Lab 7 – Improved Barrel Counter

Group G
Chang, Philbert (013179257)
Hua, Russell (013184015)
Thai, Paul (014760252)
Professor Aly

ECE 3300L.01
August 09, 2021

Problem

For this lab, the task is to create a modified and improved up down counter that could take in a load from four switches and display it onto the seven-segment display according to the selects. The way the FPGA is mapped out, there is a switch for the counter to go up or down. There is a switch that would enable to be the count to start and stop. There are three switches for the select, four for the input that is being loaded in, and then five that control the speed of the counter.

Code Detail

The code uses a lot of modules to achieve this task. There is the top module that brings everything in together. The SSD module is meant for displaying the values of the counter on the seven-segment display. The clk_divider module is used to change the speed of the clock. Inside of the BCD_32_bit module there are sub modules that controls the count, the selects, and the input that would allow us to complete this lab in the manner that the professor explained in his video.

```
23  module top(
24      input clk,
25      input reset,
26      input [4:0] SW,
27      input enable,
28      input up_down,
29      input load,
30      input [3:0] in,
31      input [2:0] select,
32      output wire [6:0] a_to_g,
33      output wire [7:0] an,
34      output wire dp
35  );
36
37  wire temp1;
38  clk_divider  GEN(
39      .clk(clk),
40      .reset(reset),
41      .SW(SW),
42      .clk_div(temp1)
43  );
44
45  wire [31:0] temp2;
46
47  BCD_32_bit  BCD(
48      .clk(temp1),
49      .reset(reset),
50      .en(enable),
51      .up_down(up_down),
52      .load(load),
53      .in(in),
54      .select(select),
55      .set(temp2)
56  );
57
58  ssd display(
59      .SW(temp2),
60      .clk(clk),
61      .reset(reset),
62      .a_to_g(a_to_g),
63      .an(an),
64      .dp(dp)
```

```
58  ssd display(
59      .SW(temp2),
60      .clk(clk),
61      .reset(reset),
62      .a_to_g(a_to_g),
63      .an(an),
64      .dp(dp)
65  );
66
67  endmodule
```

Figures 1: Verilog code - top

```
21  `define dp_off   1'b1
22  `define initial_digit 8'b11111111
23
24  module ssd(
25          input [31:0] SW,
26          input clk,
27          input reset,
28          output reg [6:0] a_to_g,
29          output reg [7:0] an,
30          output wire dp
31           );
32
33  assign dp  = `dp_off;
34
35  wire [2:0] s;
36  wire [7:0] aen;
37  reg [19:0] clkdiv;
38  reg [3:0] digit;
39  assign s = clkdiv[19:17];
40  assign aen = `initial_digit;
41
42
43
44  // clock divider
45      always@(posedge clk or posedge reset)
46          begin
47              if (reset)
48                  clkdiv <= 0;
49              else
50                  clkdiv <= clkdiv+1;
51          end
52
53  // digit select :ancode
54              always@(aen, s)
55                  begin
56                      an = 8'b11111111;
57                      if(aen[s] == 1)
58                          an[s] = 0;
59                  end
60
61              // 4-to-1 Mux: mux4x1
62              always@(s, SW)
```

```
60
61              // 4-to-1 Mux: mux4x1
62              always@(s, SW)
63                 begin
64                  case(s)
65                      0: digit = SW[3 : 0];
66                      1: digit = SW[7 : 4];
67                      2: digit = SW[11: 8];
68                      3: digit = SW[15:12];
69                      4: digit = SW[19:16];
70                      5: digit = SW[23:20];
71                      6: digit = SW[27:24];
72                      7: digit = SW[31:28];
73                default: digit = 4'bZZZZ;
74                  endcase
75                end
76
77  // 7-segement decoder : hex7seg
78
79  always@(digit)
80      begin
81          case(digit)
82                  0: a_to_g = 7'b0000001;
83                  1: a_to_g = 7'b1001111;
84                  2: a_to_g = 7'b0010010;
85                  3: a_to_g = 7'b0000110;
86                  4: a_to_g = 7'b1001100;
87                  5: a_to_g = 7'b0100100;
88                  6: a_to_g = 7'b0100000;
89                  7: a_to_g = 7'b0001111;
90                  8: a_to_g = 7'b0000000;
91                  9: a_to_g = 7'b0000100;
92
93            default: a_to_g = 7'b0000100;
94          endcase
95      end
96
97
98  endmodule
```

Figures 2: Seven segment display

```
22  module clk_divider(
23      input clk,
24      input reset,
25      input [4:0] SW,
26      output reg clk_div
27  );
28
29  reg [31:0] count;
30
31  always@(posedge clk or posedge reset)
32  begin
33      if (reset)
34          count <= 0;
35      else
36          count <= count + 1;
37  end
38
39  integer i;
40  always@(posedge clk or posedge reset)
41  begin
42      if (reset)
43          clk_div <= 1'b0;
44      else
45      begin
46          for (i = 0; i < 32; i = i + 1)
47              if (SW == i)
48                  clk_div <= count[i];
49      end
50  end
51
52  endmodule
```

