

LAB 5

Group C

Edward Enriquez, eenriquez@cpp.edu, ID 013242619
Lorenzo Antonio Fabian Lopez, lfabianlopez@cpp.edu, ID 014903070
Rafael Gaeta, rgaeta@cpp.edu, ID 014522794

Contribution:

Edward Enriquez
50% Coded working schematic implementation and testbench

Lorenzo Antonio Fabian Lopez
25% Analysis of results and debugging

Rafael Gaeta
25% Code Analysis

Abstract

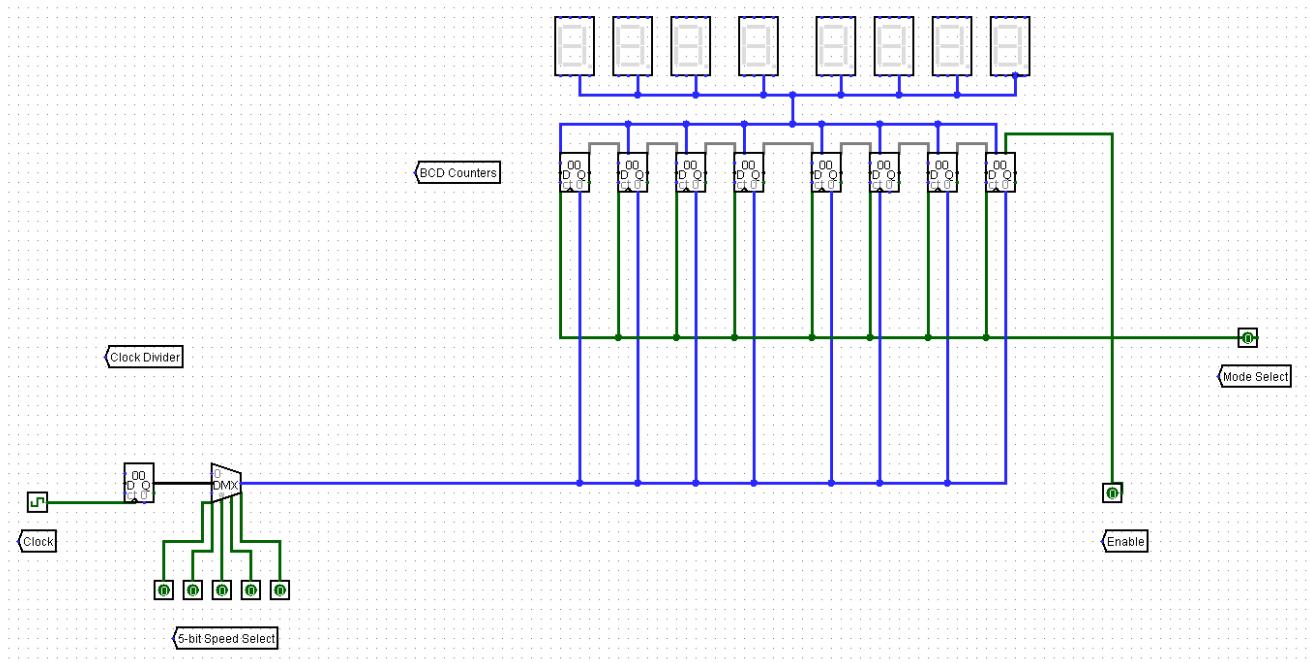
The objective of this lab is to construct a BCD up-counter & down-counter that counts from 0-99999999 & 99999999-0 respectively. The velocity of the counter will depend on the clock divider and will be controlled by the first five switches on the board. The numbers will be displayed on the seven segment display. Additionally a test bench for each component was created to test the functionality of each component.

