Weifan Lin
Csc343
02/27/2015

16-Bit Adder Subtractor Sign Extension

Objectives:

The goal of this lab was to be able to apply 16-Bit Adder Subtractor we did to DE2 Board, and to understand how to use sign extension when there is not enough switch inputs on DE2 Board.

Specifications:

16-bit adder inputs:

A[15…0]: the first operand of the adder which is made up of four bits: A[15], A[14], A[13], … and A[0].

B[15…0]: the second operand of the adder which is made up of four bits: B[15], B[14], B[13], … and B[0].

Cin: the bit that is carried in from the less significant stage.

16-bit adder outputs:

Carry: the carry output is the bit that is carried over from an addition.

S[15…0]: The sum of the two operands which is made up of the four bits S[15], S[14], S[13], and S[0].

Overflow: to determine whether or not there is an overflow.

Sign Extension inputs:

a[8..0]: the first operand of the adder, values assigned from DE2 Board switches.

b[8..0]: the second operand of the adder, values assigned from DE2 Board switches.

Sign Extension outputs:

a_out[15..0]: least 8 signification bits are the same as input a and the most 8 signification bits are the same as the 8th bit.

b_out[15..0]: least 8 signification bits are the same as input b and the most 8 signification bits are the same as the 8th bit.

Seven Segment Display:

2 seven segment displays to show HEX value of the first operand, and 2 seven segment displays to show HEX value of the second operand, and 4 seven segment displays to show the result of operation.

Functionality:

A 16-bit adder is made up of 8 2-bit full adders and it adds 2 16-bit numbers, results in a 16-bit sum and when the result required a 17th bit to carry the output, Cout yells 1, however, carry output is separated from the final sum. This adder has an additional output to determine if there was an overflow.

However, we wanted to run our program to DE2 Board and there were limited switches for inputs. So we will used sign extension. On DE2 Board we assigned 8 bits to the first operand and copy the 8th bit value to the most signification bits, and do the same thing for second operand.

Design:

Full Adder

```
1   -- 1-bit Adder
2
3   LIBRARY IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5
6   entity BIT_ADDER is
7           port( a, b, cin          : in  STD_LOGIC;
8                 sum, cout          : out STD_LOGIC );
9   end BIT_ADDER;
10
11  architecture BHV of BIT_ADDER is
12  begin
13
14          sum <=  (not a and not b and cin) or
15                          (not a and b and not cin) or
16                          (a and not b and not cin) or
17                          (a and b and cin);
18
19          cout <= (not a and b and cin) or
20                          (a and not b and cin) or
21                          (a and b and not cin) or
22                          (a and b and cin);
23  end BHV;
```

2-bit Adder

```
1   LIBRARY IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity add16 is
5     port( a, b  : in STD_LOGIC_VECTOR(15 downto 0);
6           cin   : in STD_LOGIC;
7           ans   : out STD_LOGIC_VECTOR(15 downto 0);
8           overflow  : out STD_LOGIC;
9           cout  : out STD_LOGIC  );
10  end add16;
11
12  architecture arch of add16 is
13
14  component add2
15    port( a, b  : in STD_LOGIC_VECTOR(1 downto 0);
16          cin   : in STD_LOGIC;
17          ans   : out STD_LOGIC_VECTOR(1 downto 0);
18          cout  : out STD_LOGIC  );
19  end component;
20
21  signal c1,c2,c3,c4,c5,c6,c7,c8 : STD_LOGIC;
22  signal mux : STD_LOGIC_VECTOR(15 downto 0);
23  begin
24
25    mux(0) <= cin xor b(0);
26    mux(1) <= cin xor b(1);
27    mux(2) <= cin xor b(2);
28    mux(3) <= cin xor b(3);
29    mux(4) <= cin xor b(4);
30    mux(5) <= cin xor b(5);
31    mux(6) <= cin xor b(6);
32    mux(7) <= cin xor b(7);
```

16-bit Adder

```
1   LIBRARY IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity add2 is
5     port( a, b  : in STD_LOGIC_VECTOR(1 downto 0);
6           cin   : in STD_LOGIC;
7           ans   : out STD_LOGIC_VECTOR(1 downto 0);
8           cout  : out STD_LOGIC  );
9   end add2;
10
11  architecture STRUCTURE of add2 is
12  component BIT_ADDER
13    port( a, b, cin  : in  STD_LOGIC;
14          sum, cout  : out STD_LOGIC );
15  end component;
16
17  signal c1 : STD_LOGIC;
18  begin
19  b_adder0: BIT_ADDER port map (a(0), b(0), cin, ans(0), c1);
20  b_adder1: BIT_ADDER port map (a(1), b(1), c1, ans(1), cout);
21
22  END STRUCTURE;
```

```
33    mux(8) <= cin xor b(8);
34    mux(9) <= cin xor b(9);
35    mux(10) <= cin xor b(10);
36    mux(11) <= cin xor b(11);
37    mux(12) <= cin xor b(12);
38    mux(13) <= cin xor b(13);
39    mux(14) <= cin xor b(14);
40    mux(15) <= cin xor b(15);
41    b_adder0: add2 port map (a(1 downto 0), mux(1 downto 0), cin, ans(1 downto 0), c1);
42    b_adder1: add2 port map (a(3 downto 2), mux(3 downto 2), c1, ans(3 downto 2), c2);
43    b_adder2: add2 port map (a(5 downto 4), mux(5 downto 4), c2, ans(5 downto 4), c3);
44    b_adder3: add2 port map (a(7 downto 6), mux(7 downto 6), c3, ans(7 downto 6), c4);
45    b_adder4: add2 port map (a(9 downto 8), mux(9 downto 8), c4, ans(9 downto 8), c5);
46    b_adder5: add2 port map (a(11 downto 10), mux(11 downto 10), c5, ans(11 downto 10), c6);
47    b_adder6: add2 port map (a(13 downto 12), mux(13 downto 12), c6, ans(13 downto 12), c7);
48    b_adder7: add2 port map (a(15 downto 14), mux(15 downto 14), c7, ans(15 downto 14), c8);
49
50    cout <= c8;
51    overflow <= c8 xor c7;
52
53  END arch;
```

