Lab 5

Student ID: F10915108 Name: Gerardo Sigfredo Fisch Paredes

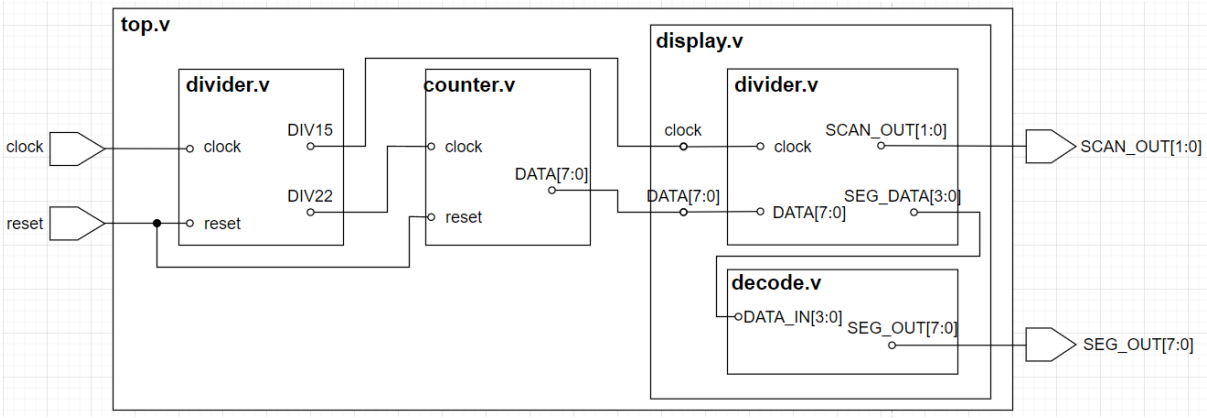
1. Topic: Seven-Segment Counter Application

a) Task I: 00 to 99 up-counter and display in 4-digit scanning seven segments (use clock about 10HZ).

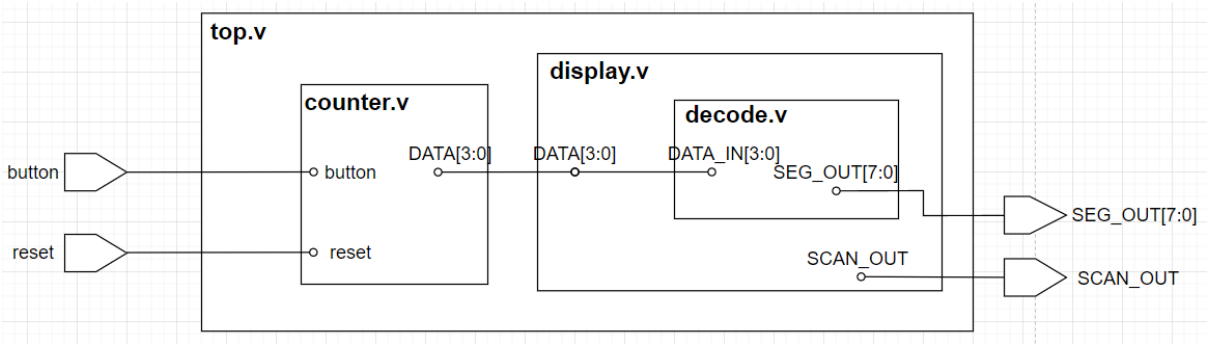
b) Task II: 0 to 9 up-counter with push button and display in 4-digit scanning seven segments

2. Circuit Block Diagram:

a) Task I



b) Task II



3. Code:

a) Task I

The program had the following structure

- top.v
  - divider.v
    - frequency_divider_by_2.v
  - counter.v
  - display.v
    - seg_scan.v
    - decode.v

i) Code:

- top.v

```
module top(reset, clock, SCAN_OUT, SEG_OUT);
    input reset, clock;
    output [1:0] SCAN_OUT;      // used to determine which position to display
    output [7:0] SEG_OUT;       // used to determine which digit to display
    wire [7:0] temp_data;
    wire DIV22, DIV15;

    divider div(reset, clock, DIV22, DIV15);
    counter count(reset, DIV22, temp_data);
    display disp(DIV15, temp_data, SCAN_OUT, SEG_OUT);
endmodule
```