Specifications,

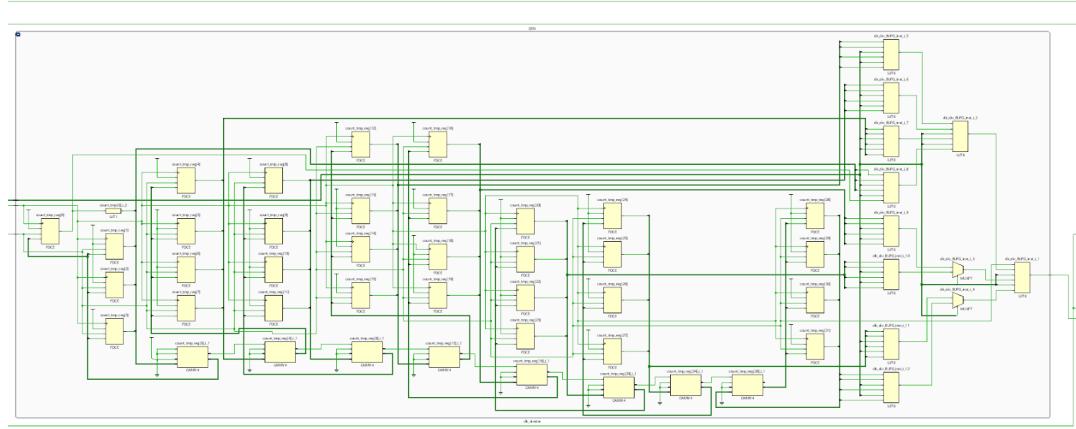
- Define Circuit Schematic/Design
- Total power ≤ 0.114 W
- Number of LUT's ≤ 84
- Number of FF ≤ 101

Circuit Description

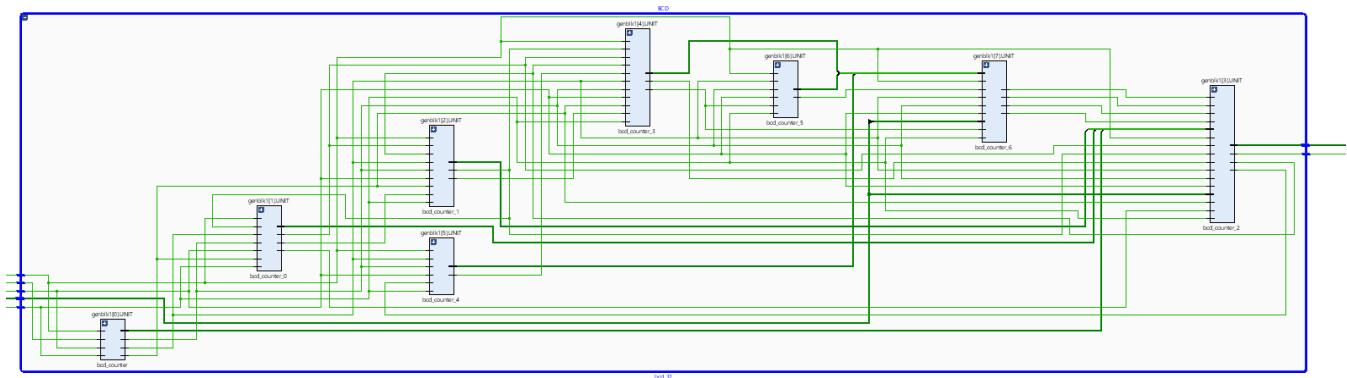
Diagram



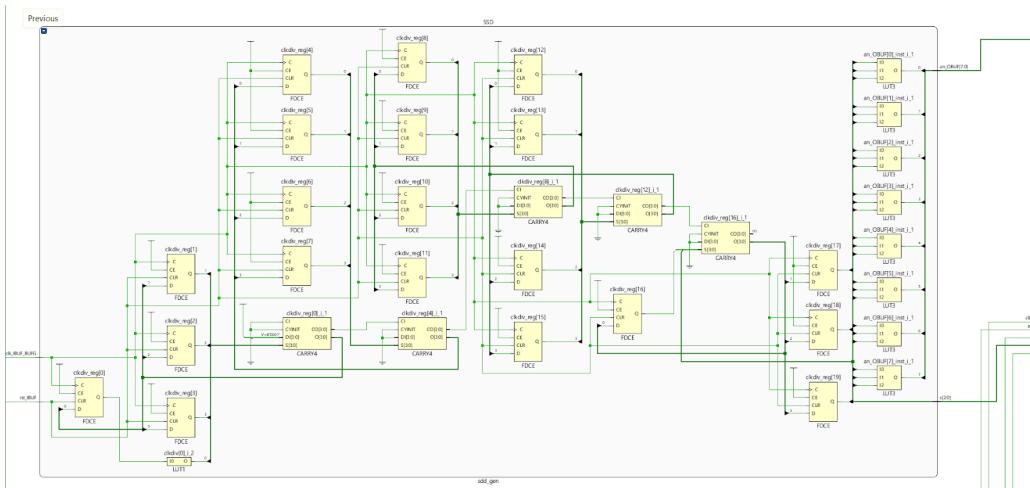
Verilog Generated Schematics with Instantiation



