1) A copy of the Verilog code used to implement the design(s). The Verilog code (including test bench code) should be complete with useful comments (5 pts).

Sseg_x4_top file:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/20/2025 10:17:09 AM
// Design Name:
// Module Name: sseg_x4 _top
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module sseg_x4_top(
    input [15:0] sw,
    input btnC,
    output [6:0] seg,
    output [3:0] an,
    output dp,
    output [4:0] JA,
    input clk
    );

    wire clkd;
    wire [3:0] not_used;
    wire [3:0] hex_num;

    //instantiate clk_gen
    clk_gen U1 (
        .clk(clk),
        .rst(btnC),
```

```
        .clkd(clkd)
    );

    //instantiate digit_selector
    digit_selector U2 (
        .clk(clkd),
        .rst(btnC),
        .digit_sel(an)
    );

    //instantiate hex_num_gen
    hex_num_gen U3(
        .digit_sel(an),
        .sw(sw),
        .hex_num(hex_num)
    );

    //instantiate sseg
    sseg U4 (
        .seg(seg),
        .an(not_used),
        .dp(dp),
        .sw(hex_num)
    );

    assign JA[0] = clkd;    // slowed clk signal
    assign JA[1] = an[0];   // Anode for digit 1
    assign JA[2] = an[1];   // Anode for digit 2
    assign JA[3] = an[2];   // Anode for digit 3
    assign JA[4] = an[3];   // Anode for digit 4
endmodule
```

Clk_gen file:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/20/2025 10:23:06 AM
// Design Name:
// Module Name: clk_gen
// Project Name:
```

```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module clk_gen(
    input clk,
    input rst,
    output clkd
    );

    //declare a 26 bit register to that we can count through
    reg [25:0] reg_count;

    //counter
    always @ (posedge clk)
        begin
            if (rst)
                reg_count <= 26'b0; //if rst is true then reset the counter to 0s
            else
                reg_count <= reg_count + 1; //if rst is false keep counting up
        end

    assign clkd = reg_count[16]; //set the new clk signal clkd to bit 16 of reg_count

endmodule



Digit_selector:
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2025 10:14:08 AM
```

```
// Design Name:
// Module Name: digit_selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module digit_selector(
    input clk,
    input rst,
    output reg [3:0] digit_sel
    );

    //count is a register we will count through
    reg [1:0] count;


    //counter to count through count
    always @ (posedge clk or posedge rst)
        begin
            if (rst)
                count <= 2'b0; //if rst is true then restart count
            else
                count <= count + 1; //if rst is false keep counting
        end

    //check the state on count and depending on its state set digit_sel to one of the 4 displays
    always @(*)
        begin
            case (count)
                2'b00: digit_sel = 4'b1110;
                2'b01: digit_sel = 4'b1101;
                2'b10: digit_sel = 4'b1011;
                2'b11: digit_sel = 4'b0111;
                default: digit_sel = 4'b1111;
```

```
            endcase
        end

endmodule
```

Hex_num_gen file:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/27/2025 11:00:06 AM
// Design Name:
// Module Name: hex_num_gen
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module hex_num_gen(
    input [3:0] digit_sel,
    input [15:0] sw,
    output reg [3:0] hex_num
    );

    //break up the 16 switches into four sets each one determiens the value of an individual
display segment
    wire [3:0] digit4 = sw[15:12];
    wire [3:0] digit3 = sw[11:8];
    wire [3:0] digit2 = sw[7:4];
    wire [3:0] digit1 = sw[3:0];

    //check the state of digit_sel
```

```
    //depending on its state set hex_num to the correct set of switches for that display
    //so we can output the correct value
    always @ (*)
        case (digit_sel)
            4'b1110: hex_num = digit1;
            4'b1101: hex_num = digit2;
            4'b1011: hex_num = digit3;
            4'b0111: hex_num = digit4;
            default: hex_num = 4'b0000;
        endcase

endmodule
```

Sseg file:

```
//timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/06/2025 10:15:19 AM
// Design Name:
// Module Name: seg7
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module sseg(
    input [3:0] sw,
    output reg [6:0] seg,
    output [3:0] an,
    output dp
    );
```

```
    assign an = 4'b1110;

    //turn dp off
    assign dp = 1'b1;