- divider.v

```
module divider(reset, clock, DIV22, DIV15);
    input reset, clock;
    output reg DIV22, DIV15;
    wire [22:0]DIV;

    // The frequency of clock is divided by 2, 22 times
    frequency_divider_by_2 FDb2_0(reset, clock, DIV[0]);
    frequency_divider_by_2 FDb2_1(reset, DIV[0], DIV[1]);
    frequency_divider_by_2 FDb2_2(reset, DIV[1], DIV[2]);
    frequency_divider_by_2 FDb2_3(reset, DIV[2], DIV[3]);
    frequency_divider_by_2 FDb2_4(reset, DIV[3], DIV[4]);
    frequency_divider_by_2 FDb2_5(reset, DIV[4], DIV[5]);
    frequency_divider_by_2 FDb2_6(reset, DIV[5], DIV[6]);
    frequency_divider_by_2 FDb2_7(reset, DIV[6], DIV[7]);
    frequency_divider_by_2 FDb2_8(reset, DIV[7], DIV[8]);
    frequency_divider_by_2 FDb2_9(reset, DIV[8], DIV[9]);
    frequency_divider_by_2 FDb2_10(reset, DIV[9], DIV[10]);
    frequency_divider_by_2 FDb2_11(reset, DIV[10], DIV[11]);
    frequency_divider_by_2 FDb2_12(reset, DIV[11], DIV[12]);
    frequency_divider_by_2 FDb2_13(reset, DIV[12], DIV[13]);
    frequency_divider_by_2 FDb2_14(reset, DIV[13], DIV[14]);
    frequency_divider_by_2 FDb2_15(reset, DIV[14], DIV[15]);
    frequency_divider_by_2 FDb2_16(reset, DIV[15], DIV[16]);
    frequency_divider_by_2 FDb2_17(reset, DIV[16], DIV[17]);
    frequency_divider_by_2 FDb2_18(reset, DIV[17], DIV[18]);
    frequency_divider_by_2 FDb2_19(reset, DIV[18], DIV[19]);
    frequency_divider_by_2 FDb2_20(reset, DIV[19], DIV[20]);
    frequency_divider_by_2 FDb2_21(reset, DIV[20], DIV[21]);
    frequency_divider_by_2 FDb2_22(reset, DIV[21], DIV[22]);

    always @(posedge clock) begin
        DIV15 <= DIV[15];
        DIV22 <= DIV[22];
    end
```

- frequency_divider_by_2.v

```
module frequency_divider_by_2 (reset, clock, out_clock);
    output reg out_clock;
    input clock;
    input reset;
    initial out_clock = 0;

    always @(posedge clock)
    begin
    if (reset)                  // if the reset button is pressed
         out_clock <= 1'b0;
    else
        // the value of out_clock changes every time that clock goes from 0 to 1
        // Therefore, its freq is half of clock
         out_clock <= ~ out_clock;
    end
endmodule
```

- counter.v

```
module counter(reset, clock, DATA);
    input clock, reset;
    output reg [7:0] DATA;
    initial DATA = 0;

    always @(posedge clock or posedge  reset) begin
        if (reset) begin        // if the reset button is pressed, DATA = 0
            DATA <= 0;
        end
        else begin
            if (DATA[7:4] >= 9) begin   // if DATA >= 90
                if (DATA[3:0] >= 9) begin   // if DATA >= 99, then DATA = 0
                    DATA <= 0;
                end
                else begin                  // if 90 <= DATA < 99, then DATA++
                    DATA <= DATA + 1;
                end
            end
            else begin                  // if DATA < 90
                if (DATA[3:0] >= 9) begin       // if DATA == X9,
                    DATA[7:4] <= DATA[7:4] + 1; // then DATA_decimal = X + 1
                    DATA[3:0] <= 0;             // and DATA_unit = 0
                end
                else begin
                    DATA <= DATA + 1;
                end
            end
        end
    end
endmodule
```

- display.v

```
module display(clock, DATA, SCAN_OUT, SEG_OUT);
    input clock;
    input [7:0] DATA;           // number that will be displayed
    output [1:0] SCAN_OUT;      // used to determine which position to display
    output [7:0] SEG_OUT;       // used to determine which digit to display
    wire[7:0] temp;

    seg_scan SS(clock, DATA, SCAN_OUT, temp);
    decode DEC(temp, SEG_OUT);
endmodule
```

- seg_scan.v

```
module seg_scan(clock, DATA, SCAN_OUT, SEG_DATA);
    input clock;
    input [7:0] DATA;           // number that will be displayed
    output [1:0] SCAN_OUT;      // used to determine which position to display
    output [3:0] SEG_DATA;      // used to determine which digit to display

    assign SEG_DATA = (clock) ? DATA[3:0] : DATA[7:4];
    assign SCAN_OUT = (clock) ? 2'b01 : 2'b10;
endmodule
```