Figure 3: Clock divider

```
module BCD_32_bit(
    input clk,
    input reset,
    input en,
    input up_down,
    input load,
    input [3:0] in,
    input [2:0] select,
    output wire [31:0] set
);

wire [7:0] temp;
wire [31:0] temp_select;

assign temp[0] = en;

genvar i;

demux_DB data (
    .in(in),
    .select(select),
    .out(temp_select)
);

generate
for (i = 0; i<8 ; i = i+1)
begin
    counter count(
        .clk(clk),
        .reset(reset),
        .enable_in(temp[i]),
        .up_down(up_down),
        .load(load),
        .in0(temp_select[i]),
        .in1(temp_select[i + 8]),
        .in2(temp_select[i + 16]),
        .in3(temp_select[i + 24]),
        .set(set [4*(i+1)-1 : 4*i]),
        .enable_out(temp[i+1])
    );
end
endgenerate

endmodule
```

Figure 4: BCD 32 bit

```
module counter(
    input clk,
    input reset,
    input enable_in,
    input up_down,
    input load,
    input in0,
    input in1,
    input in2,
    input in3,
    output reg [3:0] set,
    output wire enable_out
);

wire [3:0] in;
assign in[0] = in0;
assign in[1] = in1;
assign in[2] = in2;
assign in[3] = in3;

always@(posedge clk or posedge reset or posedge load)begin
    if (reset)begin
        if (up_down)
            set <= 4'd9;
        else
            set <= 4'd0;
      end
    else if (load)
        set <= in;
    else begin
        if (enable_in)begin
            if (up_down== 1'b1)begin
                if (set > 0)
                    set <= set - 1;
                else
                    set <= 4'd9;
            end
            else begin
                if(set <9)
                    set <= set+ 1;
                else
```

```
            else begin
                if(set <9)
                    set <= set+ 1;
                else
                    set <= 4'd0;
            end
        end
    end
end
// checks if the current counter value will result in carry to the next counter
assign enable_out = ((set[3] & set[0] & (~set[2]) & (~set[1]) & (~up_down)) | ( (~set[3]) & (~set[0]) & (~set[2]) & (~set[1]) & (up_down))) & enable_in ;

endmodule
```

Figures 5: Counter (1 BCD)

```
22  //databus for the multiplexer
23  module demux_DB(
24      input [3:0] in,
25      input [2:0] select,
26      output wire [31:0] out
27  );
28
29  generate
30  for (genvar i = 0; i < 4; i = i + 1)
31    begin: DEMUXS
32      demux_3_to_8 setter(
33          .in(in[i]),
34          .select(select),
35          .out(out[8*i + 7 : 8 * i])
36      );
37    end
38  endgenerate
39
40  endmodule
```

Figure 6: Demux_DB

```
23  module demux_3_to_8(
24      input in,
25      input [2:0] select,
26      output wire [7:0] out
27  );
28  assign out[0] = in & ~select[0] & ~select[1] & ~select[2];
29  assign out[1] = in & select[0] & ~select[1] & ~select[2];
30  assign out[2] = in & ~select[0] & select[1] & ~select[2];
31  assign out[3] = in & select[0] & select[1] & ~select[2];
32  assign out[4] = in & ~select[0] & ~select[1] & select[2];
33  assign out[5] = in & select[0] & ~select[1] & select[2];
34  assign out[6] = in & ~select[0] & select[1] & select[2];
35  assign out[7] = in & select[0] & select[1] & select[2];
36
37  endmodule
```

Figure 7: Demux 3x8

```
23  module top_tb(
24
25      );
26      reg clk_tb;
27      reg reset_tb;
28      reg [4:0] SW_tb;
29      reg enable_tb;
30      reg up_down_tb;
31      reg load_tb;
32      reg [3:0] in_tb;
33      reg [2:0] select_tb;
34      wire [6:0] a_to_g_tb;
35      wire [7:0] an_tb;
36      wire dp_tb;
37      integer i;
38      top test_top(
39          .clk     (clk_tb     ),
40          .reset   (reset_tb   ),
41          .SW      (SW_tb      ),
42          .enable  (enable_tb  ),
43          .up_down (up_down_tb ),
44          .load    (load_tb    ),
45          .in      (in_tb      ),
46          .select  (select_tb  ),
47          .a_to_g  (a_to_g_tb  ),
48          .an      (an_tb      ),
49          .dp      (dp_tb      )
50      );
51
52      initial begin
53          clk_tb = 0;
54          reset_tb = 1;
55          enable_tb = 0;
```

```
52      initial begin
53          clk_tb = 0;
54          reset_tb = 1;
55          enable_tb = 0;
56          up_down_tb = 0;
57          load_tb = 0;
58          SW_tb = 0;
59          in_tb = 0;
60          select_tb = 0;
61      end
62      initial begin
63          forever begin
64              #10 clk_tb = ~clk_tb;
65          end
66      end
67      initial begin
68          #10 reset_tb = 0;
69          for (i = 0; i < 32768; i = i + 1) begin
70              #10 SW_tb = i[4:0];
71              enable_tb = i[5];
72              up_down_tb = i[6];
73              load_tb = i[7];
74              in_tb = i[11:8];
75              select_tb = i[14:12];
76          end
77          #100 $finish;
78      end
79  endmodule
```

