Lab #4

ECE 4304 Spring 2021

Professor Aly

California State Polytechnic University, Pomona

Ahiezer Lopez	Joe Si	Sander Zuckerman
ahiezerlopez@cpp.e	joesi@cpp.edu	sazuckerman@cpp.e
du Bronco	Bronco ID:	du Bronco ID:
ID:012715521	012636091	012644671

3/3/2021

Objective:
Students will create eight BCD counters and a Seven Segment Decoder with 2 reset buttons and an up-down switch. The eight BCD counters will essentially be connected in series and as soon one BCD counts up to 9 the next BCD begins to count.There will be one reset dedicated to the seven segment and the other to the BCD counters.The switch will determine if the counter is counting up or down. In the end, numbers 0-99999999 will be displayed on the seven segments of the FPGA board.

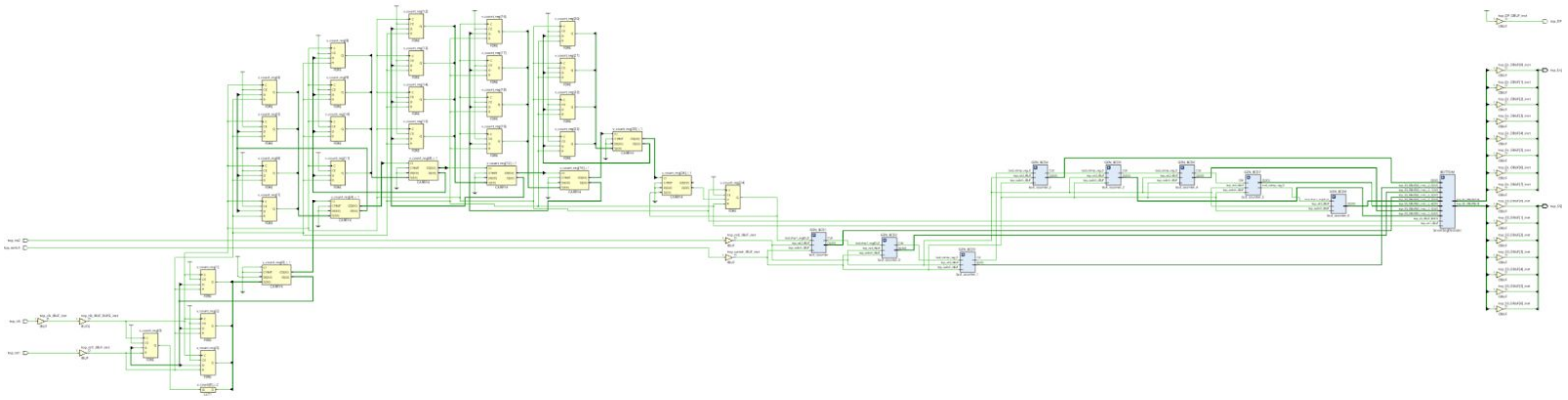
Materials:
- FPGA (Nexys A7-100T)
- Vivado Software
- Computer

Contributions:
The lab was completed with each individual's help. Ahiezer and Sander created the BCD counter module that would count from 0-99999999. Joe helped by creating the Seven Segment Module and implementing the code onto the board. The top module was worked on by everyone since it was just a simple connection between the different modules.

Design Process:
Using the flowchart below, we implement our idea on how to approach this lab. We ran into some trouble with the BCD counter because it was not counting correctly beyond 99. After taking a slightly different approach, we were able to get the entire counter to work. Given that we need 8 BCD counters, in the top module we instantiated 8 copies of the BCD counter. Overall, our design involved the use of 41 LUTs, 83 flip flops, and 0.122W.

Design:
Using the seven segment code from last week's lab, we were able to use it for this week's lab. Given that we needed a reset for the seven segment, we just added that to the code. Regarding the BCD counter, there two inputs we needed to account for, the up-down counter and the reset. We gave the reset priority and then proceeded with the up-down counter. This part was the toughest part because we need to make the following counters count at the right time. In the end, we were able to achieve the objective in this lab.

