

Project 2 Writeup

To simplify reading, I have added the full list of the code to the bottom of this page.

Part 1

Part one was straight forward, however understanding what and how structural VHDL worked was very confusing. During the first part, we googled a lot and used the slides to understand this. The first part to understand what structural VHDL is understanding that structural VHDL defines a physical implementation. In this project we declared the components that we used in our implementation, this is C++ analogy of function prototypes, it tells VHDL that this black box is used somewhere, and here are the input and output. This was defined after the architectural statement and before the begin statement. An example is shown here.

```
architecture Behavioral of top is
    component encoder
        Port (
            hex_in : in  STD_LOGIC_VECTOR(3 downto 0);
            a      : out STD_LOGIC;
            b      : out STD_LOGIC;
            c      : out STD_LOGIC;
            d      : out STD_LOGIC;
            e      : out STD_LOGIC;
            f      : out STD_LOGIC;
            g      : out STD_LOGIC
        );
    end component;
begin
```

After we declared the components, we needed to create wires or traces that would connect the components together. From using the 4-bit adder implementation used in the slides we, created signals that represented a `STD_LOGIC_VECTOR` for each input and output.

```
signal a_in, b_in, c_in, d_in, e_in, f_in, g_in : std_logic_vector(3 downto 0);
signal seg_out : std_logic_vector(6 downto 0);
signal dp_out : std_logic;
signal an_out : std_logic_vector(3 downto 0);
```

We then instantiated the parts and connected each encoder to the respective input value in the `ssd_muxer`. You can see an example here:

```
REG1: encoder port map(
    hex_in => sw(15 downto 12),
    a => a_in(3),
    b => b_in(3),
    c => c_in(3),
    d => d_in(3),
    e => e_in(3),
    f => f_in(3),
    g => g_in(3)
);
```

Project 2 Writeup

After this, we came to the test bench. Because the directions in the pdf was not very clear on what we had to do specifically, we ran multitude of trails, hoping to get the same waveform in the pdf. We attempted to change the clock signal but that did not really change anything. We also tried using a counter, similar to project 1 and decided to go through all the values that a 16bit value can hold, but quickly realized that it may not be the correct path. After many failed attempt, the simplest solution worked out and we created a process that assigned the proper vale to sw and waited for 10ns.

```
sw_proc: process
begin
    sw <= x"7b10";
    wait for clk_per;
end process sw_proc;
```

This concluded our first part of the project. We were surprised on how long this took us, especially trying to figure out how to get the test bench to work. The 4-bit encoder in the slides, helped a lot with understanding the syntax of Structural VHDL

Part 2

Part 2 was by far the hardest part of the project, just simply because we had no idea what we had to do. When we read that we had to create an accumulator, we assumed we had to create an adder, which by pressings one of the buttons adds the binary value of the switches to the current output of the display then displays the new number. The other button would clear the display to 0. We also though that we needed to use decimal to output to the display, so we went on a google spree learning BCD (binary coded decimal) and how to implement it. We just gave up on this part and decided that it would be best if we moved to the 3rd part which seemed easier and was a larger part of our grade. 2 days before the project was due, one of our group members was talking about the project and then realized that we were entirely wrong about everything and we quickly raced against the clock to fix the project and get it to work. Instead of making an 'accumulator', we had accumulated the switches, where the 7th switch was -2^7 . In essence we were making a 2's complement display. After that part was cleared up, it made coding the project much simpler and didn't take us very long to get it working. The other part that was difficult was the negative sign, we realized that our encoder was not capable of taking any more inputs (hex_in was a 4-bit value, and we were using all those bits to encode the number). We then realized that we can reserve one of the displays solely for the negative and manually set that display to a negative if we want.

```
g_in(2) <= '1';
else -- not negative
    accum <= sw;
    g_in(2) <= '0';
end if;
```

Project 2 Writeup

The largest part of VHDL is trying to understand the syntax, and the limits of the programming language.

Part 3

I believe part 3 was by far the easiest part of the project, mostly because after we did part 1, we had a working knowledge of VHDL syntax and what structural VHDL is, and after delving into processes and for loops and if statements from our attempt at a BCD in part 2, we knew most of the syntax this part required. The logic was fairly simple, we originally had the top entity divided into two processes but kept getting weird errors and critical warnings, so we switched to a single process that handled everything. We also learned about variables vs. signals. Variables are used in processes while signals are used in the actual definition of the entity. We created a temp variable to store our next value for the 15th LED then at the end of the process 'copied' the value from the variable to the actual signal.

```
GUESS: process (test)
    variable led_15 : std_logic := '0';
begin
    if (reset = '1') then
        value <= sw;
        led_15 := '0';
    end if;
```

