

# ELC 2137 Lab 10: 7-segment Display with Time-Division Multiplexing

My Nguyen

October 30, 2020

## Summary

This lab's purpose is to get a better understanding sequential circuit, develop a parameterized counter-timer module, and implement a 4-digit display using counter module. First part is to create a counter module that uses parameter to generate counter module with different number of bit for input and output. Then, create a wrapper module to test the counter module using different N parameters so the 4-digit display is smooth. Next, create show\_2c module, which which an 8-bit 2's complement and convert it to a sign bit. Afterward, create calc\_lab10 and import basys3.xdc with correct constraints as a top-level module to do on-board testing.

## Simulation Waveform

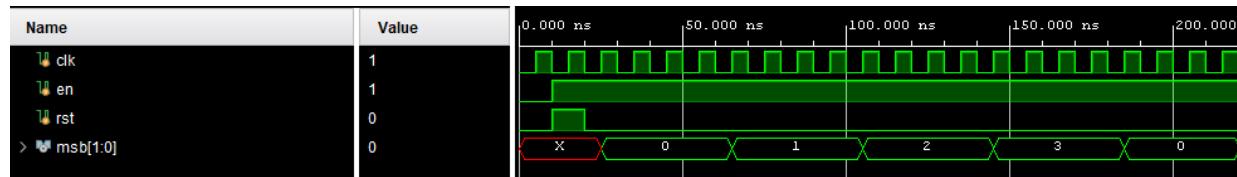


Figure 1: Counter ERT

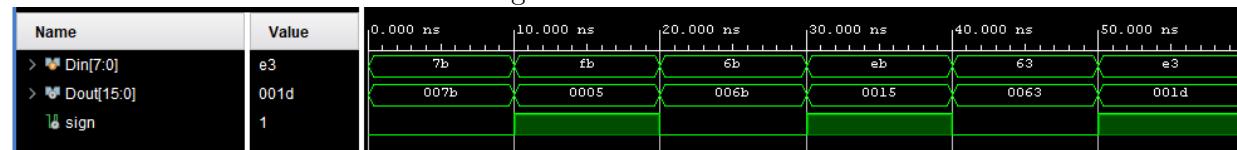


Figure 2: show\_2c ERT

## Code

Listing 1: Counter Implementation

```
module counter #(parameter N=1)
    input clk, rst, en,
    output [1:0] msb
);
reg [N-1:0] Qnext, Qreg;
always @(posedge clk, posedge rst)
begin
    if(rst)
        Qreg <= 0;
```

```

        else
            Qreg <= Qnext;
    end

    always @*
    begin
        if (en)
            Qnext = Qreg+1;
        else
            Qnext = Qreg;
    end

    assign msb = Qreg[N-1:N-2];
endmodule

```

Listing 2: Counter Test Bench

```

module counter_test();
    reg clk , en , rst ;
    wire [1:0] msb;
    counter #(N(4)) counter0(
        .clk(clk),
        .en(en),
        .rst(rst),
        .msb(msb));

    always begin
        clk = ~clk ; #5;
    end

    initial begin
        clk = 0; en = 0; rst = 0; #10;
        rst = 1; en = 1; #10;
        rst = 0; #200
        $finish ;
    end
endmodule

```

Listing 3: show\_2c Implementation

```

module show_2c(
    input [7:0] Din,
    output reg [15:0] Dout,
    output sign
);
    assign sign = Din[7];

    always @*
    begin
        if (sign)
            Dout = ~{{8{1'b1}},Din} + 1'b1;
        else
            Dout = {{8{1'b0}},Din};
    end

```

```
endmodule
```

Listing 4: show\_2c Test Bench

```
module show_2c_test();
    reg [7:0] Din;
    wire [15:0] Dout;
    wire sign;

    show_2c dut(
        .Din(Din),
        .Dout(Dout),
        .sign(sign)
    );

    initial begin
        Din = 8'b01111011; #10;
        Din = 8'b11111011; #10;
        Din = 8'b01101011; #10;
        Din = 8'b11101011; #10;
        Din = 8'b01100011; #10;
        Din = 8'b11100011; #10;
        Din = 8'b00010100; #10;
        Din = 8'b01111010; #10;
        $finish;
    end
endmodule
```