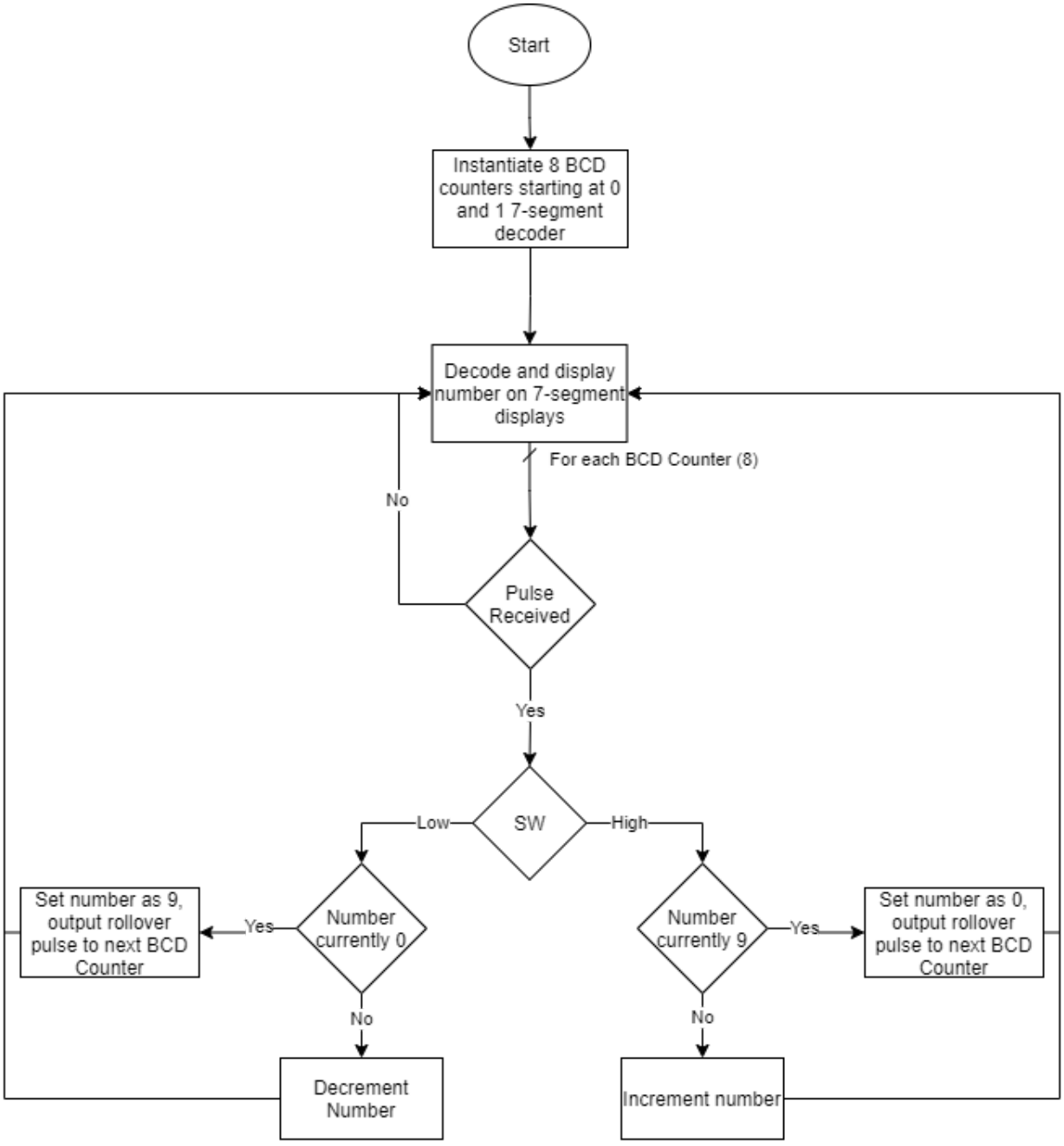
Circuit Diagram:



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
✓ synth_1	constrs_1	synth_design Complete!								41	83	0.0	0	0
✓ impl_1	constrs_1	write_bitstream Complete!	7.451	0.000	0.252	0.000	0.000	0.122	0	41	83	0.0	0	0

Implementation:

Logic Flow Diagram



Design:

Top Module

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.std_logic_unsigned.all;
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity top is
35
36            Port (
37                  top_clk       : in std_logic;
38                  top_rst1      : in std_logic;
39                  top_rst2      : in std_logic;
40                  top_switch    : in std_logic;
41                  top_En        : out std_logic_vector(7 downto 0);
42                  top_SS        : out std_logic_vector(6 downto 0);
43                  top_DP        : out std_logic
44
45                  );
46  end top;
47
48  architecture Behavioral of top is
49  signal BCD1: std_logic_vector(3 downto 0);
50  signal BCD2: std_logic_vector(3 downto 0);
51  signal BCD3: std_logic_vector(3 downto 0);
52  signal BCD4: std_logic_vector(3 downto 0);
53  signal BCD5: std_logic_vector(3 downto 0);
54  signal BCD6: std_logic_vector(3 downto 0);
55  signal BCD7: std_logic_vector(3 downto 0);
56  signal BCD8: std_logic_vector(3 downto 0);
57
58
59  signal EN1: std_logic;
60  signal EN2: std_logic;
61  signal EN3: std_logic;
62  signal EN4: std_logic;
63  signal EN5: std_logic;
64  signal EN6: std_logic;
```