Figures 8: Testbench

Flowchart

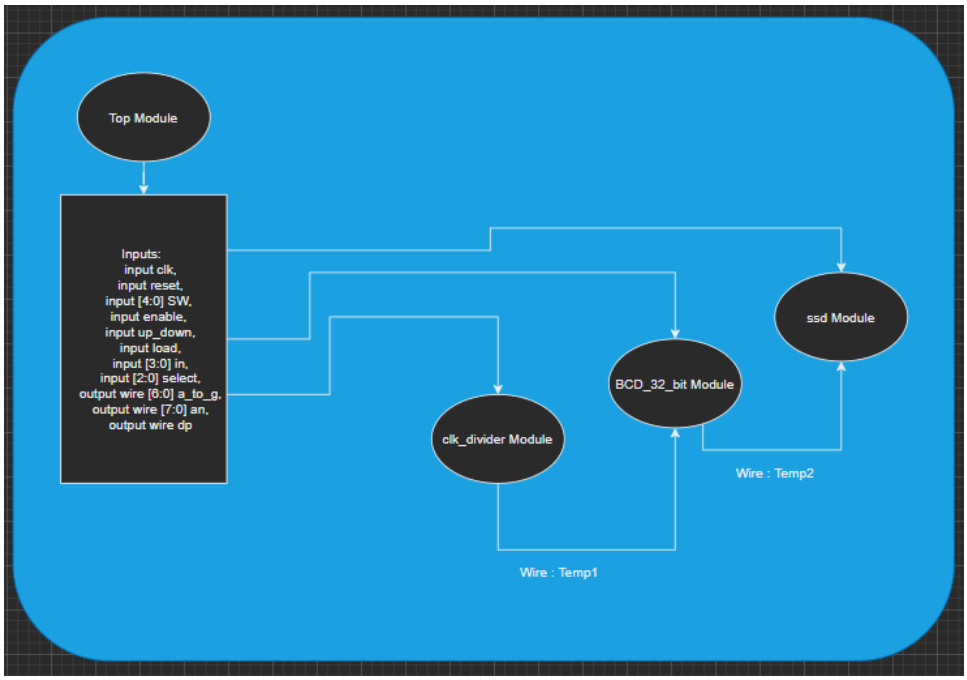

Figure 9: Top

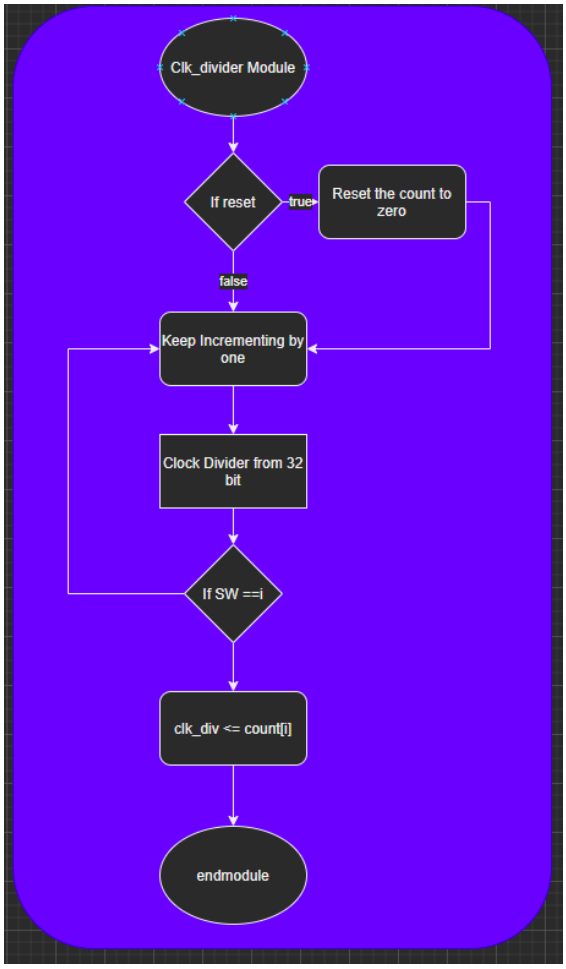

Figure 10: Clock divider

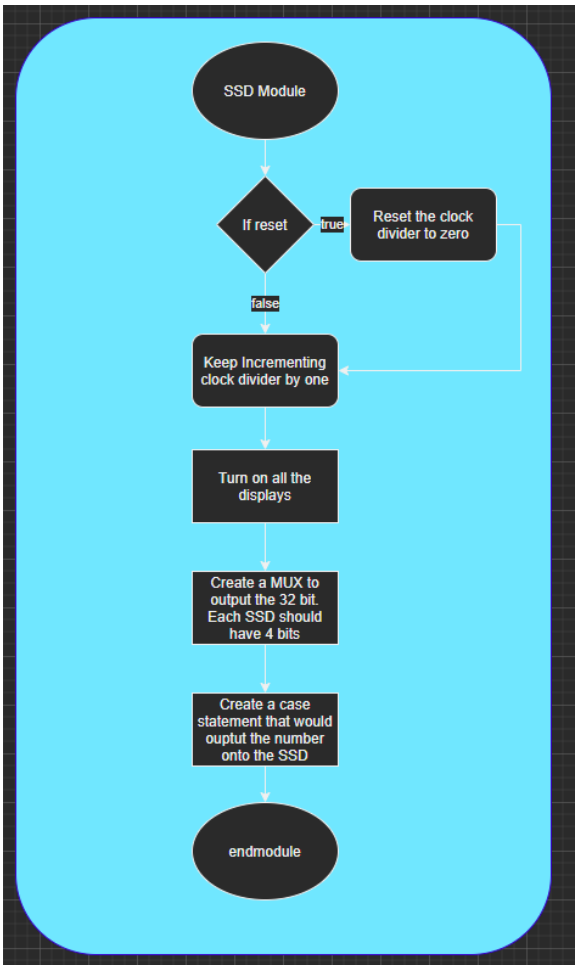

Figure 11: Display

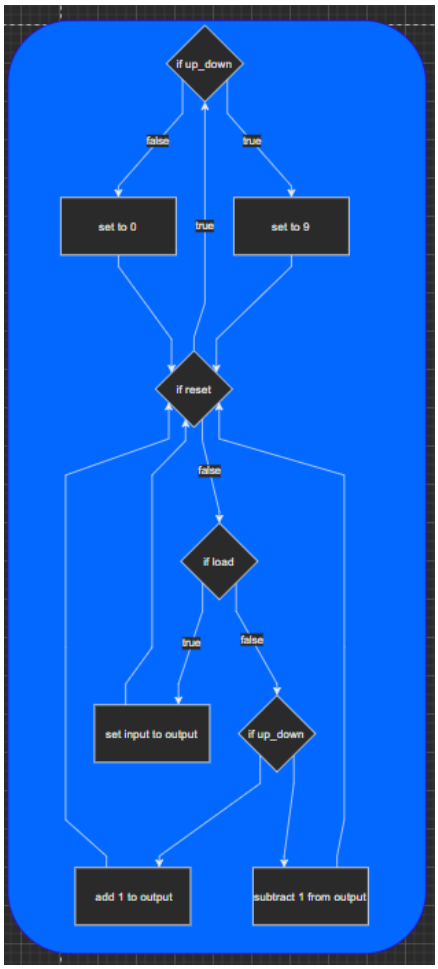

Figure 12: Counter

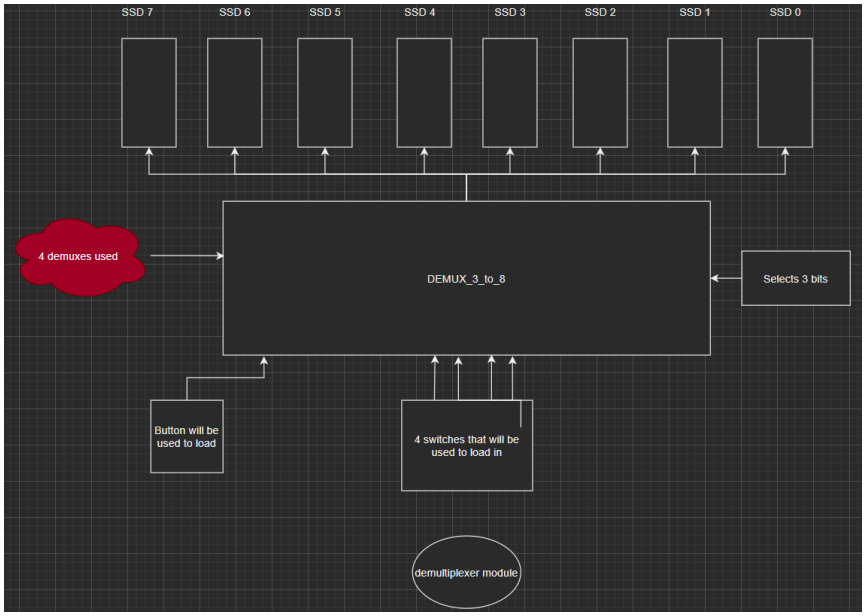

Figure 13: Demux

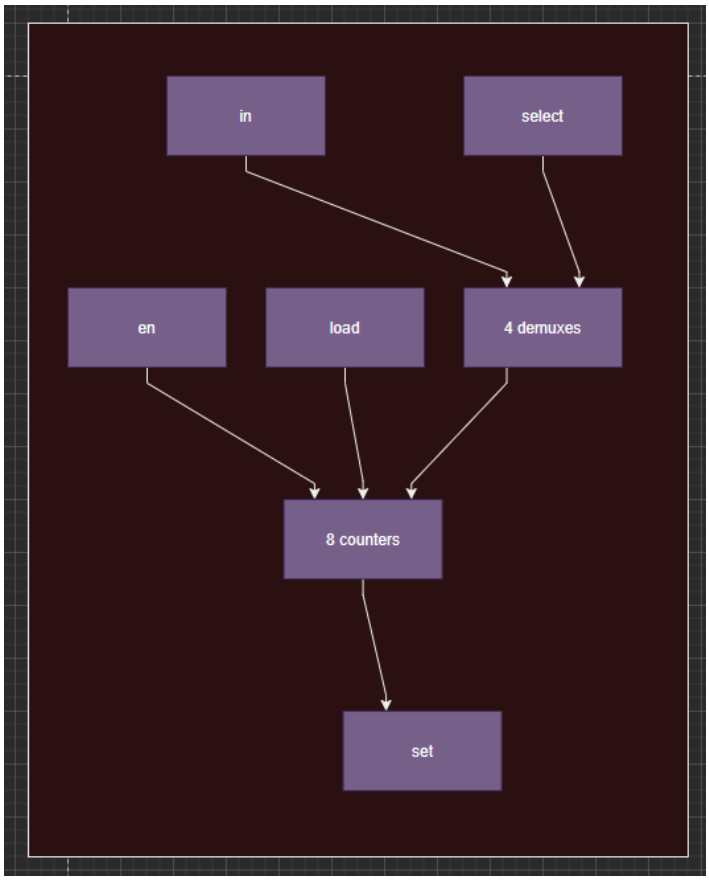