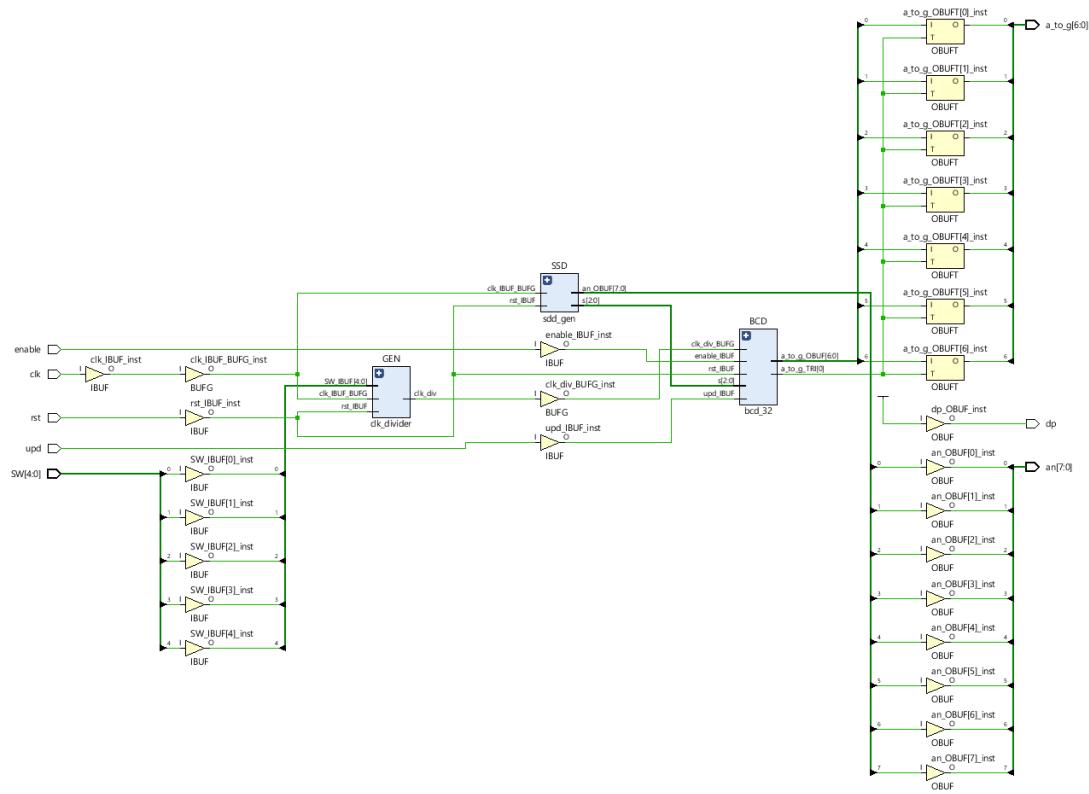
Clock Divider Generated Schematic



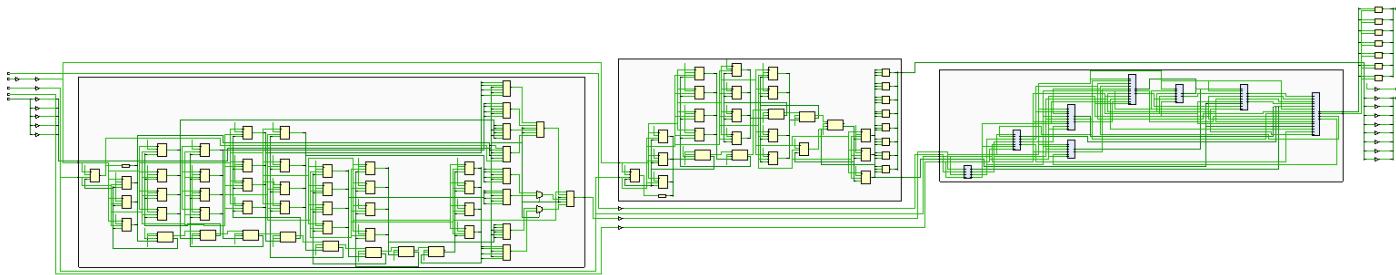
Counter Generated Schematic



Seven Segment Display Generated Schematic



Full Circuit Schematic



Full Circuit Schematic Expanded

Code Detail

The most essential part of the code was the module responsible for the individual BCD counters. The specifications for the counter were that it had to count at every positive edge of an input clock, can be switched from an up counter to a down counter, can be reset in either mode, must be able to be enabled/disabled, had to output a signal when the value of counter carried over in either mode and had to output a count of only BCD numbers (0-9). The module used behavior and data flow modeling to achieve these specifications.