Sign Extension

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity sign_extention is
7    port ( a : in std_logic_vector(7 downto 0);
8           b : in std_logic_vector(7 downto 0);
9           a_out : out std_logic_vector(15 downto 0);
10          b_out : out std_logic_vector(15 downto 0));
11
12 end sign_extention;
13
14 architecture behavioral of sign_extention is
15 begin
16   process (a) is
17 begin
18   a_out <= a(7) & a(7) & a(7) & a(7) & a(7) & a(7) & a(7) & a(7) & a;
19 end process;
20
21 process (b) is
22     begin
23     b_out <= b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b;
24 end process;
25 end behavioral;
```

Seven Segment

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity Dec_7seg is
  port( sum16 : in std_logic_vector(15 downto 0);
        inputa: in std_logic_vector(7 downto 0);
        inputb: in std_logic_vector(7 downto 0);
        seg_a : out std_logic_vector(7 downto 0);
        seg_b : out std_logic_vector(7 downto 0);
        seg_c : out std_logic_vector(7 downto 0);
        seg_d : out std_logic_vector(7 downto 0);
        seg_e : out std_logic_vector(7 downto 0);
        seg_f : out std_logic_vector(7 downto 0);
        seg_g : out std_logic_vector(7 downto 0));

end Dec_7seg;

architecture struct of Dec_7seg is
  signal seg_data1 : std_logic_vector(6 downto 0);
  signal seg_data2 : std_logic_vector(6 downto 0);
  signal seg_data3 : std_logic_vector(6 downto 0);
  signal seg_data4 : std_logic_vector(6 downto 0);
  signal seg_data5 : std_logic_vector(6 downto 0);
  signal seg_data6 : std_logic_vector(6 downto 0);
  signal seg_data7 : std_logic_vector(6 downto 0);
  signal seg_data8 : std_logic_vector(6 downto 0);

begin
  process(sum16(3 downto 0)) is
    begin
  case sum16(3 downto 0) is

  when "0000"=>seg_data1<="1111110";
  when "0001"=>seg_data1<="0110000";
  when "0010"=>seg_data1<="1101101";
  when "0011"=>seg_data1<="1111001";
  when "0100"=>seg_data1<="0110011";
  when "0101"=>seg_data1<="1011011";
  when "0110"=>seg_data1<="1011111";
  when "0111"=>seg_data1<="1110000";
```

```
    when "1000"=>seg_data1<="1111111";
    when "1001"=>seg_data1<="1111011";
    when "1010"=>seg_data1<="1110111";
    when "1011"=>seg_data1<="0011111";
    when "1100"=>seg_data1<="0001101";
    when "1101"=>seg_data1<="0111101";
    when "1110"=>seg_data1<="1001111";
    when "1111"=>seg_data1<="1000111";
    when others=>seg_data1<="0000001";
  end case;
  end process;

    process(sum16(7 downto 4)) is
      begin
    case sum16(7 downto 4) is

    when "0000"=>seg_data2<="1111110";
    when "0001"=>seg_data2<="0110000";
    when "0010"=>seg_data2<="1101101";
    when "0011"=>seg_data2<="1111001";
    when "0100"=>seg_data2<="0110011";
    when "0101"=>seg_data2<="1011011";
    when "0110"=>seg_data2<="1011111";
    when "0111"=>seg_data2<="1110000";
    when "1000"=>seg_data2<="1111111";
    when "1001"=>seg_data2<="1111011";
    when "1010"=>seg_data2<="1110111";
    when "1011"=>seg_data2<="0011111";
    when "1100"=>seg_data2<="0001101";
    when "1101"=>seg_data2<="0111101";
    when "1110"=>seg_data2<="1001111";
    when "1111"=>seg_data2<="1000111";
    when others=>seg_data2<="0000001";
  end case;
  end process;

    process(sum16(11 downto 8)) is
      begin
    case sum16(11 downto 8) is
```

```
 82
 83      when "0000"=>seg_data3<="1111110";
 84      when "0001"=>seg_data3<="0110000";
 85      when "0010"=>seg_data3<="1101101";
 86      when "0011"=>seg_data3<="1111001";
 87      when "0100"=>seg_data3<="0110011";
 88      when "0101"=>seg_data3<="1011011";
 89      when "0110"=>seg_data3<="1011111";
 90      when "0111"=>seg_data3<="1110000";
 91      when "1000"=>seg_data3<="1111111";
 92      when "1001"=>seg_data3<="1111011";
 93      when "1010"=>seg_data3<="1110111";
 94      when "1011"=>seg_data3<="0011111";
 95      when "1100"=>seg_data3<="0001101";
 96      when "1101"=>seg_data3<="0111101";
 97      when "1110"=>seg_data3<="1001111";
 98      when "1111"=>seg_data3<="1000111";
 99      when others=>seg_data3<="0000001";
100    end case;
101    end process;
102
103    process(sum16(15 downto 12)) is
104      begin
105    case sum16(15 downto 12) is
106
107      when "0000"=>seg_data4<="1111110";
108      when "0001"=>seg_data4<="0110000";
109      when "0010"=>seg_data4<="1101101";
110      when "0011"=>seg_data4<="1111001";
111      when "0100"=>seg_data4<="0110011";
112      when "0101"=>seg_data4<="1011011";
113      when "0110"=>seg_data4<="1011111";
114      when "0111"=>seg_data4<="1110000";
115      when "1000"=>seg_data4<="1111111";
116      when "1001"=>seg_data4<="1111011";
117      when "1010"=>seg_data4<="1110111";
118      when "1011"=>seg_data4<="0011111";
119      when "1100"=>seg_data4<="0001101";
120      when "1101"=>seg_data4<="0111101";
121      when "1110"=>seg_data4<="1001111";
122      when "1111"=>seg_data4<="1000111";
123      when others=>seg_data4<="0000001";
```