    //check the state of the switches and depending on the value that sw has
    //determine what the correct number is and set that to seg
        always @(sw)
            case (sw)
                4'h0: seg = 7'b1000000;
                4'h1: seg = 7'b1111001;
                4'h2: seg = 7'b0100100;
                4'h3: seg = 7'b0110000;
                4'h4: seg = 7'b0011001;
                4'h5: seg = 7'b0010010;
                4'h6: seg = 7'b0000010;
                4'h7: seg = 7'b1111000;
                4'h8: seg = 7'b0000000;
                4'h9: seg = 7'b0010000;
                4'hA: seg = 7'b0001000;
                4'hB: seg = 7'b0000011;
                4'hC: seg = 7'b1000110;
                4'hD: seg = 7'b0100001;
                4'hE: seg = 7'b0000110;
                4'hF: seg = 7'b0001110;
                default: seg = 7'b1111111;
            endcase
endmodule
```

Sseg_x4_tb file:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/20/2025 11:52:57 AM
// Design Name:
// Module Name: sseg_x4_tb
// Project Name:
// Target Devices:
// Tool Versions:
```

```
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module sseg_x4_tb(
    );

    reg clk;
    reg btnC;
    reg [15:0] sw;

    wire [6:0] seg;
    wire [3:0] an;
    wire dp;
    wire [4:0] JA;


    sseg_x4_top uut(
        .clk(clk),
        .btnC(btnC),
        .sw(sw),
        .seg(seg),
        .an(an),
        .dp(dp),
        .JA(JA)
    );

    initial
        clk = 0;
    always #5
        clk = ~clk;

    initial
    begin
        btnC = 1'b1;
        sw = 16'h4321;
```

```
        #20;

        btnC = 1'b0;

        #5000000;

        $finish;

    end
endmodule
```

2) Include a brief description of the unique (new) tools, technologies, or methods used to implement this lab (5 pts).

One of the main new methods we used was to slow down the clock with a counter. We took our clock signal as an input and then used that clock to make a counter. That then depending on the value we gave the output determined what the new signal was. We also used a ton of external files that we then had to instantiate into our main file which we haven't done to this extent before. We also got to use the JA ports to visualize via oscilloscope the waves from the Basys3 board

3) Include Vivado screen shots of simulations, oscilloscope waveforms, etc., to document your work (5 pts).