- decode.v

```
module decode(DATA_IN, SEG_OUT);
    input [3:0] DATA_IN;            // used to show which digit to display
    output [7:0] SEG_OUT;           // store the pattern of the digit to be displayed

    wire [7:0] number [15:0];       // number[i] = byte that displays i

    assign number[0] = 8'b10111111;
    assign number[1] = 8'b10000110;
    assign number[2] = 8'b11011011;
    assign number[3] = 8'b11001111;
    assign number[4] = 8'b11100110;
    assign number[5] = 8'b11101101;
    assign number[6] = 8'b11111101;
    assign number[7] = 8'b10100111;
    assign number[8] = 8'b11111111;
    assign number[9] = 8'b11101111;
    assign number[10] = 8'b11110111;
    assign number[11] = 8'b11111100;
    assign number[12] = 8'b11011000;
    assign number[13] = 8'b11011110;
    assign number[14] = 8'b11111001;
    assign number[15] = 8'b11110001;

    assign SEG_OUT = number[DATA_IN];
endmodule
```

ii) Test Bench:

- tb_top.v

```
module tb_top;
    // Inputs
    reg reset, clock;
    // Outputs
    wire [1:0] SCAN_OUT;
    wire [7:0] SEG_OUT;

    // Initial the Unit Under Test
    top uut(reset, clock, SCAN_OUT, SEG_OUT);

    initial begin
        reset = 0;
        clock = 0;
        forever begin
            #1;                 // wait 1ns;
            clock = ~clock;
         end
    end
endmodule
```

- tb_divider.v

```
module tb_divider;
    // Inputs
    reg reset, clock;
    // Output
    wire DIV22, DIV15;

    // Initial the Unit Under Test
    divider uut(reset, clock, DIV22, DIV15);

    initial begin
        reset = 0;
        clock = 0;
        forever begin
            #1;                 // wait 1ns;
            clock = ~clock;
         end
    end
endmodule
```

- tb_display.v

```
module tb_display;
    // Inputs
    reg clock;
    reg [7:0] DATA;
    // Outputs
    wire [1:0] SCAN_OUT;
    wire [7:0] SEG_OUT;

    integer i,j,k;
    // Initial the Unit Under Test
    display uut(clock, DATA, SCAN_OUT, SEG_OUT);

    initial begin
        for(i=0; i<10; i=i+1)           // DATA_decimal goes from 0 to 9
        begin
            DATA[7:4] = i;
            for(j=0; j<10; j=j+1)       // DATA_unit goes from 0 to 9
            begin
                DATA[3:0] = j;
                for(k=0; k<2; k=k+1)    // clock goes from 0 to 1
                begin
                    clock = k;
                    #1;                 // wait 1ns
                end
            end
        end
        $finish;
     end
endmodule
```

- tb_counter.v

```
module tb_counter;
    // Inputs
    reg reset, clock;
    // Output
    wire [7:0] DATA;

    // Initial the Unit Under Test
    counter uut(reset, clock, DATA);

    initial begin
        reset = 0;
        clock = 0;
        forever begin
            #10;                // wait 10ns;
            clock = ~clock;
         end
    end
endmodule
```

b) Task II

The program had the following structure

- top.v
  - counter.v
  - display.v
    - decode.v

i) Code:

- top.v

```
module top(reset, button, SCAN_OUT, SEG_OUT);
    input reset, button;
    output SCAN_OUT;
    output [7:0] SEG_OUT;
    wire [3:0] temp_data;

    counter count(reset, button, temp_data);
    display disp(temp_data, SCAN_OUT, SEG_OUT);
endmodule
```

- counter.v

```
module counter(reset, button, DATA);
    input button, reset;
    output reg [3:0] DATA;
    initial DATA = 0;

    always @(posedge button or posedge  reset) begin
        if (reset) begin        // if the reset button is pressed, DATA = 0
            DATA <= 0;
        end
        else begin
            DATA <= DATA + 1;
            if (DATA >= 9) begin
                DATA <= 0;
            end
        end
    end

endmodule
```

- display.v

```
module display(DATA, SCAN_OUT, SEG_OUT);
    input [3:0] DATA;           // number that will be displayed
    output SCAN_OUT;            // used to setup the position to display
    output [7:0] SEG_OUT;       // used to determine which digit to display

    decode DEC(DATA, SEG_OUT);
    assign SCAN_OUT = 1;
endmodule
```