Figure 14: BCD 32 bit

Results

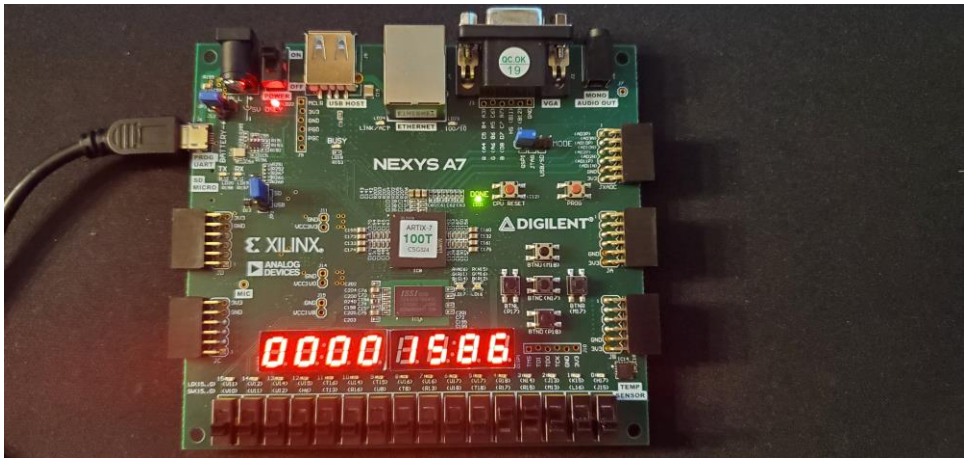

Figure 15: Initial for Counting up

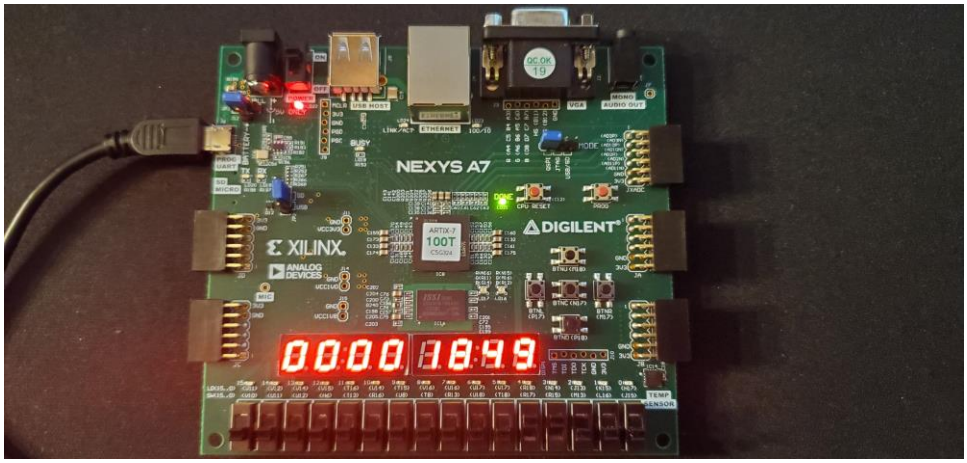

Figure 16: Result for Counting up

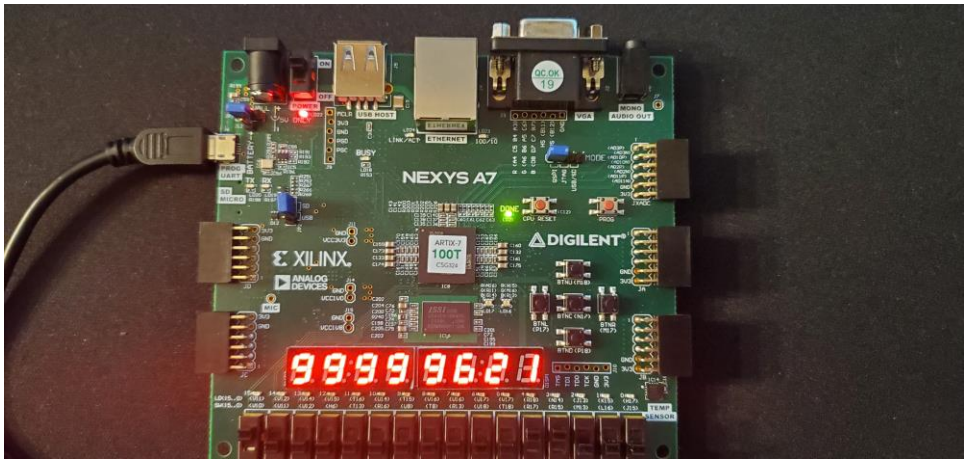

Figure 17: Initial for Counting down

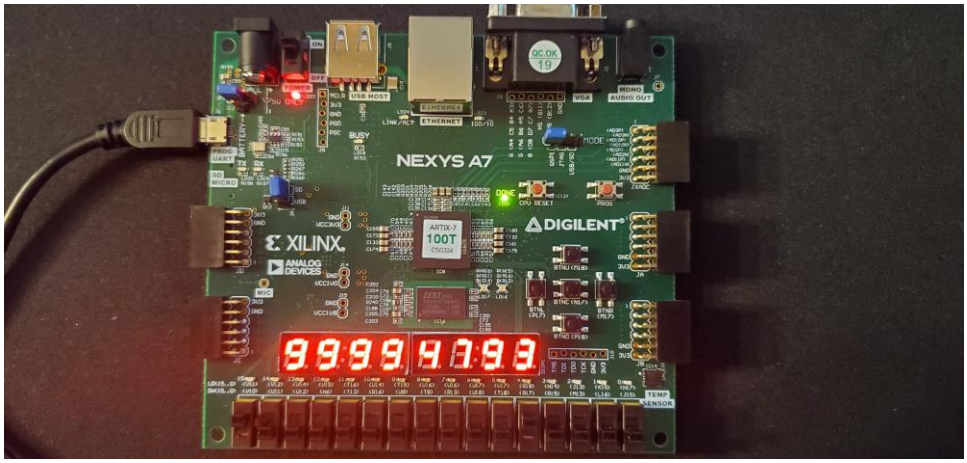

Figure 18: Final Result for Counting Down

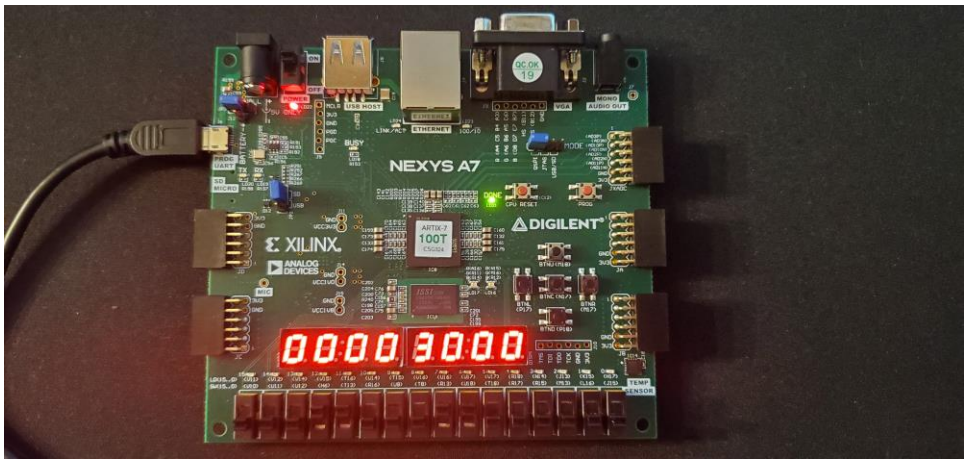