```vhdl
signal EN8: std_logic;

signal delay_sig: std_logic;

component SevenSegDecoder

Port (      clk      : in std_logic;
            rst      : in std_logic;
            SSD_in_0 : in STD_LOGIC_VECTOR(3 downto 0);
            SSD_in_1 : in STD_LOGIC_VECTOR(3 downto 0);
            SSD_in_2 : in STD_LOGIC_VECTOR(3 downto 0);
            SSD_in_3 : in STD_LOGIC_VECTOR(3 downto 0);
            SSD_in_4 : in STD_LOGIC_VECTOR(3 downto 0);
            SSD_in_5 : in STD_LOGIC_VECTOR(3 downto 0);
            SSD_in_6 : in STD_LOGIC_VECTOR(3 downto 0);
            SSD_in_7 : in STD_LOGIC_VECTOR(3 downto 0);
            EN_out   : out std_logic_vector(7 downto 0);
            DP       : out std_logic;
            SSD_out_0  : out STD_LOGIC_VECTOR(6 downto 0)
          );
end component;




component bcd_counter
                Port (
                    bcd_clk: in std_logic; -- bcd clock
                    bcd_roi: in std_logic; -- bcd rollover in
                    bcd_rst: in std_logic; -- bcd reset
                    bcd_ud : in std_logic; -- up-down switch
                    bcd_out: out std_logic_vector(3 downto 0); -- BCD output
                    bcd_ro : out std_logic -- bcd rollover out
                    );
end component;


begin

DELAY: process(top_clk, top_rst1)
       variable v_count: std_logic_vector(24 downto 0);
       begin
        if(rising_edge(top_clk))then
            if(top_rst1 ='1')then
```

```
            v_count := (others => '0');
            else
            v_count := v_count + 1;
            end if;
        end if;
        delay_sig <= v_count(24);
        end process;


BOTTOM: SevenSegDecoder
    port map(
            clk      => top_clk,
            rst      => top_rst1,
            SSD_in_0 => BCD1,
            SSD_in_1 => BCD2,
            SSD_in_2 => BCD3,
            SSD_in_3 => BCD4,
            SSD_in_4 => BCD5,
            SSD_in_5 => BCD6,
            SSD_in_6 => BCD7,
            SSD_in_7 => BCD8,
            EN_out   => top_En,
            DP       => top_DP,
            SSD_out_0 => top_SS
            );


GEN_BCD1: bcd_counter
    port map(

            bcd_clk  => top_clk,
            bcd_roi  => delay_sig,
            bcd_rst  => top_rst2,
            bcd_ud   => top_switch,
            bcd_out  => BCD1,
            bcd_ro   => EN1

            );

GEN_BCD2: bcd_counter
    port map(
```

```
            bcd_clk  => top_clk,
            bcd_roi  => EN1,
            bcd_rst  => top_rst2,
            bcd_ud   => top_switch,
            bcd_out  => BCD2,
            bcd_ro   => EN2

            );

GEN_BCD3: bcd_counter
    port map(

            bcd_clk  => top_clk,
            bcd_roi  => EN2,
            bcd_rst  => top_rst2,
            bcd_ud   => top_switch,
            bcd_out  => BCD3,
            bcd_ro   => EN3

            );
GEN_BCD4: bcd_counter
    port map(

            bcd_clk  => top_clk,
            bcd_roi  => EN3,
            bcd_rst  => top_rst2,
            bcd_ud   => top_switch,
            bcd_out  => BCD4,
            bcd_ro   => EN4

            );

GEN_BCD5: bcd_counter
    port map(

            bcd_clk  => top_clk,
            bcd_roi  => EN4,
            bcd_rst  => top_rst2,
            bcd_ud   => top_switch,
            bcd_out  => BCD5,
            bcd_ro   => EN5

            );
GEN_BCD6: bcd_counter
```

```
GEN_BCD6: bcd_counter
    port map(

            bcd_clk  => top_clk,
            bcd_roi  => EN5,
            bcd_rst  => top_rst2,
            bcd_ud   => top_switch,
            bcd_out  => BCD6,
            bcd_ro   => EN6

            );
GEN_BCD7: bcd_counter
    port map(

            bcd_clk  => top_clk,
            bcd_roi  => EN6,
            bcd_rst  => top_rst2,
            bcd_ud   => top_switch,
            bcd_out  => BCD7,
            bcd_ro   => EN7

            );
GEN_BCD8: bcd_counter
    port map(

            bcd_clk  => top_clk,
            bcd_roi  => EN7,
            bcd_rst  => top_rst2,
            bcd_ud   => top_switch,
            bcd_out  => BCD8,
            bcd_ro   => EN8