- decode.v

```
module decode(DATA_IN, SEG_OUT);
    input [3:0] DATA_IN;            // used to show which digit to display
    output [7:0] SEG_OUT;           // store the pattern of the digit to be displayed

    wire [7:0] number [15:0];       // number[i] = byte that displays i

    assign number[0] = 8'b10111111;
    assign number[1] = 8'b10000110;
    assign number[2] = 8'b11011011;
    assign number[3] = 8'b11001111;
    assign number[4] = 8'b11100110;
    assign number[5] = 8'b11101101;
    assign number[6] = 8'b11111101;
    assign number[7] = 8'b10100111;
    assign number[8] = 8'b11111111;
    assign number[9] = 8'b11101111;
    assign number[10] = 8'b11110111;
    assign number[11] = 8'b11111100;
    assign number[12] = 8'b11011000;
    assign number[13] = 8'b11011110;
    assign number[14] = 8'b11111001;
    assign number[15] = 8'b11110001;

    assign SEG_OUT = number[DATA_IN];
endmodule
```

ii) Test Bench:

- tb_top.v

```
module tb_top;
    // Inputs
    reg reset, button;
    // Outputs
    wire SCAN_OUT;
    wire [7:0] SEG_OUT;

    integer i, j;

    // Initial the Unit Under Test
    top uut(reset, button, SCAN_OUT, SEG_OUT);

    initial begin
        reset = 0;
        button = 0;
        for(i=0; i<11; i=i+1)
        begin
            // The counter goes to i-1, and then it resets
            for(j=0; j<i; j=j+1)
            begin
                #1 button = 1;      // This simulates the counter button being pressed
                #1 button = 0;
            end
            #1 reset = 1;           // This simulates the reset button being pressed
            #1 reset = 0;
        end
        $finish;
     end
endmodule
```

- tb_counter.v

```
module tb_counter;
    // Inputs
    reg reset, button;
    // Output
    wire [3:0] DATA;

    integer i,j;
    // Initial the Unit Under Test
    counter uut(reset, button, DATA);

    initial begin
        reset = 0;
        button = 0;
        for(i=0; i<11; i=i+1)
        begin
            // The counter goes to i-1, and then it resets
            for(j=0; j<i; j=j+1)
            begin
                #1 button = 1;      // This simulates the counter button being pressed
                #1 button = 0;
            end
            #1 reset = 1;           // This simulates the reset button being pressed
            #1 reset = 0;
        end
        $finish;
     end
endmodule
```

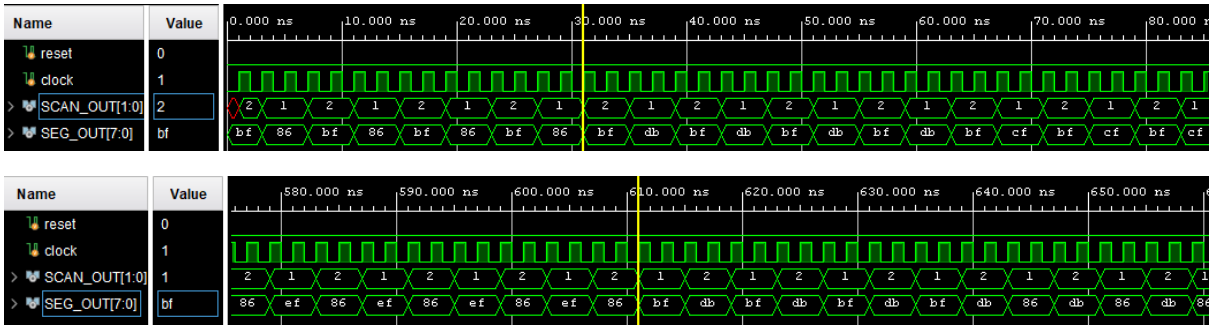
- tb_display.v

```
module tb_display;
    // Input
    reg [3:0] DATA;
    // Outputs
    wire SCAN_OUT;
    wire [7:0] SEG_OUT;

    integer i;
    // Initial the Unit Under Test
    display uut(DATA, SCAN_OUT, SEG_OUT);

    initial begin
        for(i=0; i<10; i=i+1)    // DATA goes from 0 to 9
        begin
            DATA = i;
            # 10;               // wait 10ns
        end
        $finish;
     end
endmodule
```