The output signal at every carry over (named *en_out*) will be set as the enable signal for the next BCD. This is so that once a digit reaches 9 or 0, the next digit could respond accordingly to keep the proper total count across the 8 seven segment displays.

```
1 : timescale 1ns / 1ps
2 : module bcd_counter(
3 :     input clk,
4 :     input rst,
5 :     input en_in,
6 :     input upd,
7 :     output reg [3:0] op,
8 :     output wire en_out
9 : );
10:     always@(posedge clk or posedge rst)
11:         begin
12:             if (rst)
13:                 begin
14:                     if (upd)
15:                         op <= 4'd9;
16:                     else
17:                         op <= 4'd0;
18:                 end
19:             else
20:                 begin
21:                     if (en_in)
22:                         begin
23:                             if (upd==1'b1)
24:                                 begin
25:                                     if (op > 0)
26:                                         op <= op - 1;
27:                                     else
28:                                         op <= 4'd9;
29:                                 end
30:                             else
31:                                 begin
32:                                     if (op <9)
33:                                         op <= op+ 1;
34:                                     else
35:                                         op <= 4'd0;
36:                                 end
37:                         end
38:                     end
39:                 end
40:             assign en_out = (op[3] & op[0] & (~op[2]) & (~op[1]) & (~upd) & (en_in)) | ( (~op[3]) & (~op[0]) & (~op[2]) & (~op[1]) & (upd) & (en_in)) ;
41: endmodule
```

bcd_counter.v

The next module was the `bcd_32.v` module which instantiated 8 counter modules using a for loop and generate block. The module sets a common clock, reset, and up/down mode selection signal for all the counters. It also connects the `en_out` signal of the previous counter to the enable of the current counter via a 8 bit wire named `tmp`. The last function of this module was to combine the individual 4-bit counts of the 8 counters into a single 32-bit output.

```
1  `timescale 1ns / 1ps
2  module bcd_32(
3      input clk,
4      input rst,
5      input en,
6      input upd,
7      output wire [31:0] count
8  );
9      wire [7:0] tmp;
10
11      assign tmp[0] = en;
12
13      genvar i;
14
15
16      generate
17
18      for (i = 0; i<8 ; i = i+1)
19      begin
20          bcd_counter UNIT (
21              .clk(clk),
22              .rst(rst),
23              .en_in(tmp[i]),
24              .upd(upd),
25              .op(count [4*(i+1)-1 : 4*i]),
26              .en_out(tmp[i+1])
27          );
28      end
29      endgenerate
30 endmodule
```

bcd_32.v

The clock divider was modeled as usual with structural modeling. An input clock was used to count up a 32-bit count. Then based on a 5-bit input, a single bit of the 32-bit count was used as the new output clock. Each bit had the a frequency of the original frequency divided by a power of 2. The higher the bit, the slower the new output clock. A reset was also implemented to reset the count if needed.

```
1  `timescale 1ns / 1ps
2  module clk_divider
3    (
4      input clk,
5      input rst,
6      input [4:0] SW,
7      output reg clk_div
8    );
9
10   reg [31:0] count_tmp;
11
12   initial #1 count_tmp<=0;
13
14   always@(posedge clk or posedge rst)
15   begin
16     if(rst)
17       count_tmp<=0;
18     else
19       count_tmp<=count_tmp+1;
20   end
21
22   always@(count_tmp[SW] or rst)
23   begin
24     if(rst)
25       clk_div<=1'b0;
26     else
27       clk_div<=count_tmp[SW];
28   end
29
30 endmodule
31
```

clk_divider.v

The last component module was the seven segment display. This module controlled which display was being controlled as well as what value it was to be changed to. The seven segment display code contained 4 `always@` blocks.

The first of these `always@` blocks was a 32-bit counter which counted up at every positive edge of the input clock. Bits 17-19 of the count became the new clock. This new output clock will count from 0 to 7 as it is 3 bits and was named `s`. This `always@` block functioned as a sort of clock divider.

The next `always@` block chose which seven segment display we were updating. At every change of the new 3-bit clock (`s`), the bit of the enable output (`an`) which corresponded to the value of the clock was changed to be enabled while the rest were disabled. Since the clock counted from 0 to 7, every display was enabled and disable sequentially at the rate of the clock.