            );


end Behavioral;

```

BCD Counter Module:

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.numeric_std.all;
25  use IEEE.std_logic_unsigned.all;
26
27  -- Uncomment the following library declaration if using
28  -- arithmetic functions with Signed or Unsigned values
29  --use IEEE.NUMERIC_STD.ALL;
30
31  -- Uncomment the following library declaration if instantiating
32  -- any Xilinx leaf cells in this code.
33  --library UNISIM;
34  --use UNISIM.VComponents.all;
35
36  entity bcd_counter is
37                Port (
38                      bcd_clk: in std_logic; -- bcd clock
39                      bcd_roi: in std_logic; -- bcd rollover in
40                      bcd_rst: in std_logic; -- bcd reset
41                      bcd_ud : in std_logic; -- up-down switch
42                      bcd_out: out std_logic_vector(3 downto 0); -- BCD output
43                      bcd_ro : out std_logic -- bcd rollover out
44                      );
45  end bcd_counter;
46
47  architecture Behavioral of bcd_counter is
48
49  signal bcd_tmp1: std_logic_vector(3 downto 0);
50  signal bcd_rotmp: std_logic;
51
52  begin
53
54  GEN_BCD1: process(bcd_clk, bcd_rst, bcd_roi)
55            begin
56            if(rising_edge(bcd_clk)) then
57              bcd_rotmp <= '0';    -- reset rollover to 0
58            end if;
59            if(bcd_rst = '1')then -- reset output to 0 if 1
60              bcd_tmp1 <= (others => '0');
61              bcd_rotmp <= '0';
62            elsif(rising_edge(bcd_roi))then
63                  if(bcd_ud = '1')then -- count up if 1
64                           bcd_tmp1 <= bcd_tmp1 + 1;
65                           bcd_rotmp <= '0'; -- reset rollover to 0
```

```
                        if(bcd_tmp1 = "1001")then -- reset to 0 if 9
                           bcd_tmp1 <= "0000";
                           bcd_rotmp <= '1'; -- send rollover pulse
                           end if;
                    else if(bcd_ud = '0')then --count down if 0
                         bcd_tmp1 <= bcd_tmp1 - 1;
                         bcd_rotmp <= '0';
                           if(bcd_tmp1 = "0000")then -- if equal to 0 set to 9
                               bcd_tmp1 <= "1001";
                               bcd_rotmp <= '1'; -- send rollover pulse
                               end if;
                    end if;
                end if;
            end if;


end process;
bcd_out <= bcd_tmp1;
bcd_ro <= bcd_rotmp;
end Behavioral;
```

Seven Segment Module:

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.std_logic_unsigned.all;
25
26  -- Uncomment the following library declaration if using
27  -- arithmetic functions with Signed or Unsigned values
28  --use IEEE.NUMERIC_STD.ALL;
29
30  -- Uncomment the following library declaration if instantiating
31  -- any Xilinx leaf cells in this code.
32  --library UNISIM;
33  --use UNISIM.VComponents.all;
34
35  entity SevenSegDecoder is
36      Port ( clk      : in std_logic;
37             rst      : in std_logic;
38             SSD_in_0 : in STD_LOGIC_VECTOR(3 downto 0);
39             SSD_in_1 : in STD_LOGIC_VECTOR(3 downto 0);
40             SSD_in_2 : in STD_LOGIC_VECTOR(3 downto 0);
41             SSD_in_3 : in STD_LOGIC_VECTOR(3 downto 0);
42             SSD_in_4 : in STD_LOGIC_VECTOR(3 downto 0);
43             SSD_in_5 : in STD_LOGIC_VECTOR(3 downto 0);
44             SSD_in_6 : in STD_LOGIC_VECTOR(3 downto 0);
45             SSD_in_7 : in STD_LOGIC_VECTOR(3 downto 0);
46             EN_out   : out std_logic_vector(7 downto 0);
47             DP       : out std_logic;
48             SSD_out_0  : out STD_LOGIC_VECTOR(6 downto 0)
49           );
50  end SevenSegDecoder;
51
52
53
54
55  architecture Behavioral of SevenSegDecoder is
56
57
58  signal counter_out: std_logic_vector(2 downto 0);
59  signal dout_out: std_logic_vector(5 downto 0);
60  signal E: std_logic_vector(7 downto 0);
61
62
63  begin
64
65  process(clk, rst)
```

```
begin
    if(rising_edge(clk))then -- need to fix the reset here
        if(rst ='1')then
            v_count := (others => '0');
          else
          v_count := v_count + 1;
          end if;
    end if;
    counter_out <= v_count(18 downto 16);
    end process;


process(counter_out, SSD_in_0, SSD_in_1)
begin
    case counter_out is
        when "000" => E <= "00000001";
                      dout_out <= '1' & SSD_in_0(3 downto 0) & '1';
        when "001" => E <= "00000010";
                      dout_out <= '1' & SSD_in_1(3 downto 0) & '1';
        when "010" => E <= "00000100";
                      dout_out <= '1' & SSD_in_2(3 downto 0) & '1';
        when "011" => E <= "00001000";
                      dout_out <= '1' & SSD_in_3(3 downto 0) & '1';
        when "100" => E <= "00010000";
                      dout_out <= '1' & SSD_in_4(3 downto 0) & '1' ;
        when "101" => E <= "00100000";
                      dout_out <= '1' & SSD_in_5(3 downto 0) & '1';
        when "110" => E <= "01000000";
                      dout_out <= '1' & SSD_in_6(3 downto 0) & '1';
        when "111" => E <= "10000000";
                      dout_out <= '1' & SSD_in_7(3 downto 0) & '1';
        when others => E <= "11111111";
    end case;
 end process;