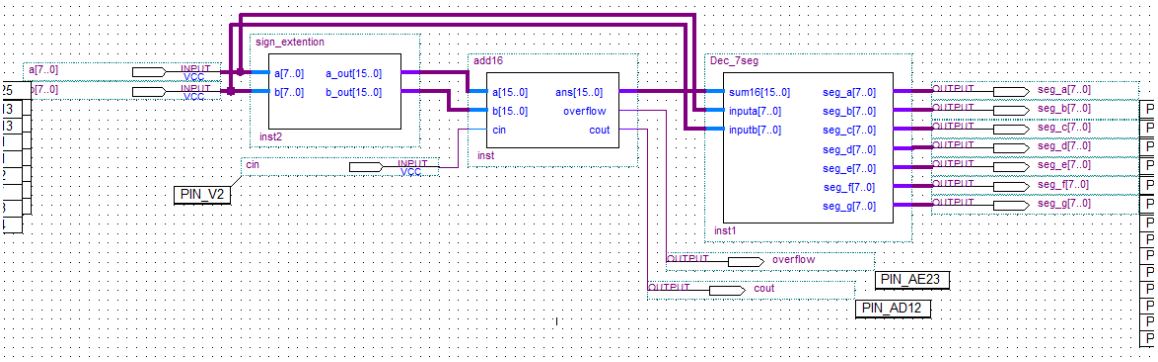
```
124   end case;
125   end process;
126
127
128     |
129     process(inputa(3 downto 0)) is
130       begin
131     case inputa(3 downto 0) is
132
133     when "0000"=>seg_data5<="1111110";
134     when "0001"=>seg_data5<="0110000";
135     when "0010"=>seg_data5<="1101101";
136     when "0011"=>seg_data5<="1111001";
137     when "0100"=>seg_data5<="0110011";
138     when "0101"=>seg_data5<="1011011";
139     when "0110"=>seg_data5<="1011111";
140     when "0111"=>seg_data5<="1110000";
141     when "1000"=>seg_data5<="1111111";
142     when "1001"=>seg_data5<="1111011";
143     when "1010"=>seg_data5<="1110111";
144     when "1011"=>seg_data5<="0011111";
145     when "1100"=>seg_data5<="0001101";
146     when "1101"=>seg_data5<="0111101";
147     when "1110"=>seg_data5<="1001111";
148     when "1111"=>seg_data5<="1000111";
149     when others=>seg_data5<="0000001";
150   end case;
151   end process;
152
153 process(inputa(7 downto 4)) is
154       begin
155     case inputa(7 downto 4) is
156
157     when "0000"=>seg_data6<="1111110";
158     when "0001"=>seg_data6<="0110000";
159     when "0010"=>seg_data6<="1101101";
160     when "0011"=>seg_data6<="1111001";
161     when "0100"=>seg_data6<="0110011";
162     when "0101"=>seg_data6<="1011011";
163     when "0110"=>seg_data6<="1011111";
164     when "0111"=>seg_data6<="1110000";
165     when "1000"=>seg_data6<="1111111";
```

```
166     when "1001"=>seg_data6<="1111011";
167     when "1010"=>seg_data6<="1110111";
168     when "1011"=>seg_data6<="0011111";
169     when "1100"=>seg_data6<="0001101";
170     when "1101"=>seg_data6<="0111101";
171     when "1110"=>seg_data6<="1001111";
172     when "1111"=>seg_data6<="1000111";
173     when others=>seg_data6<="0000001";
174   end case;
175   end process;
176
177
178 process(inputb(3 downto 0)) is
179       begin
180   case inputb(3 downto 0) is
181
182     when "0000"=>seg_data7<="1111110";
183     when "0001"=>seg_data7<="0110000";
184     when "0010"=>seg_data7<="1101101";
185     when "0011"=>seg_data7<="1111001";
186     when "0100"=>seg_data7<="0110011";
187     when "0101"=>seg_data7<="1011011";
188     when "0110"=>seg_data7<="1011111";
189     when "0111"=>seg_data7<="1110000";
190     when "1000"=>seg_data7<="1111111";
191     when "1001"=>seg_data7<="1111011";
192     when "1010"=>seg_data7<="1110111";
193     when "1011"=>seg_data7<="0011111";
194     when "1100"=>seg_data7<="0001101";
195     when "1101"=>seg_data7<="0111101";
196     when "1110"=>seg_data7<="1001111";
197     when "1111"=>seg_data7<="1000111";
198     when others=>seg_data7<="0000001";
199   end case;
200   end process;
201
202 process(inputb(7 downto 4)) is
203       begin
204   case inputb(7 downto 4) is
205
206     when "0000"=>seg_data8<="1111110";
207     when "0001"=>seg_data8<="0110000";
```