4. Simulation Waveforms:

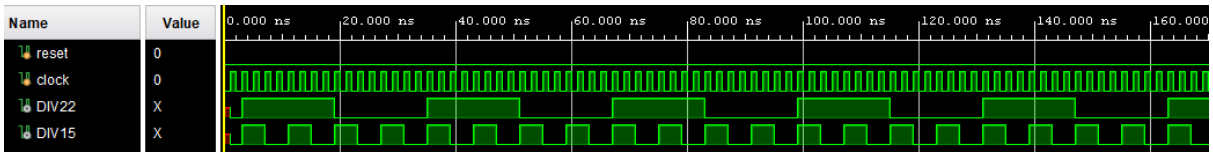
a) Task I

- tb_top.v



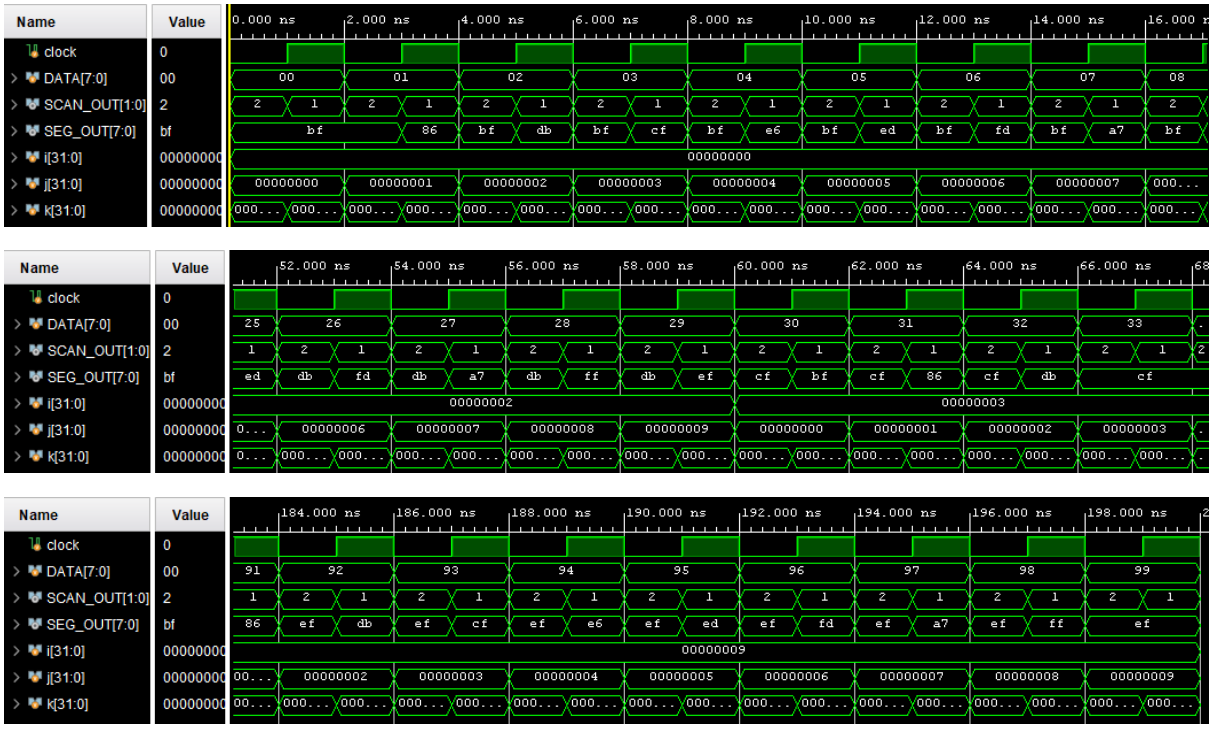
Obs.: To be able to see the results the values of DIV15 and DIV22 were changed to DIV[1] and DIV[3], respectively (see frequency_divider_by_2.v for reference).