The third `always@` block assigned the value of the register `digit`. Depending on the value of `s`, a certain 4-bit chunk of the `SW` input was assigned to `digit`. The `SW` input is the collective BCD number which is to be represented on the displays. This paired with the previous `always@` is what allows the proper digit to be assigned to the proper display given a certain 32-bit BCD number.

The final `always@` at block is a decoder which takes in the digit to be displayed and assigns the output to be the correct combination of bits which light up the LEDs to display that number.

```

1 : `timescale 1ns / 1ps
2 : //`define an_control 8'b00000000
3 : `define dp_off 1'b1
4 : `define initial_digit 8'b11111111
5 :
6 module ssd_gen(
7   input [31:0] SW,
8   input clk,
9   input rst,
10  output reg [6:0] a_to_g,
11  output reg [7:0] an,
12  output wire dp
13 );
14
15 assign dp = `dp_off;
16
17 wire [2:0] s;
18 wire [7:0] aen;
19 reg [19:0] clkdiv;
20 reg [3:0] digit;
21 assign s = clkdiv[19:17];
22 assign aen = `initial_digit;
23
24
25 // clock divider
26 always@(posedge clk or posedge rst)
27 begin
28   if (rst)
29     if (rst)
30       clkdiv <= 0;
31     else
32       clkdiv <= clkdiv+1;
33 end
34
35 // digit select : ancode
36 always@(aen, s)
37 begin
38   begin
39     an = 8'b11111111;
40     if(aen[s] == 1)
41       an[s] = 0;
42   end
43 end
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80

```

ssd_gen.v

The module which brought all the components together was named *top.v*. This module had an input clock, reset, 5-bit select, enable, and mode switch. The outputs were the seven segment display signals and enables.

The instantiations of each module had to be done in a certain order. The first instantiation was the clock divider which outputted on a wire the new clock based on the 5-bit select input.

The next instantiation was that of the 32-bit BCD counter. For this instantiation, the clock divider output clock was used as the input clock while the reset, mode switch, and enable were all set to the corresponding *top.v* inputs. The output count was connected to a wire named *TMP*.

The final instantiation was the seven segment display module. This module used the original *top.v* clock and not the clock divider output. It was passed down the value of the 32-bit BCD counter as the *SW* input. The reset and the output signals were all connected to the corresponding values of the *top.v* module.

```
1  `timescale 1ns / 1ps
2  module top(
3      input clk,
4      input rst,
5      input [4:0] SW,
6      input enable,
7      input upd,
8      output wire [6:0] a_to_g,
9      output wire [7:0] an,
10     output wire dp
11 );
12
13     wire clk_div;
14     clk_divider GEN (
15         .clk(clk),
16         .rst(rst),
17         .SW(SW),
18         .clk_div(clk_div)
19     );
20
21     wire [31:0] TMP;
22
23     bcd_32 BCD (
24         .clk(clk_div),
25         .rst(rst),
26         .en(enable),
27         .upd(upd),
28         .count(TMP)
29     );
30
31     ssd_gen SSD (
32         .SW(TMP),
33         .clk(clk),
34         .rst(rst),
35         .a_to_g(a_to_g),
36         .an(an),
37         .dp(dp)
38     );
39
40 endmodule
41
```

top.v

Test Bench

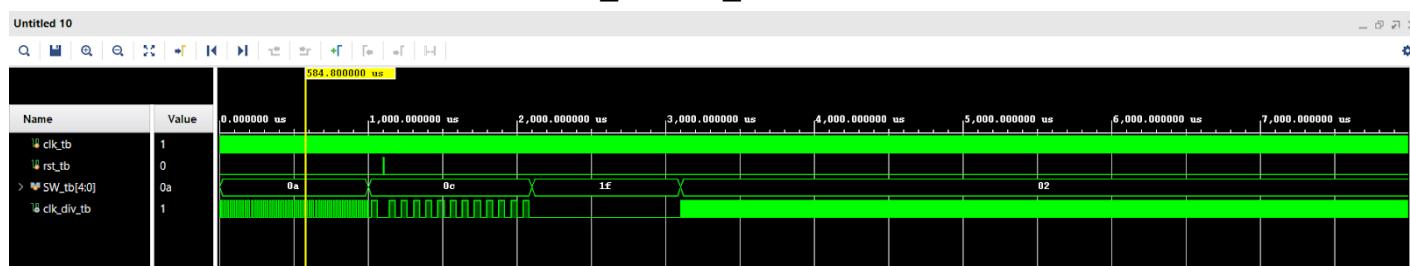
A testbench file was created for each module discussed above to check where the error occurred and how exactly to fix the error. A test bench was also created for the circuit as a whole (*top.v*).