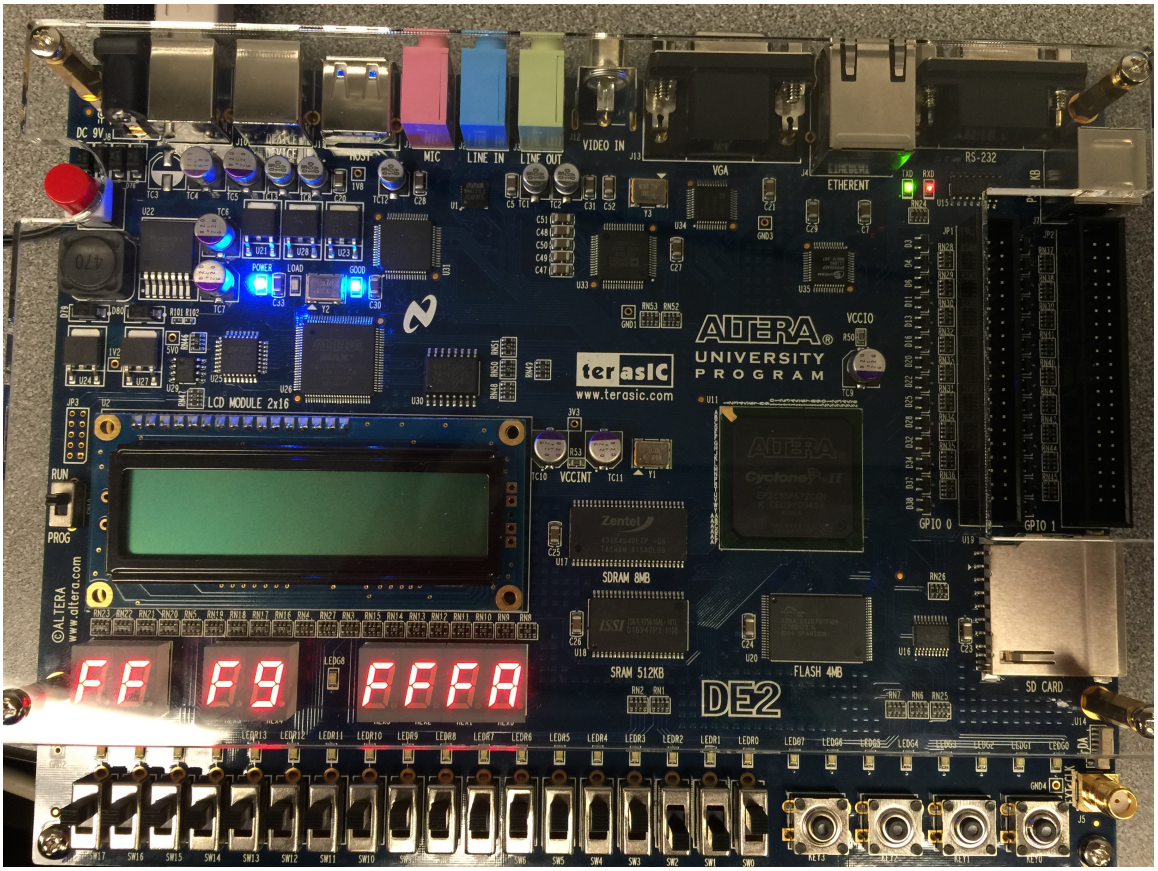
```
    when "0010"=>seg_data8<="1101101";
    when "0011"=>seg_data8<="1111001";
    when "0100"=>seg_data8<="0110011";
    when "0101"=>seg_data8<="1011011";
    when "0110"=>seg_data8<="1011111";
    when "0111"=>seg_data8<="1110000";
    when "1000"=>seg_data8<="1111111";
    when "1001"=>seg_data8<="1111011";
    when "1010"=>seg_data8<="1110111";
    when "1011"=>seg_data8<="0011111";
    when "1100"=>seg_data8<="0001101";
    when "1101"=>seg_data8<="0111101";
    when "1110"=>seg_data8<="1001111";
    when "1111"=>seg_data8<="1000111";
    when others=>seg_data8<="0000001";
end case;
end process;

seg_a(7) <= NOT seg_data8(6);
seg_b(7) <= NOT seg_data8(5);
seg_c(7) <= NOT seg_data8(4);
seg_d(7) <= NOT seg_data8(3);
seg_e(7) <= NOT seg_data8(2);
seg_f(7) <= NOT seg_data8(1);
seg_g(7) <= NOT seg_data8(0);

seg_a(6) <= NOT seg_data7(6);
seg_b(6) <= NOT seg_data7(5);
seg_c(6) <= NOT seg_data7(4);
seg_d(6) <= NOT seg_data7(3);
seg_e(6) <= NOT seg_data7(2);
seg_f(6) <= NOT seg_data7(1);
seg_g(6) <= NOT seg_data7(0);

seg_a(5) <= NOT seg_data6(6);
seg_b(5) <= NOT seg_data6(5);
seg_c(5) <= NOT seg_data6(4);
seg_d(5) <= NOT seg_data6(3);
seg_e(5) <= NOT seg_data6(2);
seg_f(5) <= NOT seg_data6(1);
seg_g(5) <= NOT seg_data6(0);
```

```
249
250   seg_a(4) <= NOT seg_data5(6);
251   seg_b(4) <= NOT seg_data5(5);
252   seg_c(4) <= NOT seg_data5(4);
253   seg_d(4) <= NOT seg_data5(3);
254   seg_e(4) <= NOT seg_data5(2);
255   seg_f(4) <= NOT seg_data5(1);
256   seg_g(4) <= NOT seg_data5(0);
257
258   seg_a(3) <= NOT seg_data4(6);
259   seg_b(3) <= NOT seg_data4(5);
260   seg_c(3) <= NOT seg_data4(4);
261   seg_d(3) <= NOT seg_data4(3);
262   seg_e(3) <= NOT seg_data4(2);
263   seg_f(3) <= NOT seg_data4(1);
264   seg_g(3) <= NOT seg_data4(0);
265   seg_a(2) <= NOT seg_data3(6);
266   seg_b(2) <= NOT seg_data3(5);
267   seg_c(2) <= NOT seg_data3(4);
268   seg_d(2) <= NOT seg_data3(3);
269   seg_e(2) <= NOT seg_data3(2);
270   seg_f(2) <= NOT seg_data3(1);
271   seg_g(2) <= NOT seg_data3(0);
272   seg_a(1) <= NOT seg_data2(6);
273   seg_b(1) <= NOT seg_data2(5);
274   seg_c(1) <= NOT seg_data2(4);
275   seg_d(1) <= NOT seg_data2(3);
276   seg_e(1) <= NOT seg_data2(2);
277   seg_f(1) <= NOT seg_data2(1);
278   seg_g(1) <= NOT seg_data2(0);
279   seg_a(0) <= NOT seg_data1(6);
280   seg_b(0) <= NOT seg_data1(5);
281   seg_c(0) <= NOT seg_data1(4);
282   seg_d(0) <= NOT seg_data1(3);
283   seg_e(0) <= NOT seg_data1(2);
284   seg_f(0) <= NOT seg_data1(1);
285   seg_g(0) <= NOT seg_data1(0);
286
287   END struct;
```