En_out <= not E;

process(dout_out) is
begin
if(rst = '1')then
    case dout_out(4 downto 1) is
        when "0000" => SSD_out_0(6 downto 0) <= "0000001";
        when "0001" => SSD_out_0(6 downto 0) <= "0000001";
        when "0010" => SSD_out_0(6 downto 0) <= "0000001";
```

```
110         when "0010" => SSD_out_0(6 downto 0) <= "0000001";
111         when "0011" => SSD_out_0(6 downto 0) <= "0000001";
112         when "0100" => SSD_out_0(6 downto 0) <= "0000001";
113         when "0101" => SSD_out_0(6 downto 0) <= "0000001";
114         when "0110" => SSD_out_0(6 downto 0) <= "0000001";
115         when "0111" => SSD_out_0(6 downto 0) <= "0000001";
116         when "1000" => SSD_out_0(6 downto 0) <= "0000001";
117         when "1001" => SSD_out_0(6 downto 0) <= "0000001";
118         when others => SSD_out_0(6 downto 0) <= "1111111";
119     end case;
120     else
121     case dout_out(4 downto 1) is
122         when "0000" => SSD_out_0(6 downto 0) <= "0000001";
123         when "0001" => SSD_out_0(6 downto 0) <= "1001111";
124         when "0010" => SSD_out_0(6 downto 0) <= "0010010";
125         when "0011" => SSD_out_0(6 downto 0) <= "0000110";
126         when "0100" => SSD_out_0(6 downto 0) <= "1001100";
127         when "0101" => SSD_out_0(6 downto 0) <= "0100100";
128         when "0110" => SSD_out_0(6 downto 0) <= "0100000";
129         when "0111" => SSD_out_0(6 downto 0) <= "0001111";
130         when "1000" => SSD_out_0(6 downto 0) <= "0000000";
131         when "1001" => SSD_out_0(6 downto 0) <= "0000100";
132         when others => SSD_out_0(6 downto 0) <= "1111111";
133     end case;
134     end if;
135 
136 end process;