Clock Divider Testbench

In the clock divider testbench the input clock was set to 100MHz using a *forever* statement. At every 1000000 nanoseconds, the 5-bit select was changed 4 times to cover corner cases. The reset was also set and cleared at a certain time.

```
1 : `timescale 1ns / 1ps
2 : module clk_divider_tb(
3 :
4 : );
5 :   reg clk_tb;
6 :   reg rst_tb;
7 :   reg [4:0] SW_tb;
8 :   wire clk_div_tb;
9 :
10:
11:   clk_divider test
12:   (
13:     .clk(clk_tb),
14:     .rst(rst_tb),
15:     .SW(SW_tb),
16:     .clk_div(clk_div_tb)
17:   );
18:
19:
20:   initial
21:   begin
22:     clk_tb<=1'b1;
23:     forever #5 clk_tb<=~clk_tb;
24:   end
25:
26:   initial
27:   begin
28:     SW_tb<=5'b01010;
29:     rst_tb<=0;
30:     #1000000 SW_tb<=5'b01100;
31:     #1000000 rst_tb<=1;
32:     #500
33:     rst_tb<=0;
34:     #1000000 SW_tb<=5'b11111;
35:     #1000000 SW_tb<=5'b00010;
36:   end
37:
38: endmodule
39:
```

clk_divider_tb.v



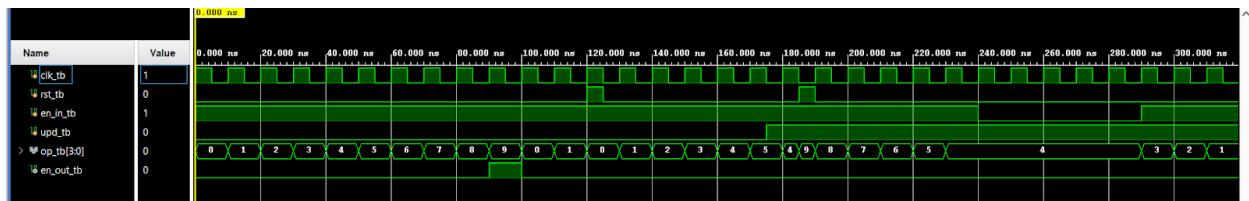
The waveform shows the output clock changing speeds at every change of the 5-bit switch. When the reset is pressed a clock cycle is delayed.

Counter Testbench

The counter testbench had to initialize a clock using an *initial* and *forever* statement. All other inputs (enable,reset, and mode switch) were changed throughout the testbench to see the effects.

```
1 | `timescale 1ns / 1ps
2 | module bcd_counter_tb(
3 |
4 | );
5 |     reg clk_tb;
6 |     reg rst_tb;
7 |     reg en_in_tb;
8 |     reg upd_tb;
9 |     wire [3:0] op_tb;
10 |    wire en_out_tb;
11 |
12 |    bcd_counter test
13 |    (
14 |        .clk(clk_tb),
15 |        .rst(rst_tb),
16 |        .en_in(en_in_tb),
17 |        .upd(upd_tb),
18 |        .op(op_tb),
19 |        .en_out(en_out_tb)
20 |    );
21 |
22 | initial
23 | begin
24 |     clk_tb<=1'b1;
25 |     forever #5 clk_tb<=~clk_tb;
26 | end
27 |
28 | initial
29 | begin
30 |     en_in_tb<=1;
31 |     upd_tb<=1'b0;
32 |     rst_tb<=1'b0;#120
33 |     rst_tb<=1'b1;#5
34 |     rst_tb<=1'b0;#50
35 |     upd_tb<=1'b1;#10
36 |     rst_tb<=1'b1;#5
37 |     rst_tb<=1'b0;#50
38 |     en_in_tb<=0; #50
39 |     en_in_tb<=1;#50;
40 | end
41 | endmodule
```

counter_tb.v



The output waveform shows the output count counting first from 0 to 9 when the mode select is high. Also whenever the output count is 9, the *en_out* signal is high as well. When the reset is pressed the count also resets to 0. When the mode select is switched to low, the output count starts to count down. Also when a reset is pressed, now the count goes to 9. The waveform also shows no change in the output when the *en_in* signal is set to low.