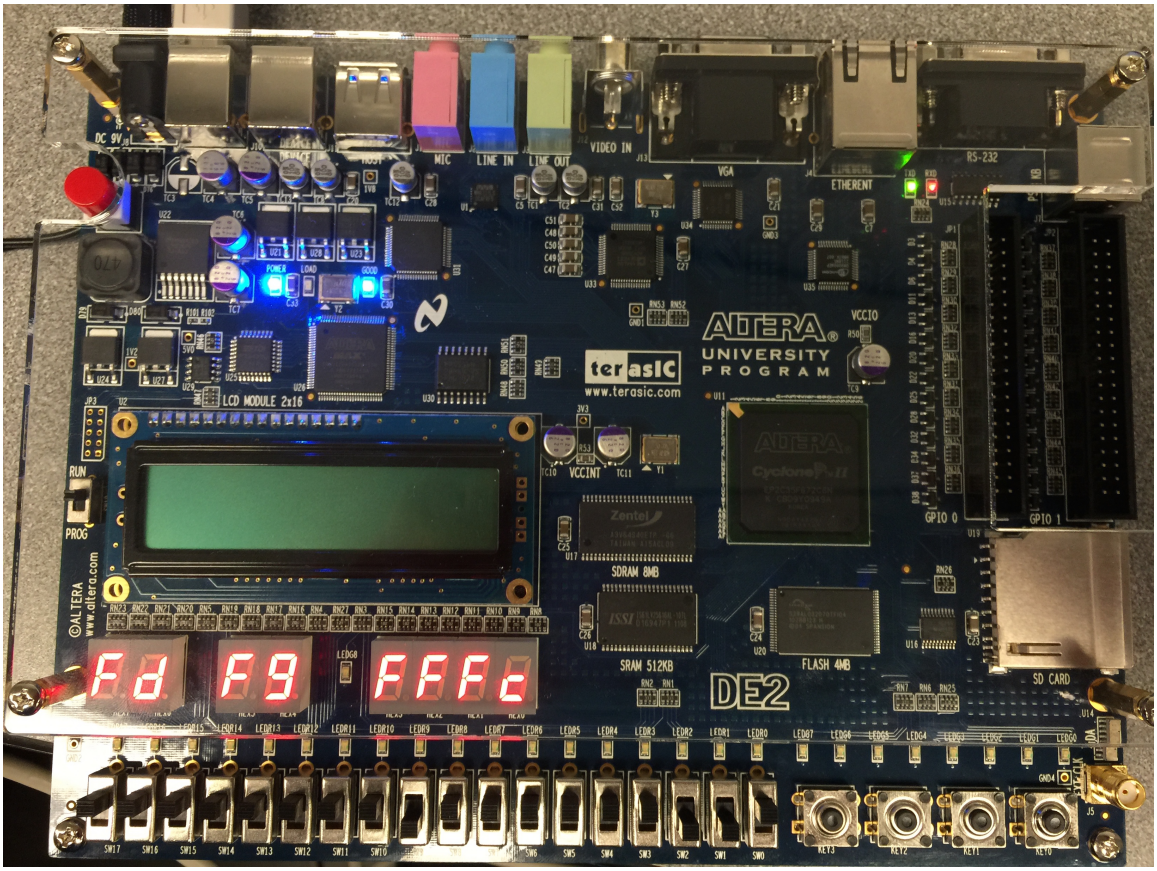
Sign Extension, 16-bit Adder Subtractor, Decoder to Seven Segment Display Combine Circuit