137 
138 DP <= dout_out(0);
139 end Behavioral;
```

Test Bench:

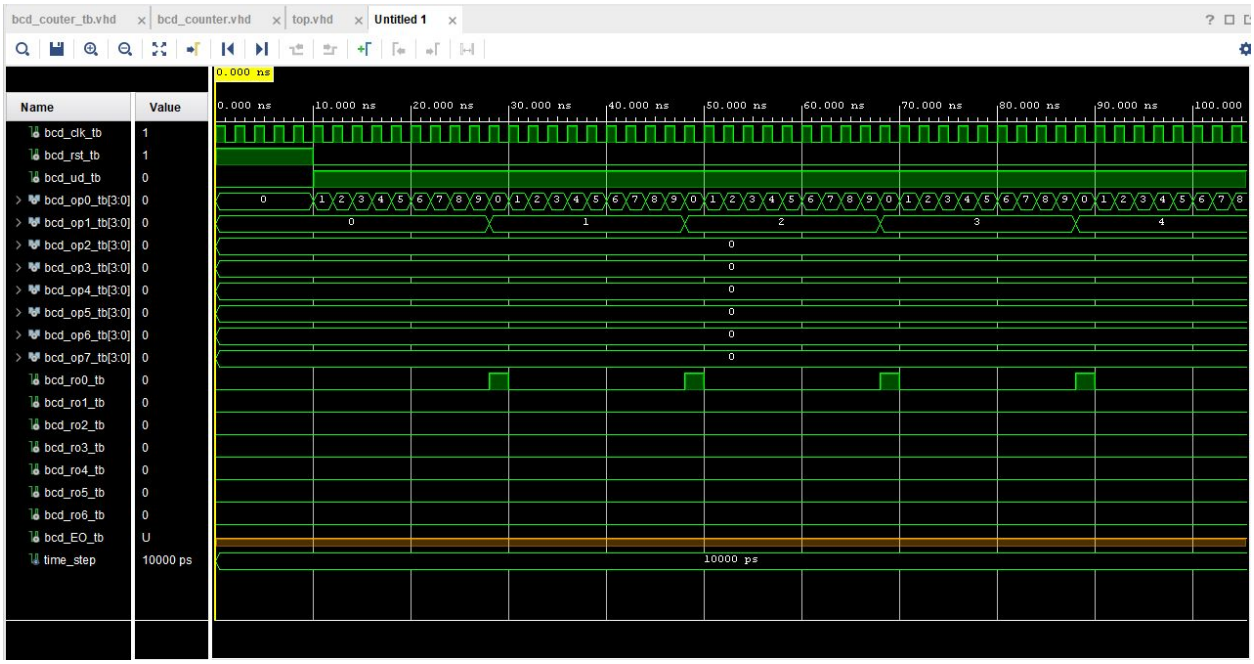
```
34      entity bcd_tb is
35      --  Port ( );
36      end bcd_tb;
37
38      architecture Behavioral of bcd_tb is
39
40      component BCD_counter
41                  Port (
42                        bcd_clk: in std_logic; -- bcd clock
43                        bcd_roi: in std_logic; -- Roll over in
44                        bcd_rst: in std_logic; -- bcd reset
45                        bcd_ud : in std_logic; -- up-down switch
46                        bcd_out: out std_logic_vector(3 downto 0); -- BCD output
47                        bcd_ro : out std_logic -- bcd rollover out
48                        );
49      end component;
50
51      constant time_step: time:=10ns;
52
53      --testbench list of signals
54      signal bcd_clk_tb: std_logic;
55      signal bcd_rst_tb: std_logic;
56      signal bcd_ud_tb: std_logic;
57
58      signal bcd_op0_tb: std_logic_vector(3 downto 0);
59      signal bcd_op1_tb: std_logic_vector(3 downto 0);
60      signal bcd_op2_tb: std_logic_vector(3 downto 0);
61      signal bcd_op3_tb: std_logic_vector(3 downto 0);
62      signal bcd_op4_tb: std_logic_vector(3 downto 0);
63      signal bcd_op5_tb: std_logic_vector(3 downto 0);
64      signal bcd_op6_tb: std_logic_vector(3 downto 0);
65      signal bcd_op7_tb: std_logic_vector(3 downto 0);
```

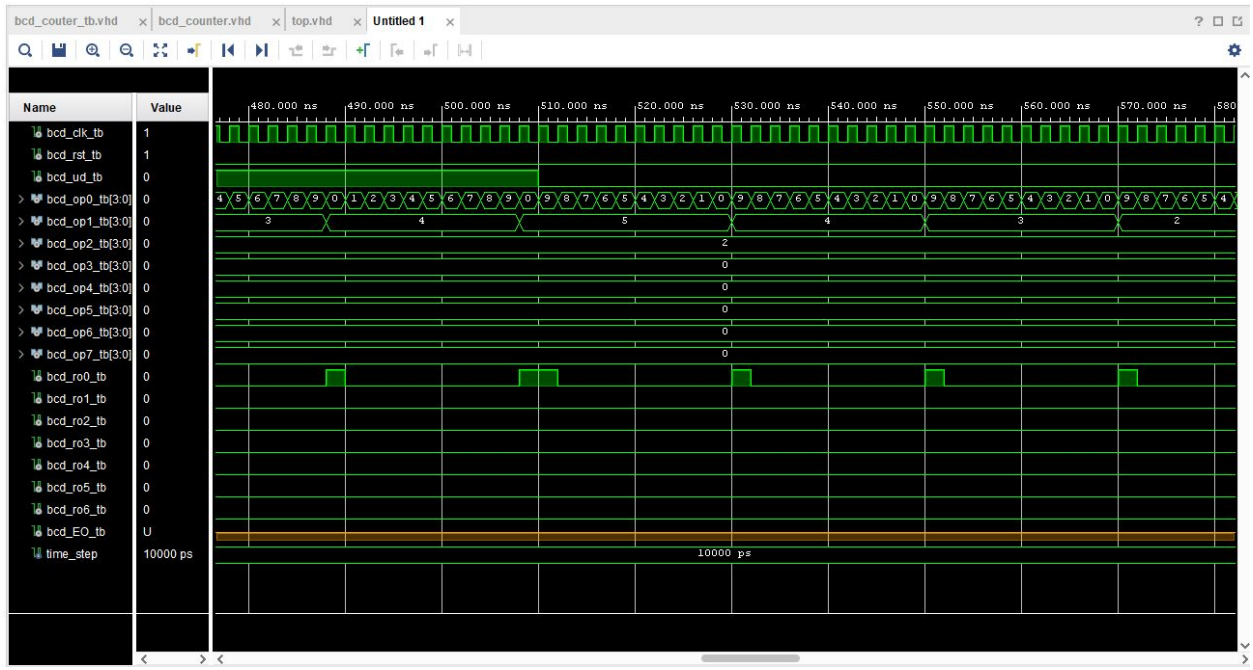
```
68      signal bcd_ro0_tb: std_logic;
69      signal bcd_ro1_tb: std_logic;
70      signal bcd_ro2_tb: std_logic;
71      signal bcd_ro3_tb: std_logic;
72      signal bcd_ro4_tb: std_logic;
73      signal bcd_ro5_tb: std_logic;