32 Bit Counter Testbench

The 32 bit counter testbench waveform should display the 8 digit number that will be shown in the 7 segment displays. The reset and enable pin were also changed to see how this will affect the output waveform.

```

1 :  `timescale 1ns / 1ps
2 : module bcd_32_tb(
3 : );
4 :   reg clk_tb;
5 :   reg rst_tb;
6 :   reg en_tb;
7 :   reg upd_tb;
8 :   wire [31:0] count_tb;
9 :
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41: endmodule

```

bcd_32_tb.v



The count increases once the enable pin is set to high. When the enable is low, the count pauses. The count continued once the enable was set to high again.



Once the count reaches a BCD in the ones digit, the tens digit increases. When the reset hits a positive edge, the count is set to 0. When the mode switch is set to high and the reset is pressed again, the count becomes all 9s (99999999).

Seven Segment Display Testbench

The test bench for the seven segment display had a clock initialized and tried multiple values for the *SW* which controlled the number which was to be displayed on the displays. The reset was also pressed once in the testbench.

```
1  `timescale 1ns / 1ps
2  module ssd_gen_tb(
3
4      );
5
6      reg [31:0] SW_tb;
7      reg clk_tb;
8      reg rst_tb;
9      wire [6:0] a_to_g_tb;
10     wire [7:0] an_tb;
11     wire dp_tb;
12
13     ssd_gen test
14     (
15         .SW(SW_tb),
16         .clk(clk_tb),
17         .rst(rst_tb),
18         .a_to_g(a_to_g_tb),
19         .an(an_tb),
20         .dp(dp_tb)
21     );
22
23     initial
24     begin
25         clk_tb<=1'b1;
26         forever #1 clk_tb<=~clk_tb;
27     end
28
29     initial
30     begin
31         rst_tb<=0;
32         SW_tb<=32'h88882121;#1
33         rst_tb<=1; #1
34         rst_tb<=0;#2100000
35         SW_tb<=32'h99322109;#2100000
36         SW_tb<=32'h97700779;#1050000
37         rst_tb<=1;#5
38         rst_tb<=0;#1050000
39         SW_tb<=32'h11111112;
40     end
41
42     endmodule
43
```

ssd_gen_tb.v



The *a_to_g_tb* output shows the proper signal values needed for the LEDs to light up and represent a certain digit. The *an_tb* is what chooses the seven segment display to which the number is assigned to.

When the reset is hit, the *an_tb* is reset back to 8'b01111111.

Full Circuit Testbench

The final full circuit was tested at a single speed to demonstrate the varying speed from one display to the other rather than the speed from one switch value to another.

```
1  `timescale 1ns / 1ps
2  module top_tb(
3
4    );
5
6    reg clk_tb;
7    reg rst_tb;
8    reg [4:0] SW_tb;
9    reg enable_tb;
10   reg upd_tb;
11   wire [6:0] a_to_g_tb;
12   wire [7:0] an_tb;
13   wire dp_tb;
14
15   top test
16   (
17     .clk(clk_tb),
18     .rst(rst_tb),
19     .SW(SW_tb),
20     .enable(enable_tb),
21     .upd(upd_tb),
22     .a_to_g(a_to_g_tb),
23     .an(an_tb),
24     .dp(dp_tb)
25   );
26
27   initial
28   begin
29     clk_tb<=1'b1;
30     forever #5 clk_tb<=~clk_tb;
31   end
32
33   initial
34   begin
35     enable_tb<=1;
36     upd_tb<=0;
37     rst_tb<=0;
38     SW_tb<=5'b01010;
39     #10000;
40   end
41 endmodule
```

top.v



The waveform shows the first output values of *a_to_g_tb* changing too fast to see. This is because at that particular value of *an_tb* the code is setting the number the first display which changes at a faster rate. The other display change at a much slower and so the values displayed on the later seven segment displays are more accurate (even at higher clock speeds).

Error

The error with the given code was that each counter after the first counter would change when the previous counter was at a certain value (0 when counting down and 9 when counting up). The problem is that the previous counter would stay at this number for a short duration of time. In this time the next counter would jump up a few digits before the previous counter would change.

To fix the problem we had to shorten the length of which the enable was active. To do this we added a condition for the enable_out of each counter which was that the enable_in (enable of current counter) had to be active as well. This made all the enable only active for the duration of a single digit change of the first counter. Since the first counter was the fastest, all the other counters could not change more than one digit when the enable was active.