We did have this weird issue where the board would crash and disconnect from the computer, but after testing on a friend's board, we chalked it up to the USB wire and not the code.

```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  -- Uncomment the following library declaration if using
5  -- arithmetic functions with Signed or Unsigned values
6  --use IEEE.NUMERIC_STD.ALL;
7
8  -- Uncomment the following library declaration if instantiating
9  -- any Xilinx leaf cells in this code.
10 --library UNISIM;
11 --use UNISIM.VComponents.all;
12
13 entity encoder is
14     Port (
15         hex_in : in STD_LOGIC_VECTOR(3 DOWNTO 0);
16         a : out STD_LOGIC;
17         b : out STD_LOGIC;
18         c : out STD_LOGIC;
19         d : out STD_LOGIC;
20         e : out STD_LOGIC;
21         f : out STD_LOGIC;
22         g : out STD_LOGIC
23     );
24 end encoder;
25
26 architecture Behavioral of encoder is
27
28     -- temporary signal to make our assignment of values (a through g) simpler
29     signal seven_seg : std_logic_vector(6 downto 0);
30
31 begin
32
33     --seven_seg <= -- Enter your code to assign values for the entire 7 bits (a through
34     g) based on the hex input
35     process(hex_in)
36     begin
37         case hex_in is
38             when "0000" => seven_seg <= "1111110";
39             when "0001" => seven_seg <= "0110000";
40             when "0010" => seven_seg <= "1101101";
41             when "0011" => seven_seg <= "1111001";
42
43             when "0100" => seven_seg <= "0110011";
44             when "0101" => seven_seg <= "1011011";
45             when "0110" => seven_seg <= "1011111";
46             when "0111" => seven_seg <= "1110000";
47
48             when "1000" => seven_seg <= "1111111";
49             when "1001" => seven_seg <= "1111011";
50             when "1010" => seven_seg <= "1110111";
51             when "1011" => seven_seg <= "0011111";
52
53             when "1100" => seven_seg <= "1001110";
54             when "1101" => seven_seg <= "0111101";
55             when "1110" => seven_seg <= "1001111";
56             when "1111" => seven_seg <= "1000111";
57
58             when others => seven_seg <= "0000000";
59
60         end case;
61     end process;
62
63     -- Extract each individual bit and assign it to the 7 outputs
64     a <= seven_seg(6);
65     b <= seven_seg(5);
66     c <= seven_seg(4);
67     d <= seven_seg(3);

```

```
66      e <= seven_seg(2) ;  
67      f <= seven_seg(1) ;  
68      g <= seven_seg(0) ;  
69  
70  end Behavioral ;  
71
```

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 11/16/2017 02:45:02 PM
6  -- Design Name:
7  -- Module Name: top - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
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     Port (
36         sw : in std_logic_vector(15 downto 0);
37         clk : in std_logic;
38         seg : out std_logic_vector(6 downto 0);
39         dp : out std_logic;
40         an : out std_logic_vector(3 downto 0);
41         led : out std_logic_vector(15 downto 0) );
42 end top;
43
44 architecture Behavioral of top is
45     component ssd_muxer is
46         Port (
47             a_in : in std_logic_vector(3 downto 0);
48             b_in : in std_logic_vector(3 downto 0);
49             c_in : in std_logic_vector(3 downto 0);
50             d_in : in std_logic_vector(3 downto 0);
51             e_in : in std_logic_vector(3 downto 0);
52             f_in : in std_logic_vector(3 downto 0);
53             g_in : in std_logic_vector(3 downto 0);
54             decp0_in : in std_logic;
55             decp1_in : in std_logic;
56             decp2_in : in std_logic;
57             decp3_in : in std_logic;
58             seg_out : out std_logic_vector(6 downto 0);
59             dp_out : out std_logic;
60             an_out : out std_logic_vector(3 downto 0);
61             clk : in STD_LOGIC
62         );
63     end component;
64     signal a_in, b_in, c_in, d_in, e_in, f_in, g_in : std_logic_vector(3 downto 0);
65     signal seg_out : std_logic_vector(6 downto 0);
66     signal dp_out : std_logic;

```