Figure 19: Loading 3 at the 4th Seven Segment Display

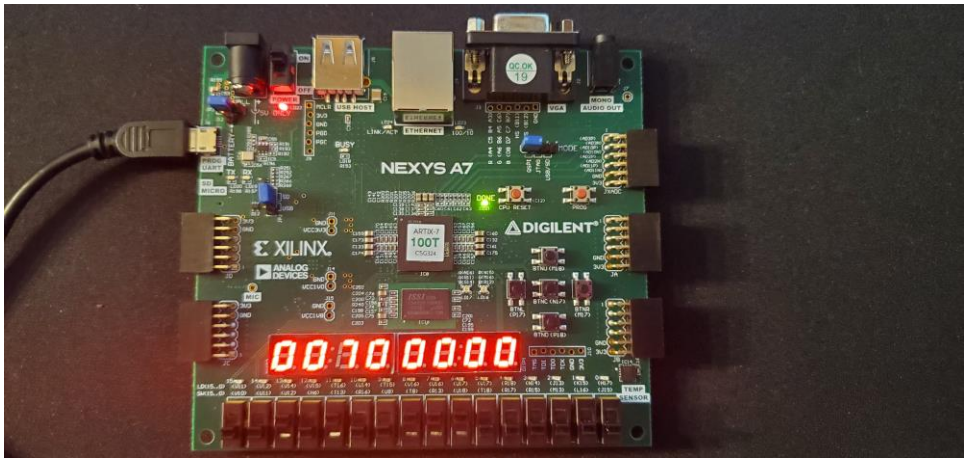

Figure 20: Loading 7 at the 6th Seven Segment Display

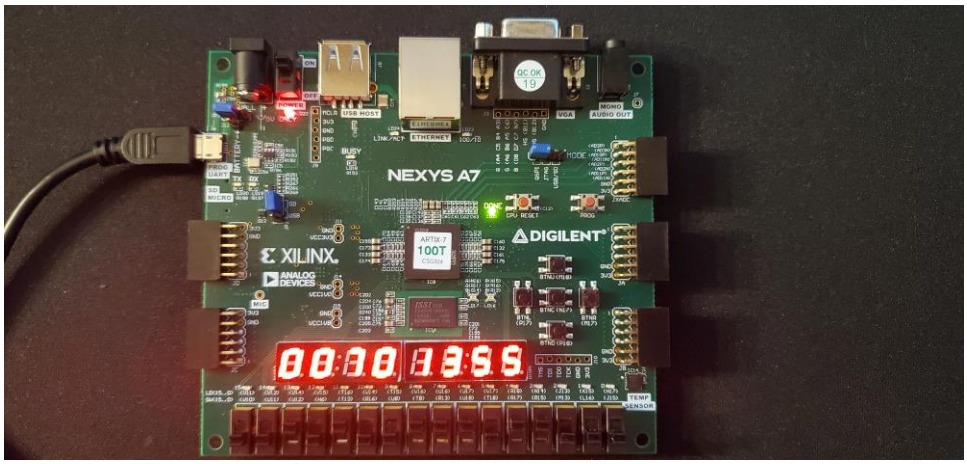

Figure 21: Counting up from the load in

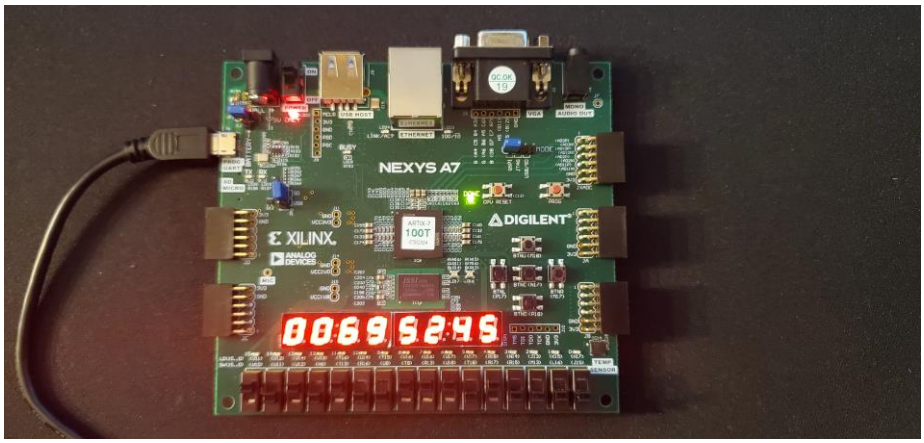

Figure 22: Counting down from the load in

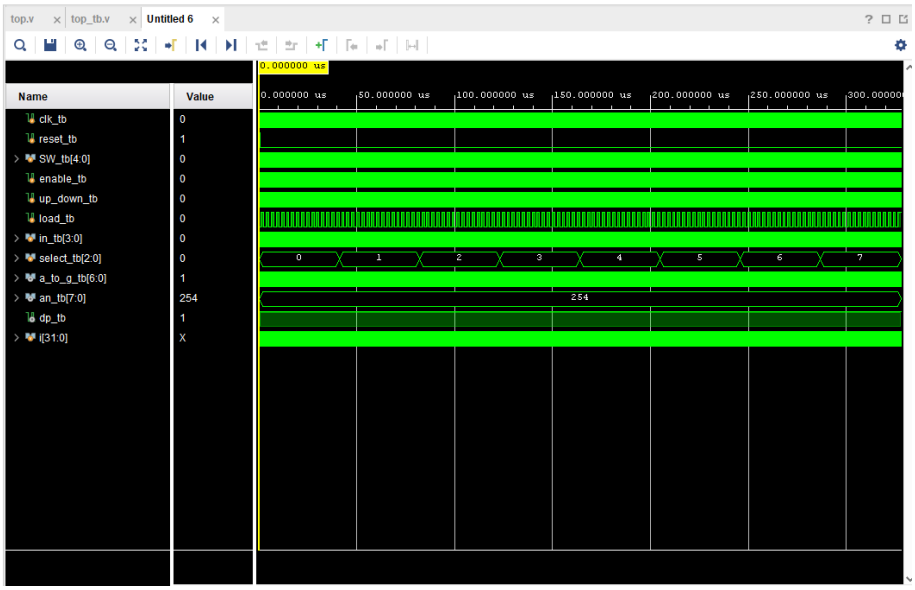