74      signal bcd_ro6_tb: std_logic;
75
76      signal bcd_EO_tb: std_logic;
77
78
79      begin
80
81      BCD0: BCD_counter
82          port map(
83              bcd_clk => bcd_clk_tb,
84              bcd_roi => bcd_clk_tb,
85              bcd_rst => bcd_rst_tb,
86              bcd_ud  => bcd_ud_tb,
87              bcd_out => bcd_op0_tb,
88              bcd_ro => bcd_ro0_tb
89              );
90
91      BCD1: BCD_counter
92          port map(
93              bcd_clk => bcd_clk_tb,
94              bcd_roi => bcd_ro0_tb,
95              bcd_rst => bcd_rst_tb,
96              bcd_ud  => bcd_ud_tb,
97              bcd_out => bcd_op1_tb,
98              bcd_ro => bcd_ro1_tb
99              );
```

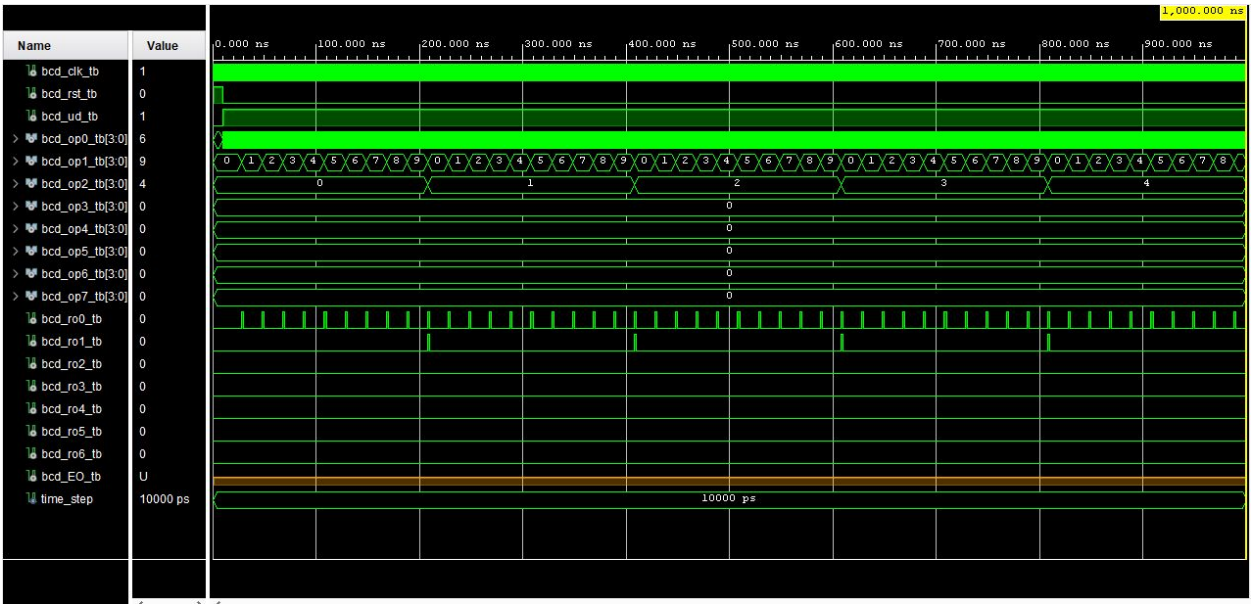
```
101     BCD2: BCD_counter
102         port map(
103             bcd_clk => bcd_clk_tb,
104             bcd_roi => bcd_ro1_tb,
105             bcd_rst => bcd_rst_tb,
106             bcd_ud  => bcd_ud_tb,
107             bcd_out => bcd_op2_tb,
108             bcd_ro => bcd_ro2_tb
109             );
110
111     BCD3: BCD_counter
112         port map(
113             bcd_clk => bcd_clk_tb,
114             bcd_roi => bcd_ro2_tb,
115             bcd_rst => bcd_rst_tb,
116             bcd_ud  => bcd_ud_tb,
117             bcd_out => bcd_op3_tb,
118             bcd_ro => bcd_ro3_tb
119             );
120
121     BCD4: BCD_counter
122         port map(
123             bcd_clk => bcd_clk_tb,
124             bcd_roi => bcd_ro3_tb,
125             bcd_rst => bcd_rst_tb,
126             bcd_ud  => bcd_ud_tb,
127             bcd_out => bcd_op4_tb,
128             bcd_ro => bcd_ro4_tb
129             );
130
```