Listing 5: Wrapper Implementation

```
module wrapper(
    input clk, btnC,
    output [6:0] seg,
    output [3:0] an,
    output dp
);

    wire [1:0] counter_sseg4;
    counter #(N(21)) c(
        .clk(clk),
        .en(1'b1),
        .rst(btC),
        .msb(counter_sseg4)
    );

    sseg4 dut(
        .data(15'hAB),
        .hex_dec(1'b0),
        .sign(1'b0),
        .digit_sel(counter_sseg4),
        .seg(seg),
        .an(an),
        .dp(dp)
    );

```

```
endmodule
```

---

Listing 6: Top-Level Implementation

---

```
module calc_lab10(
    input [15:0] sw,
    input btnC, btnD, btnU, clk,
    output [6:0] seg,
    output [3:0] an,
    output dp
);

wire [15:0] tmpLED;
top_lab9 dut(
    .sw(sw[11:0]),
    .btnU(btnU),
    .btnD(btnD),
    .btnC(btnC),
    .clk(clk),
    .led(tmpLED)
);

wire [15:0] Dout_sseg4;
wire sign_sseg4;

show_2c show0(
    .Din(tmpLED[15:8]),
    .Dout(Dout_sseg4),
    .sign(sign_sseg4)
);

wire [1:0] msb_sseg4;
counter #(N(21)) c(
    .clk(clk),
    .en(1'b1),
    .rst(btnC),
    .msb(msb_sseg4)
);

sseg4 main(
    .data(Dout_sseg4),
    .sign(sign_sseg4),
    .hex_dec(sw[15]),
    .digit_sel(msb_sseg4),
    .seg(seg),
    .dp(dp),
    .an(an)
);

endmodule
```