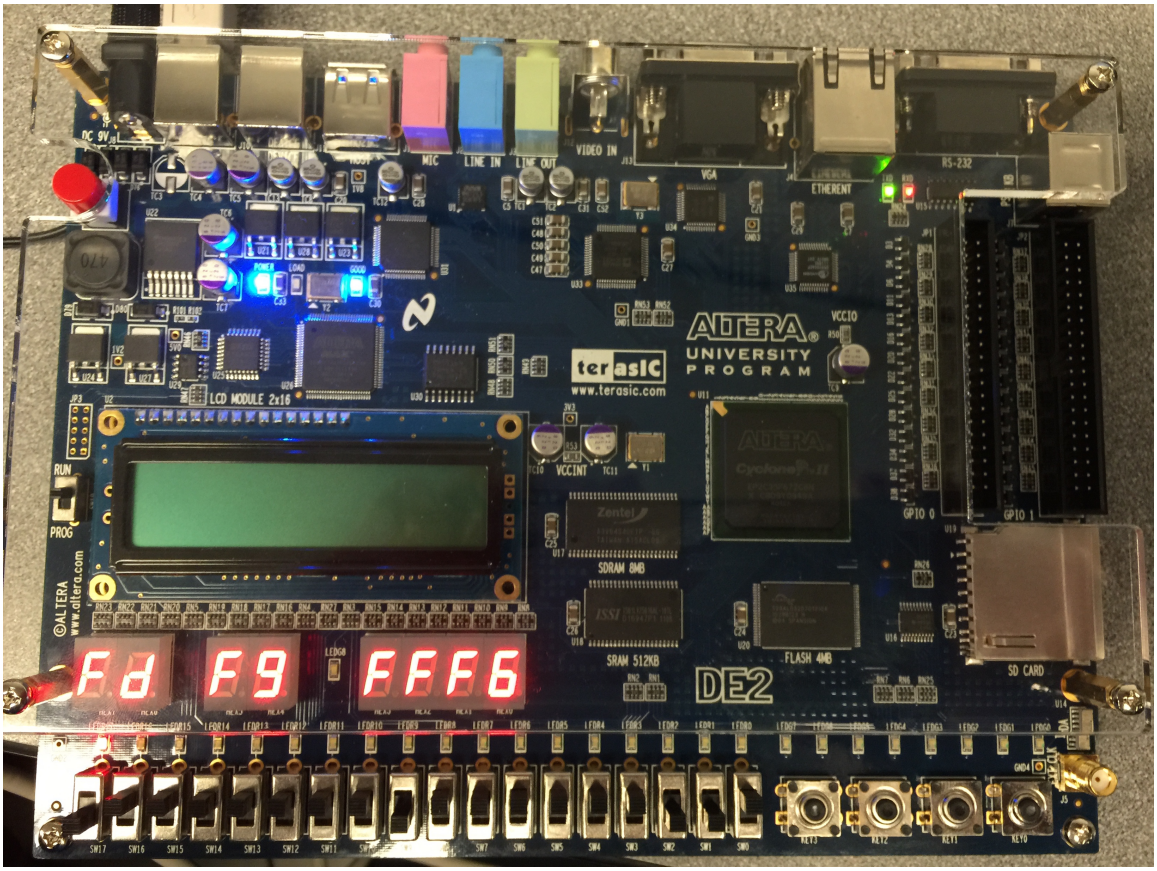
Simulation:



The first operand is F9 and the second operand is FF. F9-FF=FA => -7-(-1)=-6.