Error

```
assign en_out = (op[3] & op[0] & (~op[2]) & (~op[1]) & (~upd)) | ((~op[3]) & (~op[0]) & (~op[2]) & (~op[1]) & (upd));
```

Solution

```
assign en_out = (op[3] & op[0] & (~op[2]) & (~op[1]) & (~upd) & (en_in)) | ((~op[3]) & (~op[0]) & (~op[2]) & (~op[1]) & (upd) & (en_in));
```

Corner Cases

Corner cases for this lab include displaying numbers properly instead of random numbers when the counter direction is changed. If the up-down switch counter is enabled, numbers must start at the highest BCD number and reset once the number reaches the smallest value 0. Similarly, when the down-up switch is enabled the BCD numbers should reset when the highest number is reached, in this case, 9. Proper testing is also required to properly display the counter, using a high frequency will cause the seven segment displays to resemble a display with all LEDs turned on.

Technical Report

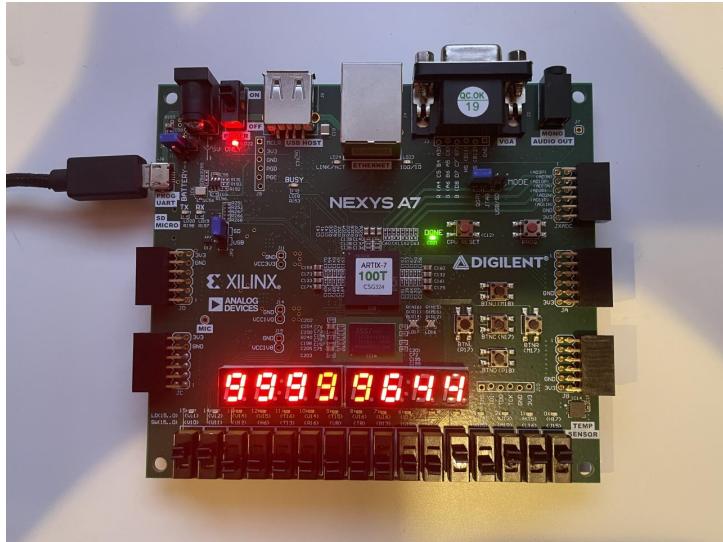
After the simulation was performed. The results obtained from the report are as follows. The total power consumption is 0.109W, 108 LUT's and 100 Flip-Flops. All within the specifications required for the assignment

Synthesis results for design

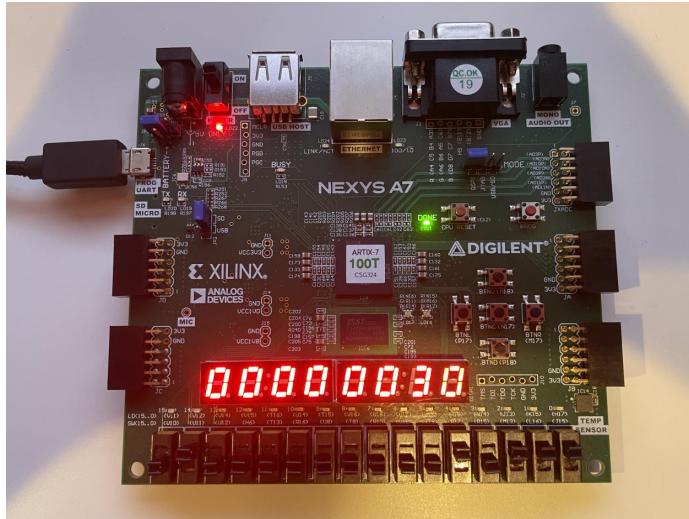
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed
✓ synth_1	constrs_1	synth_design Complete!								109	100	0.00	0	0	10/11/21, 10:57 PM	00:00:18
✓ impl_1	constrs_1	write_bitstream Complete!	7.299	0.000	0.252	0.000	0.000	0.109	0	108	100	0.00	0	0	10/11/21, 10:57 PM	00:01:11

After generating bitstream, the file was transferred to the Nexys 7. The behavior of the programmed board was identical to that of the required specifications and the switches & seven-segment display worked as expected. The counter displayed numbers properly.

Programmed Nexys 7



At select=5'b10111
Down counter activated with starting point 99999999



At select=5'b10111
Up counter activated at starting point 00000000

Conclusion

The board exhibited the expected behavior. The numbers were incrementing on the seven segment display when the switch was set to up-counter. Similarly if the counter was interrupted and switched to down-counter the counter was able to perform its functions without displaying any random numbers. The demonstration will show the board functionality and confirm that the code was fixed to address the corner cases.