- tb_divider.v

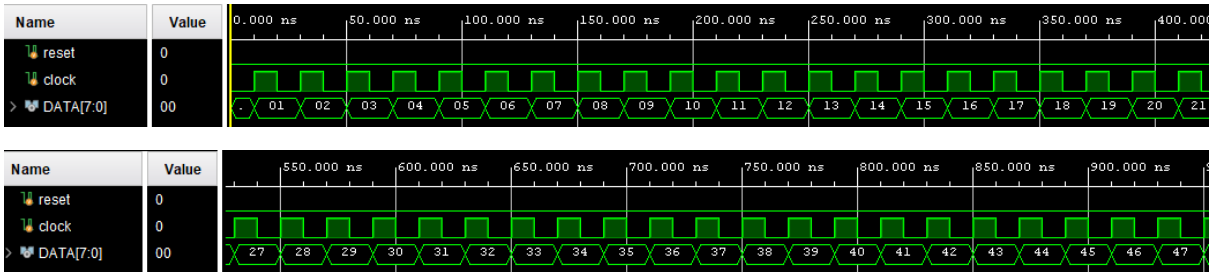


Obs.: To be able to see the results the values of DIV15 and DIV22 were changed to DIV[1] and DIV[3], respectively (see frequency_divider_by_2.v for reference).