```

67     signal an_out : std_logic_vector(3 downto 0);
68
69     component encoder
70     Port (
71         hex_in : in  STD_LOGIC_VECTOR(3 DOWNT0 0);
72         a      : out STD_LOGIC;
73         b      : out STD_LOGIC;
74         c      : out STD_LOGIC;
75         d      : out STD_LOGIC;
76         e      : out STD_LOGIC;
77         f      : out STD_LOGIC;
78         g      : out STD_LOGIC
79     );
80     end component;
81
82     -- signal st, ed : integer := 0;
83
84 begin
85     seg <= seg_out;
86     dp <= dp_out;
87     an <= an_out;
88     led <= sw;
89
90     SSD_MUX: ssd_muxer port map(
91         a_in => a_in,
92         b_in => b_in,
93         c_in => c_in,
94         d_in => d_in,
95         e_in => e_in,
96         f_in => f_in,
97         g_in => g_in,
98
99         decp0_in => '0',
100        decp1_in => '0',
101        decp2_in => '0',
102        decp3_in => '0',
103
104        seg_out => seg_out,
105        dp_out => dp_out,
106        an_out => an_out,
107
108        clk => clk
109    );
110
111    --      -- https://www.ics.uci.edu/~jmoorkan/vhdlref/generate.html
112    --      REG_ENC: for I in 0 to 3 generate
113    --      --
114    --      https://stackoverflow.com/questions/10375858/how-to-slice-an-std-logic-vector-in-vhdl
115    --      REG_PMP: encoder port map(
116    --          hex_in => sw((I * 4) downto (((I + 1)) * 4 - 1)),
117    --          a => a_in(I),
118    --          b => b_in(I),
119    --          c => c_in(I),
120    --          d => d_in(I),
121    --          e => e_in(I),
122    --          f => f_in(I),
123    --          g => g_in(I)
124    --      );
125    --      end generate;
126
127    REG1: encoder port map(
128        hex_in => sw(15 downto 12),
129        a => a_in(3),
130        b => b_in(3),
131        c => c_in(3),

```

```
132         d => d_in(3),
133         e => e_in(3),
134         f => f_in(3),
135         g => g_in(3)
136     );
137
138     REGE2: encoder port map(
139         hex_in => sw(11 downto 8),
140         a => a_in(2),
141         b => b_in(2),
142         c => c_in(2),
143         d => d_in(2),
144         e => e_in(2),
145         f => f_in(2),
146         g => g_in(2)
147     );
148
149     REGE3: encoder port map(
150         hex_in => sw(7 downto 4),
151         a => a_in(1),
152         b => b_in(1),
153         c => c_in(1),
154         d => d_in(1),
155         e => e_in(1),
156         f => f_in(1),
157         g => g_in(1)
158     );
159
160     REGE4: encoder port map(
161         hex_in => sw(3 downto 0),
162         a => a_in(0),
163         b => b_in(0),
164         c => c_in(0),
165         d => d_in(0),
166         e => e_in(0),
167         f => f_in(0),
168         g => g_in(0)
169     );
170
171     end Behavioral;
172
```



```

1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  -- Uncomment the following library declaration if using
5  -- arithmetic functions with Signed or Unsigned values
6  --use IEEE.NUMERIC_STD.ALL;
7
8  -- Uncomment the following library declaration if instantiating
9  -- any Xilinx leaf cells in this code.
10 --library UNISIM;
11 --use UNISIM.VComponents.all;
12
13 entity top_tb is
14 -- Port ( );
15 end top_tb;
16
17 architecture Behavioral of top_tb is
18
19     --this entity needs to be defined in your top.vhd file
20     component top is
21     Port (
22         sw      : in  STD_LOGIC_VECTOR (15 downto 0);
23         seg     : out std_logic_vector(6 downto 0);
24         dp      : out std_logic;
25         an      : out std_logic_vector(3 downto 0);
26         led     : out STD_LOGIC_VECTOR (15 downto 0);
27         clk     : in  std_logic
28     );
29     end component;
30
31     --clock period. Set to 100 MHz here
32     constant clk_per : time := 10ns;
33
34     --signals to do the binding to the "top" entity
35     signal sw      : std_logic_vector(15 downto 0);
36     signal seg     : std_logic_vector(6 downto 0);
37     signal dp      : std_logic;
38     signal an      : std_logic_vector(3 downto 0);
39     signal led     : STD_LOGIC_VECTOR (15 downto 0);
40     signal clk     : std_logic := '0';
41
42     --signal counter : unsigned(15 downto 0) := '0000000000000000';
43 begin
44
45     uut: top port map (
46         sw      => sw,
47         seg     => seg,
48         dp      => dp,
49         an      => an,
50         led     => led,
51         clk     => clk
52     );
53
54     --add code here to test your outputs for correct operation (change the value of "sw")
55     --only needs code to be added in simulation, not in the final file to be uploaded
56     --to the FPGA
57     --this is step 7 and 8 in Part 1
58
59     sw_proc: process
60     begin
61         sw <= x"7b10";
62         wait for clk_per;
63     end process sw_proc;
64
65     --clock process, high and low for half the clock period
66     clk_proc: process

```