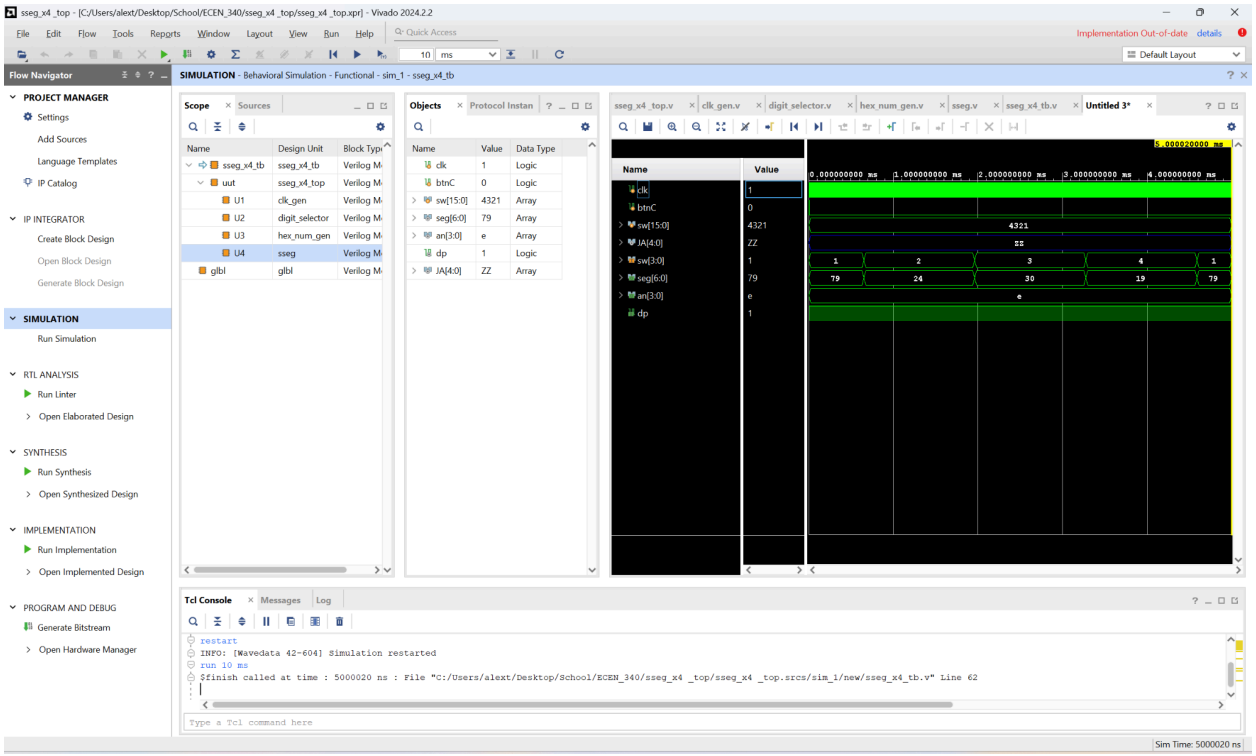

Figure 1: Working simulation of completed lab

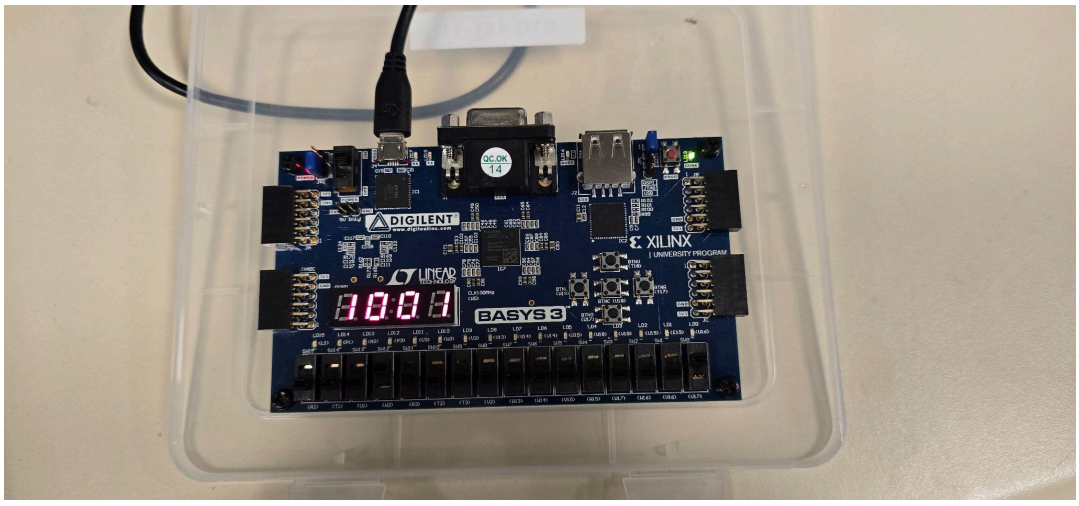

Figure 2: Basys3 running the program

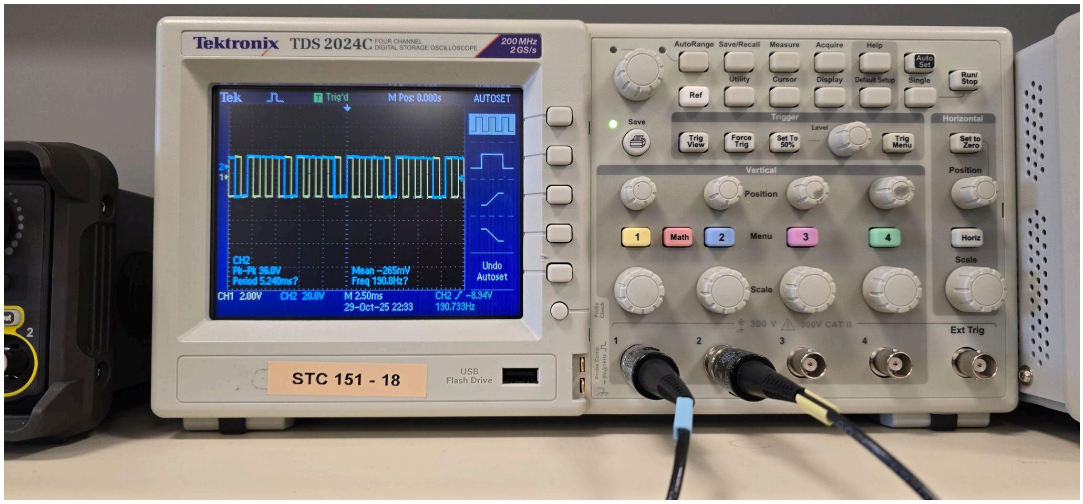

Figure 3: Working simulation of clkd and An0

3) The report should be professional quality—meaning it will be neat and use proper English (5 pts).

4) In your conclusion statement, discuss your results, the method of testing, and include the level of functionality of the lab (5 pts).

I was able to achieve full functionality. All of the individual displays updated whose value would change depending on the values of the switches. In order to test I flipped the switches to various numbers and ensured the values on the display matched up. I also used the JA[4:0] pins to check critical waveforms on the oscilloscope.