- tb_display.v

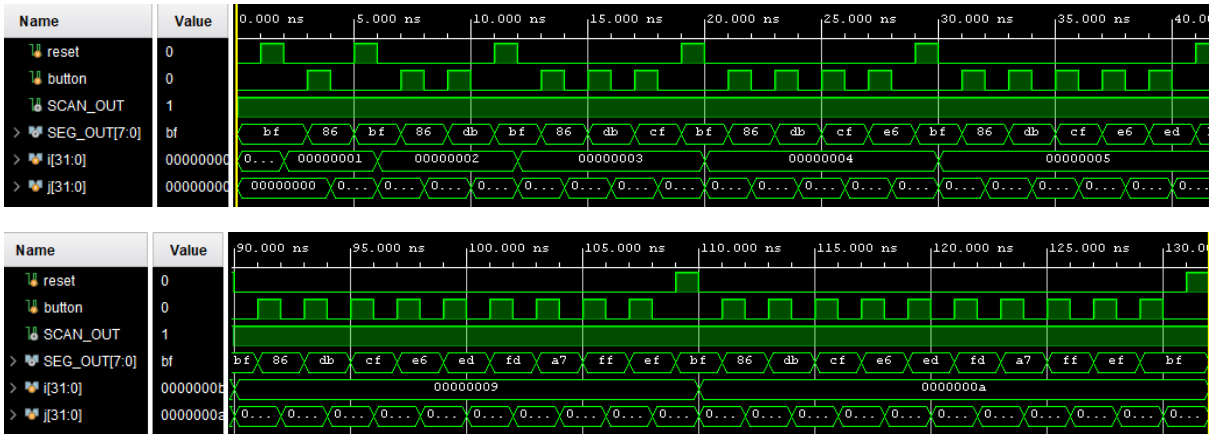


- tb_counter.v

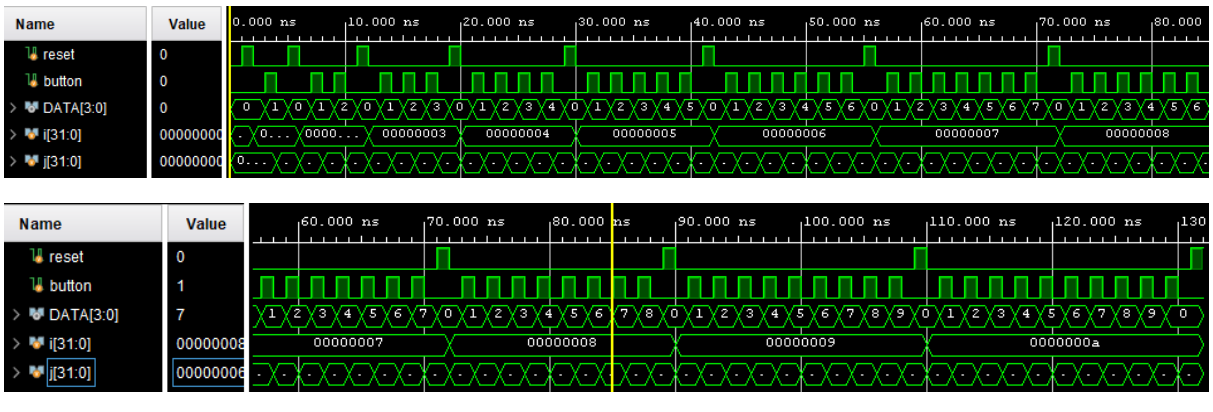


b) Task II

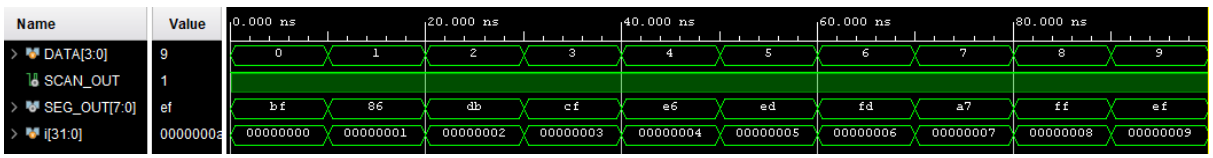
- tb_top.v



- tb_counter.v



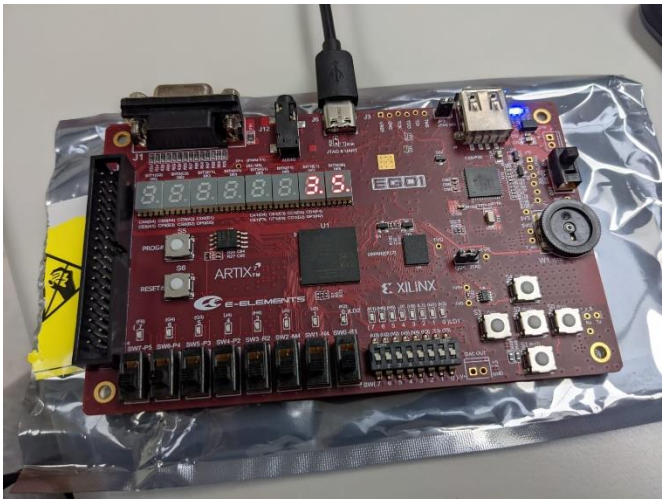
- tb_display.v

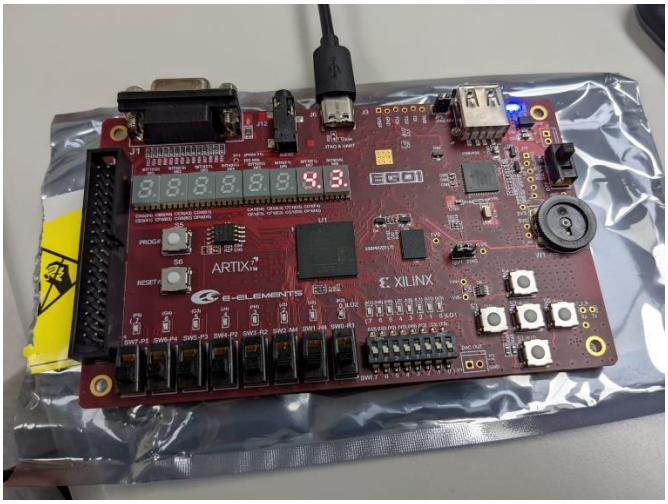


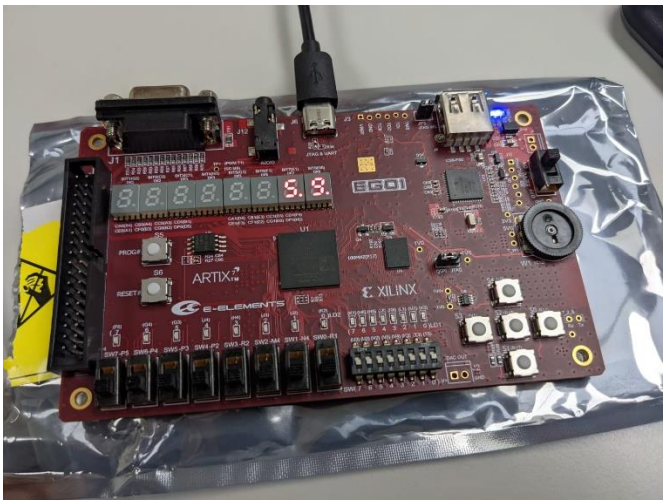
5. FPGA Results:

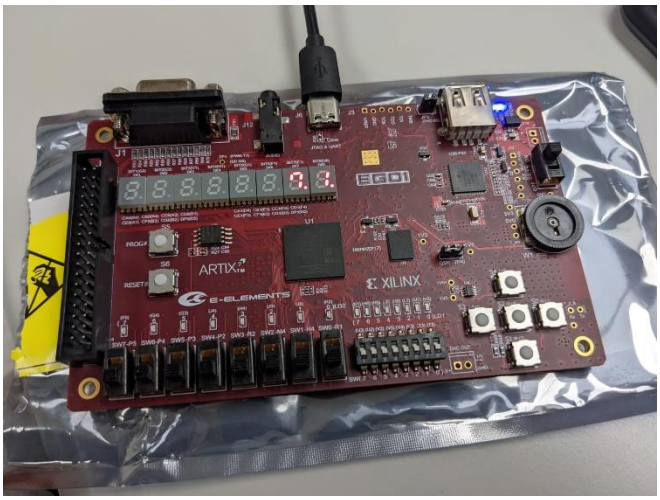
a) Task I

Pictures showing the board at different values.