```
66     begin
67         wait for clk_per;
68         clk <= not (clk);
69
70         -- counter <= counter + 1;
71     end process clk_proc;
72
73 end Behavioral;
```

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 11/18/2017 01:22:39 PM
6  -- Design Name:
7  -- Module Name: top - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
25 -- Uncomment the following library declaration if using
26 -- arithmetic functions with Signed or Unsigned values
27 use IEEE.NUMERIC_STD.ALL;
28
29 --use IEEE.std_logic_arith.all;
30 --use IEEE.std_logic_unsigned.all;
31
32 -- Uncomment the following library declaration if instantiating
33 -- any Xilinx leaf cells in this code.
34 --library UNISIM;
35 --use UNISIM.VComponents.all;
36
37 entity top is
38     Port (
39         sw      : in std_logic_vector(7 downto 0);
40         btnR    : in std_logic;
41         btnL    : in std_logic;
42         clk     : in std_logic;
43
44         seg     : out std_logic_vector(6 downto 0);
45         dp      : out std_logic;
46         an      : out std_logic_vector(3 downto 0);
47         led     : out std_logic_vector(7 downto 0)
48     );
49 end top;
50
51 architecture Behavioral of top is
52
53     component encoder is
54         Port (
55             hex_in : in STD_LOGIC_VECTOR(3 DOWNTO 0);
56             a      : out STD_LOGIC;
57             b      : out STD_LOGIC;
58             c      : out STD_LOGIC;
59             d      : out STD_LOGIC;
60             e      : out STD_LOGIC;
61             f      : out STD_LOGIC;
62             g      : out STD_LOGIC
63         );
64     end component encoder;
65
66

```

```

67     component ssd_muxer is
68         Port (
69             a_in      : in  std_logic_vector(3 downto 0);
70             b_in      : in  std_logic_vector(3 downto 0);
71             c_in      : in  std_logic_vector(3 downto 0);
72             d_in      : in  std_logic_vector(3 downto 0);
73             e_in      : in  std_logic_vector(3 downto 0);
74             f_in      : in  std_logic_vector(3 downto 0);
75             g_in      : in  std_logic_vector(3 downto 0);
76             decp0_in  : in  std_logic;
77             decp1_in  : in  std_logic;
78             decp2_in  : in  std_logic;
79             decp3_in  : in  std_logic;
80             seg_out   : out std_logic_vector(6 downto 0);
81             dp_out    : out std_logic;
82             an_out    : out std_logic_vector(3 downto 0);
83             clk       : in  STD_LOGIC
84         );
85     end component ssd_muxer;
86     signal a_in, b_in, c_in, d_in, e_in, f_in, g_in : std_logic_vector(3 downto 0) :=
    "0000";
87
88     component debounce is
89         Port (
90             CLK_100M : in std_logic;
91             SW       : in std_logic;
92             sglPulse : out std_logic;
93             Sig      : out std_logic);
94     end component debounce;
95     signal sglPulse : std_logic_vector(2 downto 0);
96     signal Sig      : std_logic_vector(2 downto 0);
97
98     signal accum : std_logic_vector(7 downto 0);
99
100    --signal hundreds, tens, ones : unsigned(7 downto 0);
101
102    --signal bcd : std_logic_vector(11 downto 0) := "00";
103
104    begin
105
106        led <= sw;
107
108        DEBR: debounce port map (
109            CLK_100M => clk,
110            SW      => btnR,
111            sglPulse => sglPulse(0),
112            Sig      => Sig(0)
113        );
114
115        DEBL: debounce port map (
116            CLK_100M => clk,
117            SW      => btnL,
118            sglPulse => sglPulse(1),
119            Sig      => Sig(1)
120        );
121
122        -- set up all the connectors
123        -- A true master starts at 0 (who am I kidding?)
124        ENC0: encoder port map (
125            hex_in => accum(3 downto 0),
126            a      => a_in(0),
127            b      => b_in(0),
128            c      => c_in(0),
129            d      => d_in(0),
130            e      => e_in(0),
131            f      => f_in(0),