---

## Pictures

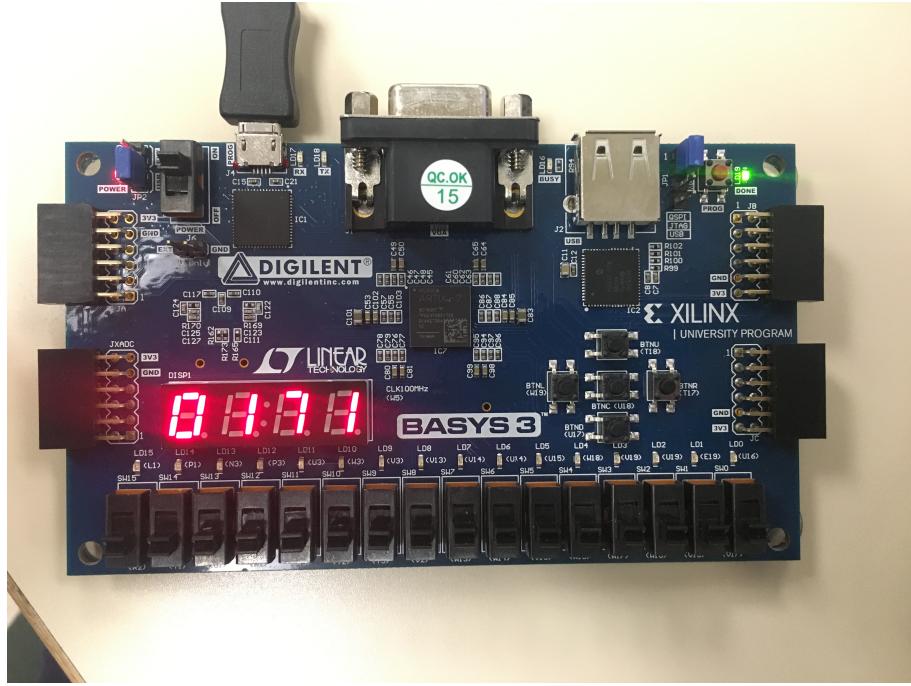


Figure 3: Displaying 4 digits with  $N = 20$

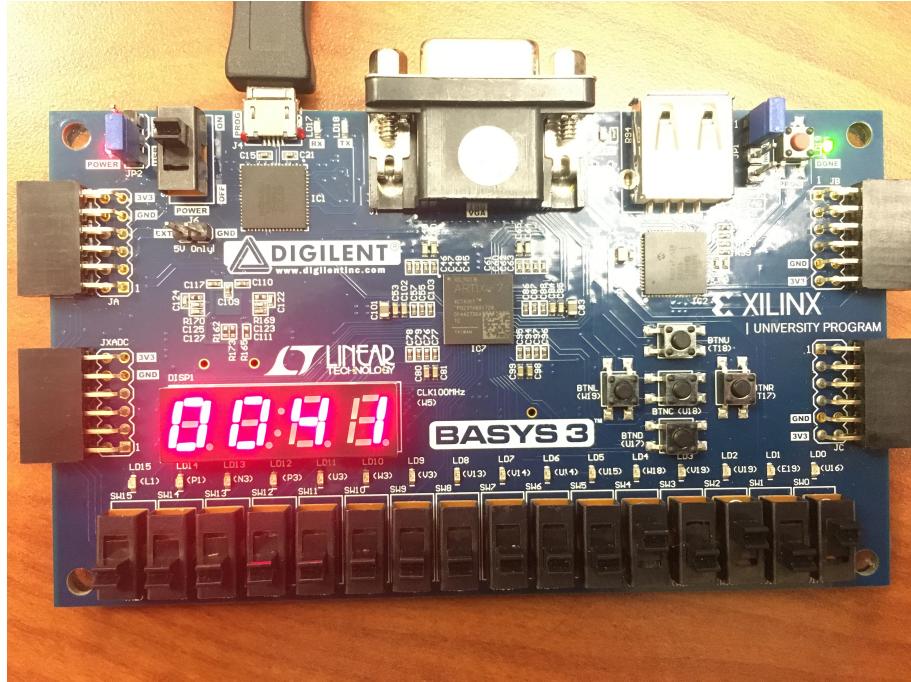


Figure 4: Adding hexadecimal 14 and 15 and outputting a decimal

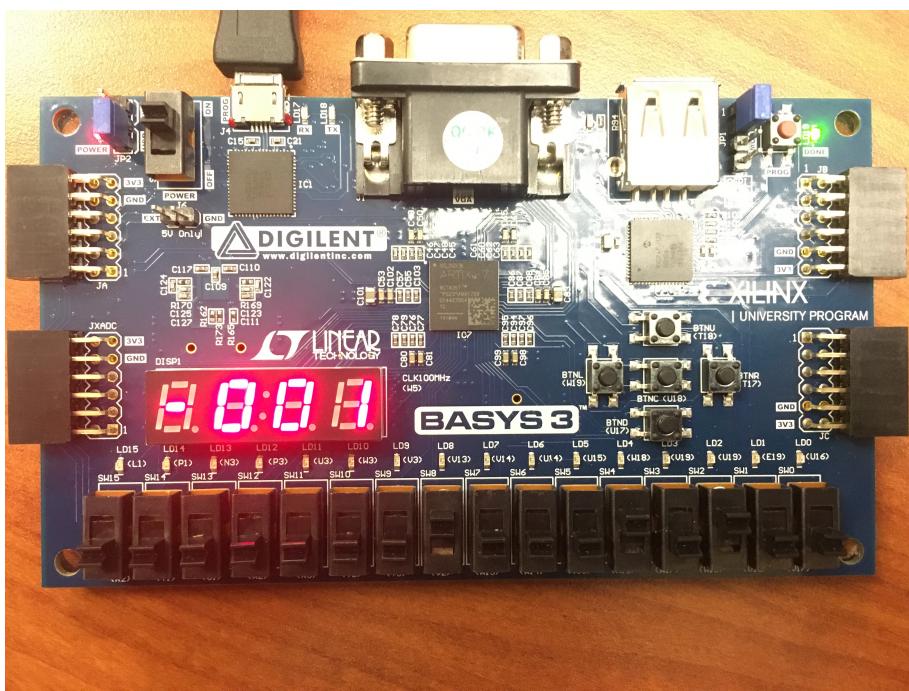


Figure 5: Subtracting hexadecimal 14 to 15 and outputting a decimal