst_n;
wire [31:0] count;

clock_divider uut( .clk(clk), .rst_n(rst_n), .count(count));

initial begin
clk = 0;
forever #10 clk=~clk;
end

initial begin
        rst_n = 1;
        #25;
        rst_n = 0;
        #500;
        $finish;
    end
    initial begin
    $monitor("Time=%0t , rst_n=%b , count=%h", $time, rst_n, count);
end
endmodule
```

Seg7 Driver:

The driver had all the case statements for the 7 segment displays to properly light all numbers. Based on the edge of the run the tmp would be set to 0 or increment to determine whether to apply case s or not .

```
module seg7_driver(
    input clk,
    input rst_n,
    input [7:0] bits,
    output reg [6:0] Cnode,
    output dp,
    output [7:0] AN
    );
    wire [31:0] bits2;
    reg [19:0] tmp;
    reg [3:0] digit;
    assign dp = 1'b1;
    assign bits2 = ~0;
    always@(digit)
    case(digit)
        4'd0:Cnode=7'b0000001; 4'd1: Cnode=7'b1001111; 4'd2: Cnode=7'b0010010;
        4'd3:Cnode=7'b0000110; 4'd4: Cnode=7'b1001100; 4'd5: Cnode=7'b0100100;
        4'd6:Cnode=7'b0100000; 4'd7: Cnode=7'b0001111; 4'd8: Cnode=7'b0000000;
        4'd9:Cnode=7'b0001100; 4'd10:Cnode=7'b0001000;4'd11:Cnode=7'b1100000;
        4'd12:Cnode=7'b0110001;4'd13:Cnode=7'b1000010;4'd14:Cnode=7'b0110000;
        4'd15:Cnode=7'b1111111;default: Cnode=7'b1111111;
    endcase
    always@(posedge clk) begin
    if(rst_n)
        tmp<=0;
    else
        tmp<=tmp+1;
    end
    wire [2:0] s = tmp[19:17];
    always@(s, bits)
    case (s)
        3'd0:digit=bits[3:0]; 3'd1:digit=bits[7:4];
        3'd2:digit=bits2[11:8]; 3'd3:digit=bits2[15:12];
        3'd4:digit=bits2[19:16];3'd5:digit=bits2[23:20];
        3'd6:digit=bits2[27:24];3'd7:digit=bits2[31:28];
        default:digit=4'b0000;
```

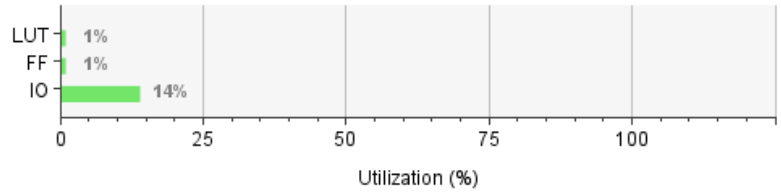
Top-Lab5:

The Top Lab wraps together all of our files to integrate them together.

```
module top_lab5(
    input clk,
    input rst_n,
    input [4:0] SW,
    input sel,
    output [7:0] AN,
    output [6:0] SEG,
    output [4:0] LED,
    output dp
    );

    wire [31:0] count;
    wire clk_out;
    wire [3:0] ones;
    wire [3:0] tens;
    wire [7:0] bits;
    wire dir;
    wire rollover;
    assign LED = SW;
    assign bits[7:0] = {tens[3:0],ones[3:0]};

        toggle_switch dir_toggle (.clk(clk), .rst(rst_n), .btn_raw(sel), .state(dir));

        clock_divider clock (.clk(clk), .rst_n(rst_n), .count(count));

        mux32x1 mux32 (.sel(SW), .count(count), .clk_out(clk_out));

        bcd_up_down_counter one (.clk_div(clk_out), .dir(dir), .rst_n(rst_n), .digit(ones), .rollover(rollover));
        bcd_up_down_counter ten (.clk_div(rollover), .dir(dir), .rst_n(rst_n), .digit(tens));


        seg7_driver seg7_driver (.clk(clk), .rst_n(rst_n), .bits(bits), .Cnode(SEG), .AN(AN), .dp(dp));

endmodule
```

Vivado Utilizations (LUTs, FFs, Power):

Resource	Utilization	Available	Utilization %
LUT	67	63400	0.11
FF	67	126800	0.05
IO	29	210	13.81



1.1 On-Chip Components

On-Chip	Power (W)	Used	Available	Utilization (%)
Clocks	<0.001	3	---	---
Slice Logic	<0.001	176	---	---
LUT as Logic	<0.001	67	63400	0.11
Register	<0.001	67	126800	0.05
CARRY4	<0.001	13	15850	0.08
F7/F8 Muxes	<0.001	11	63400	0.02
Others	0.000	10	---	---
Signals	<0.001	105	---	---
I/O	0.015	29	210	13.81
Static Power	0.097			
Total	0.114			

Video Link:
https://youtu.be/uqiD5FtZ4Ss

Reflections:
Being the first time we wrote the full code beginning to end we ran into some trouble. While we got the mux working immediately we had some trouble with the bcd up down counter. We were able to get the code to run to display incrementing and decrementing numbers however when it came to switching from ones to tens there were hiccups as we needed to use the button.

Partner Contributions:
The lab was worked on in person so work was 50/50. Some of the modules were written by Heba and some were written by Sean, alternating each other's code to debug. The test benches were

split by who got each module to run properly. The code summaries were worked on by Heba, and the simulation pictures were input and finalized by Sean.