Figure 23: simulation zoomed to fit

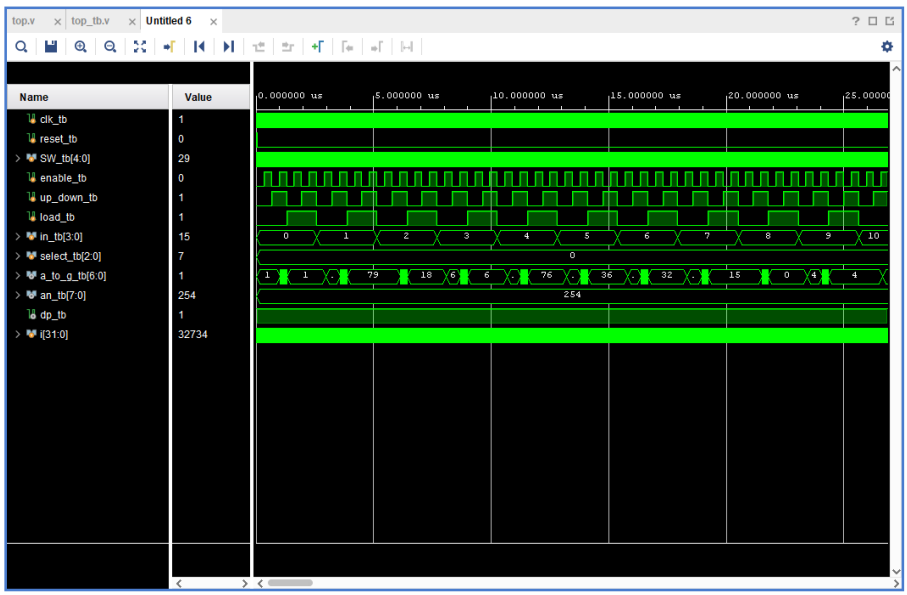

Figure 23: Simulation zoomed in

Hierarchy

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	Slice (15850)	LUT as Logic (63400)	Bonded IOB (210)	BUFGCTRL (32)
top	174	149	8	114	174	33	2
BCD (BCD_32_bit)	159	96	4	96	159	0	0
display (ssd)	5	20	0	9	5	0	0
GEN (clk_divider)	10	33	4	12	10	0	0

Figure 24: Utilization

Discussion

The counter works without problems. There are no glitches in the display or inputs. Possible optimizations to the code could be made to improve performance, power dissipation, or complexity. Another possible feature could add a register that would hold the value of the loaded input, so we don't have to reset the other values of the counter to zero when we load in a value. That way, we could just load in a value mid count, and have it kept on doing what it is doing without resetting.

Conclusion

We were able to accomplish all the objectives that were given in this lab. Given more time, adding a register to hold the value of the displays can be added relatively quickly.

Work Distribution

Russell: everything / Philbert: everything / Paul: everything