The first operand is F9 and the second operand is FD. F9-FD=FC => -7-(-3)=-4.



The first operand is F9 and the second operand is FD. F9+FD=FC => -7+(-3)=-10.

Conclusion:

The completion of this lab helped me practice a lot using Seven Segment Displays on Quartus. Building a 16-bit adder subtractor and use sign extension on DE2 board for inputs was not a very difficult task. First we had a function for sign extension to extend 8 bits input to 16 bits. And we do operation on 16 bit adder subtractor. Lastly, the correct result will be shown on seven segment displays.

Appendix:

Pin Assignments:

to, location

a[0], PIN_N25

a[1], PIN_N26

a[2], PIN_P25

a[3], PIN_AE14

a[4], PIN_AF14

a[5], PIN_AD13

a[6], PIN_AC13

a[7], PIN_C13

b[0], PIN_B13

b[1], PIN_A13

b[2], PIN_N1

b[3], PIN_P1

b[4], PIN_P2

b[5], PIN_T7

b[6], PIN_U3

b[7], PIN_U4

cin, PIN_V2

cout, PIN_AD12

overflow, PIN_AE23

seg_a[0], PIN_AF10

seg_b[0], PIN_AB12

seg_c[0], PIN_AC12

seg_d[0], PIN_AD11

seg_e[0], PIN_AE11

seg_f[0], PIN_V14

seg_g[0], PIN_V13

seg_a[1], PIN_V20

seg_b[1], PIN_V21

seg_c[1], PIN_W21

seg_d[1], PIN_Y22

seg_e[1], PIN_AA24

seg_f[1], PIN_AA23

seg_g[1], PIN_AB24

seg_a[2], PIN_AB23

seg_b[2], PIN_V22

seg_c[2], PIN_AC25

seg_d[2], PIN_AC26

seg_e[2], PIN_AB26

seg_f[2], PIN_AB25

seg_g[2], PIN_Y24

seg_a[3], PIN_Y23

seg_b[3], PIN_AA25

seg_c[3], PIN_AA26

seg_d[3], PIN_Y26

seg_e[3], PIN_Y25

seg_f[3], PIN_U22

seg_g[3], PIN_W24

seg_a[4], PIN_U9

seg_b[4], PIN_U1

seg_c[4], PIN_U2

seg_d[4], PIN_T4

seg_e[4], PIN_R7

seg_f[4], PIN_R6

seg_g[4], PIN_T3

seg_a[5], PIN_T2

seg_b[5], PIN_P6

seg_c[5], PIN_P7

seg_d[5], PIN_T9

seg_e[5], PIN_R5

seg_f[5], PIN_R4

seg_g[5], PIN_R3

seg_a[6], PIN_R2

seg_b[6], PIN_P4

seg_c[6], PIN_P3

seg_d[6], PIN_M2

seg_e[6], PIN_M3

seg_f[6], PIN_M5

seg_g[6], PIN_M4

seg_a[7], PIN_L3

seg_b[7], PIN_L2

seg_c[7], PIN_L9

seg_d[7], PIN_L6

seg_e[7], PIN_L7

seg_f[7], PIN_P9

seg_g[7], PIN_N9

```
-------------------------1-bit Full Adder-------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity BIT_ADDER is
        port( a, b, cin         : in  STD_LOGIC;
              sum, cout         : out STD_LOGIC );
end BIT_ADDER;

architecture BHV of BIT_ADDER is
begin
        sum <=  (not a and not b and cin) or
                        (not a and b and not cin) or
                        (a and not b and not cin) or
                        (a and b and cin);
        cout <= (not a and b and cin) or
                        (a and not b and cin) or
                        (a and b and not cin) or
                        (a and b and cin);
end BHV;
```

```
-------------------------2-bit Adder-------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity add2 is
        port( a, b              : in    STD_LOGIC_VECTOR(1 downto 0);
              cin   : in STD_LOGIC;
              ans           : out   STD_LOGIC_VECTOR(1 downto 0);
              cout              : out   STD_LOGIC           );
end add2;

architecture STRUCTURE of add2 is
component BIT_ADDER
        port( a, b, cin         : in  STD_LOGIC;
              sum, cout         : out STD_LOGIC );
end component;

signal c1 : STD_LOGIC;
begin

b_adder0: BIT_ADDER port map (a(0), b(0), cin, ans(0), c1);
b_adder1: BIT_ADDER port map (a(1), b(1), c1, ans(1), cout);
END STRUCTURE;


-------------------------16-bit Adder Sub-------------------------
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity add16 is
        port( a, b              : in    STD_LOGIC_VECTOR(15 downto 0);
              cin   : in STD_LOGIC;--_VECTOR(15 downto 0);
              ans           : out   STD_LOGIC_VECTOR(15 downto 0);
              overflow          : out   STD_LOGIC;
              cout              : out   STD_LOGIC           );