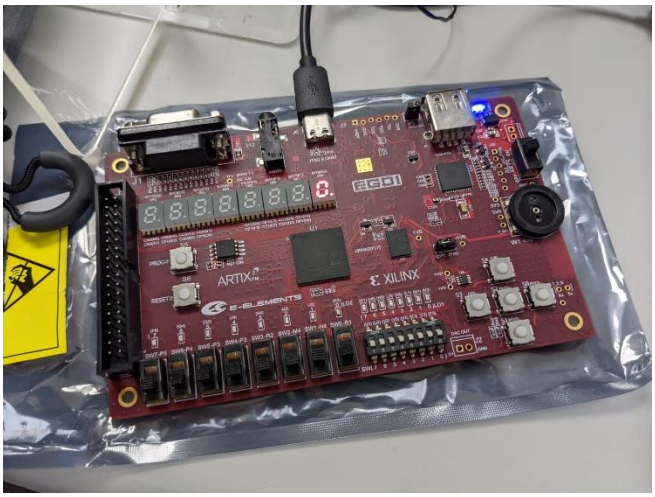


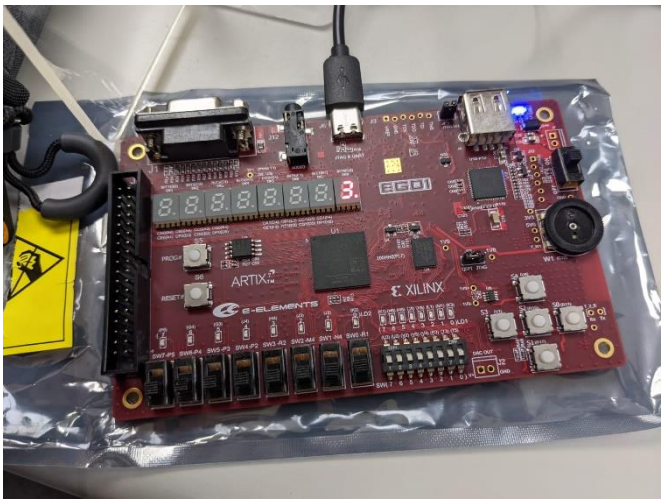


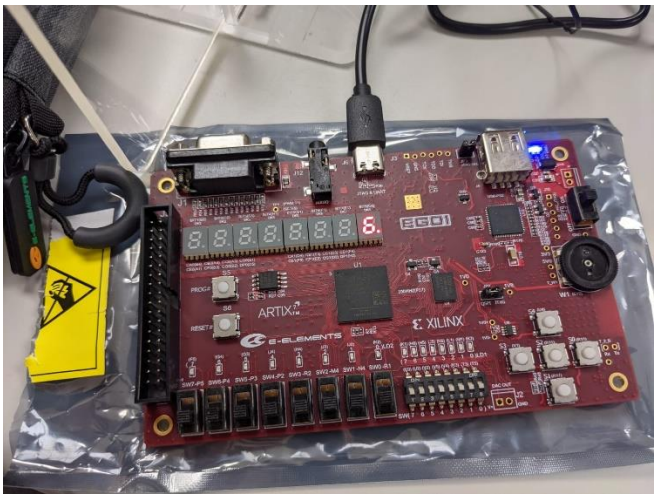


b) Task II

Pictures showing the board at different values.







6. Reflection:

This lab presented several challenges that needed to be handled one by one. The first task needed two clocks with different frequency, the first needed to update the counter and the second one to determine the speed at which the digits of the four-digit seven segment display are turned on. On the second task, once the code was done and the constraints were defined an extra line of code was needed to be able to use the “always” command with only buttons, this extra line of code was “set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets button_IBUF]”. Another challenge was to do test bench on modules with clocks. It can be seen from those test benches that the code is different compared to the ones that do not have a clock as a parameter.

Other than that, mentioned problems, the rest of the lab went smoothly. And from the results we can see that both experiments went as expected.