```

```

132         g => g_in(0)
133     );
134
135     ENC1: encoder port map (
136         hex_in => accum(7 downto 4),
137         a => a_in(1),
138         b => b_in(1),
139         c => c_in(1),
140         d => d_in(1),
141         e => e_in(1),
142         f => f_in(1),
143         g => g_in(1)
144     );
145
146     -- ENC2: encoder port map (
147     --     hex_in => bcd(3 downto 0),
148     --     a => a_in(0),
149     --     b => b_in(0),
150     --     c => c_in(0),
151     --     d => d_in(0),
152     --     e => e_in(0),
153     --     f => f_in(0),
154     --     g => g_in(0)
155     -- );
156
157     a_in(2) <= '0';
158     b_in(2) <= '0';
159     c_in(2) <= '0';
160     d_in(2) <= '0';
161     e_in(2) <= '0';
162     f_in(2) <= '0';
163     --g_in(2) <= '0';
164     a_in(3) <= '0';
165     b_in(3) <= '0';
166     c_in(3) <= '0';
167     d_in(3) <= '0';
168     e_in(3) <= '0';
169     f_in(3) <= '0';
170     g_in(3) <= '0';
171
172
173     MUX: ssd_muxer port map (
174         a_in => a_in,
175         b_in => b_in,
176         c_in => c_in,
177         d_in => d_in,
178         e_in => e_in,
179         f_in => f_in,
180         g_in => g_in,
181
182         decp0_in => '0',
183         decp1_in => '0',
184         decp2_in => '0',
185         decp3_in => '0',
186
187         seg_out => seg,
188         dp_out => dp,
189         an_out => an,
190
191         clk => clk
192     );
193
194     --bcd <= conv_std_logic_vector(hundreds, 4) & conv_std_logic_vector(tens, 4) &
conv_std_logic_vector(ones, 4);
195
196     LOGIC: process (accum, sw, Sig)

```

```
197     variable counter : unsigned(3 downto 0) := "0000";
198     variable temp     : std_logic_vector(7 downto 0) := "00000000";
199 begin
200     if (Sig(1) = '1') then -- if we hit the accumulate button (button left)
201         if (sw(7) = '1') then -- we got a negative value from out switches
202             for counter in 0 to 7 loop
203                 if (sw(counter) = '0') then
204                     temp(counter) := '1';
205                 else
206                     temp(counter) := '0';
207                 end if;
208             end loop;
209             accum <= std_logic_vector(unsigned(temp) + 1); -- convert to insiged so
                we can use + overload
210             g_in(2) <= '1';
211         else -- not negative
212             accum <= sw;
213             g_in(2) <= '0';
214         end if;
215     else
216         accum <= accum; -- store old value
217     end if;
218
219     if (Sig(0) = '1') then -- if Sig 2 is on then we want to reset the accum
220         accum <= "00000000";
221         g_in(2) <= '0';
222     end if;
223 end process LOGIC;
224
225
226 end Behavioral;
227
```

```

1  -----
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date: 11/30/2017 07:17:01 PM
6  -- Design Name:
7  -- Module Name: top - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool Versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 -----
20
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24
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     Port (
36         sw      : in std_logic_vector(7 downto 0);
37         btnR    : in std_logic;
38         btnL    : in std_logic;
39         clk     : in std_logic;
40
41         led     : out std_logic_vector(15 downto 0)
42     );
43 end top;
44
45 architecture Behavioral of top is
46     component debounce is
47         Port (
48             CLK_100M : in std_logic;
49             SW       : in std_logic;
50             sglPulse  : out std_logic;
51             Sig       : out std_logic
52         );
53     end component;
54     signal reset, test : std_logic := '0';
55
56     signal value : std_logic_vector(7 downto 0);
57
58 begin
59
60     DEBL: debounce port map (
61         CLK_100M => clk,
62         SW       => btnL,
63         sglPulse => open,
64         Sig      => test
65     );
66

```

```
67     DEBR: debounce port map (
68         CLK_100M => clk,
69         SW => btnR,
70         sglPulse => open,
71         Sig => reset
72     );
73
74     -- test <= btnL;
75     -- reset <= btnR;
76
77     --led(15) <= '0';
78     led(14) <= '0';
79     led(13) <= '0';
80     led(12) <= '0';
81     led(11) <= '0';
82     led(10) <= '0';
83     led(9) <= '0';
84     led(8) <= '0';
85     led(7) <= sw(7);
86     led(6) <= sw(6);
87     led(5) <= sw(5);
88     led(4) <= sw(4);
89     led(3) <= sw(3);
90     led(2) <= sw(2);
91     led(1) <= sw(1);
92     led(0) <= sw(0);
93
94     GUESS: process (test)
95         variable led_15 : std_logic := '0';
96     begin
97         if (reset = '1') then
98             value <= sw;
99             led_15 := '0';
100        end if;
101
102        if (test = '1') then
103            if (value = sw) then
104                led_15 := '1';
105            else
106                led_15 := '0';
107            end if;
108
109            value <= value;
110        end if;
111
112        led(15) <= led_15;
113    end process GUESS;
114
115 end Behavioral;
```