```
131     BCD5: BCD_counter
132         port map(
133             bcd_clk => bcd_clk_tb,
134             bcd_roi => bcd_ro4_tb,
135             bcd_rst => bcd_rst_tb,
136             bcd_ud  => bcd_ud_tb,
137             bcd_out => bcd_op5_tb,
138             bcd_ro => bcd_ro5_tb
139             );
140
141     BCD6: BCD_counter
142         port map(
143             bcd_clk => bcd_clk_tb,
144             bcd_roi => bcd_ro5_tb,
145             bcd_rst => bcd_rst_tb,
146             bcd_ud  => bcd_ud_tb,
147             bcd_out => bcd_op6_tb,
148             bcd_ro => bcd_ro6_tb
149             );
150
151     BCD7: BCD_counter
152         port map(
153             bcd_clk => bcd_clk_tb,
154             bcd_roi => bcd_ro6_tb,
155             bcd_rst => bcd_rst_tb,
156             bcd_ud  => bcd_ud_tb,
157             bcd_out => bcd_op7_tb
158             );
```

```
161     clk_gen: process
162             begin
163             bcd_clk_tb <= '1';
164             wait for 1ns;
165             bcd_clk_tb <= '0';
166             wait for 1ns;
167
168
169             end process;
170
171
172
173     TST_CASE_1: process
174                 begin
175                 --test case 1: testing the upcounter moving from the right starting point
176                  bcd_rst_tb <= '1';
177                  bcd_ud_tb <= '0';
178                  wait for time_step;
179                  bcd_rst_tb <= '0';
180                  bcd_ud_tb <= '1';
181                  wait for 1000ns;
182                  bcd_rst_tb <= '0';
183                  bcd_ud_tb <= '0';
184                  wait for 500ns;
185
186
187
188                  wait;
189
190                 end process;
191
```


bcd_couter_tb.vhd
bcd_counter.vhd
top.vhd
Untitled 1
Name
Value
bcd_clk_tb 1
bcd_rst_tb 1
bcd_ud_tb 0
bcd_op0_tb[3:0] 0
bcd_op1_tb[3:0] 0
bcd_op2_tb[3:0] 0
bcd_op3_tb[3:0] 0
bcd_op4_tb[3:0] 0
bcd_op5_tb[3:0] 0
bcd_op6_tb[3:0] 0
bcd_op7_tb[3:0] 0
bcd_ro0_tb 0
bcd_ro1_tb 0
bcd_ro2_tb 0
bcd_ro3_tb 0
bcd_ro4_tb 0
bcd_ro5_tb 0
bcd_ro6_tb 0
bcd_EO_tb U
time_step 10000 ps
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000
10000 ps


bcd_couter_tb.vhd
bcd_counter.vhd
top.vhd
Untitled 1
Name
Value
bcd_clk_tb 1
bcd_rst_tb 1
bcd_ud_tb 0
bcd_op0_tb[3:0] 0
bcd_op1_tb[3:0] 0
bcd_op2_tb[3:0] 0
bcd_op3_tb[3:0] 0
bcd_op4_tb[3:0] 0
bcd_op5_tb[3:0] 0
bcd_op6_tb[3:0] 0
bcd_op7_tb[3:0] 0
bcd_ro0_tb 0
bcd_ro1_tb 0
bcd_ro2_tb 0
bcd_ro3_tb 0
bcd_ro4_tb 0
bcd_ro5_tb 0
bcd_ro6_tb 0
bcd_EO_tb U
time_step 10000 ps
480.000 ns
490.000 ns
500.000 ns
510.000 ns
520.000 ns
530.000 ns
540.000 ns
550.000 ns
560.000 ns
570.000 ns
10000 ps



Analysis:

The BCD counters for this lab each independently keep track of their own number, but are influenced by a pulse that leads into it. The least significant BCD counter is influenced by a divided down system clock (which in this lab is about a 4 Hz clock). That BCD counter has a rollover output which is connected into the pulse input for the second least significant BCD counter, and this model “waterfalls” through all eight BCD counters.

When a BCD counter is set to count up, is currently at 9, and receives an increment pulse, it is set to 0 and outputs a rollover pulse. Similarly, when a BCD counter is set to count down, is currently at 0, and receives a decrement pulse, it is set to 9 and outputs a rollover pulse. In this fashion, the total design effectively creates a base-10 BCD waterfall up-down counter, in a very similar fashion to a base-2 waterfall up-down counter.

The numbers of each BCD are then decoded and displayed onto a 7-segment display, which are each connected to a proper multiplexer and 100 MHz system clock to cycle through each display as usual. As the waveform above shows, the logic shown above successfully works.

Conclusion:

In this lab, we were successfully able to create a bcd counter that used all eight 7-segment

displays which continuously counted up till 9999999. The main problem was creating the carry over from each 7-segment and the top module where we needed to create a delay counter. With help from older verilog code, we were able to solve these problems with ease. The 7-segment module was only edited slightly to view each 7-segment as its own individual digit. The BCD converter was changed significantly to account for counting up and counting down. With thorough testing on both the board and simulation, we were able to have the counter go up or down at a readable pace with each number changing how it is supposed to. This lab allowed us to further understand time delays in VHDL as well as carry overs between bits.