end add16;

architecture arch of add16 is

component add2
        port( a, b              : in    STD_LOGIC_VECTOR(1 downto 0);
              cin   : in STD_LOGIC;
              ans           : out   STD_LOGIC_VECTOR(1 downto 0);
              cout              : out   STD_LOGIC           );
end component;

signal c1,c2,c3,c4,c5,c6,c7,c8 : STD_LOGIC;
signal mux : STD_LOGIC_VECTOR(15 downto 0);
begin
```

```
  mux(0) <= cin xor b(0);
  mux(1) <= cin xor b(1);
  mux(2) <= cin xor b(2);
  mux(3) <= cin xor b(3);
  mux(4) <= cin xor b(4);
  mux(5) <= cin xor b(5);
  mux(6) <= cin xor b(6);
  mux(7) <= cin xor b(7);
  mux(8) <= cin xor b(8);
  mux(9) <= cin xor b(9);
  mux(10) <= cin xor b(10);
  mux(11) <= cin xor b(11);
  mux(12) <= cin xor b(12);
  mux(13) <= cin xor b(13);
  mux(14) <= cin xor b(14);
  mux(15) <= cin xor b(15);
  b_adder0: add2 port map (a(1 downto 0), mux(1 downto 0), cin, ans(1 downto 0), c1);
  b_adder1: add2 port map (a(3 downto 2), mux(3 downto 2), c1, ans(3 downto 2), c2);
  b_adder2: add2 port map (a(5 downto 4), mux(5 downto 4), c2, ans(5 downto 4), c3);
  b_adder3: add2 port map (a(7 downto 6), mux(7 downto 6), c3, ans(7 downto 6), c4);
  b_adder4: add2 port map (a(9 downto 8), mux(9 downto 8), c4, ans(9 downto 8), c5);
  b_adder5: add2 port map (a(11 downto 10), mux(11 downto 10), c5, ans(11 downto 10),
c6);
  b_adder6: add2 port map (a(13 downto 12), mux(13 downto 12), c6, ans(13 downto 12),
c7);
  b_adder7: add2 port map (a(15 downto 14), mux(15 downto 14), c7, ans(15 downto 14),
c8);

  cout <= c8;
  overflow <= c8 xor c7;

END arch;


-------------------------Sign Extension-------------------------
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity sign_extention is
  port ( a : in std_logic_vector(7 downto 0);
        b : in std_logic_vector(7 downto 0);
        a_out : out std_logic_vector(15 downto 0);
        b_out : out std_logic_vector(15 downto 0));

end sign_extention;

architecture behavioral of sign_extention is
begin
  process (a) is
begin
  a_out <= a(7) & a(7) & a(7) & a(7) & a(7) & a(7) & a(7) & a(7) & a;
end process;

process (b) is
          begin
          b_out <= b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b(7) & b;
end process;
end behavioral;
```

----------Seven Segment Decoder----------

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity Dec_7seg is
  port( sum16 : in std_logic_vector(15 downto 0);
        inputa: in std_logic_vector(7 downto 0);
        inputb: in std_logic_vector(7 downto 0);
        seg_a : out std_logic_vector(7 downto 0);
        seg_b : out std_logic_vector(7 downto 0);
        seg_c : out std_logic_vector(7 downto 0);
        seg_d : out std_logic_vector(7 downto 0);
        seg_e : out std_logic_vector(7 downto 0);
        seg_f : out std_logic_vector(7 downto 0);
        seg_g : out std_logic_vector(7 downto 0));

end Dec_7seg;

architecture struct of Dec_7seg is
  signal seg_data1 : std_logic_vector(6 downto 0);
  signal seg_data2 : std_logic_vector(6 downto 0);
  signal seg_data3 : std_logic_vector(6 downto 0);
  signal seg_data4 : std_logic_vector(6 downto 0);
  signal seg_data5 : std_logic_vector(6 downto 0);
```

```
  signal seg_data6 : std_logic_vector(6 downto 0);
  signal seg_data7 : std_logic_vector(6 downto 0);
  signal seg_data8 : std_logic_vector(6 downto 0);

begin
  process(sum16(3 downto 0)) is
    begin
  case sum16(3 downto 0) is

  when "0000"=>seg_data1<="1111110";
  when "0001"=>seg_data1<="0110000";
  when "0010"=>seg_data1<="1101101";
  when "0011"=>seg_data1<="1111001";
  when "0100"=>seg_data1<="0110011";
  when "0101"=>seg_data1<="1011011";
  when "0110"=>seg_data1<="1011111";
  when "0111"=>seg_data1<="1110000";
  when "1000"=>seg_data1<="1111111";
  when "1001"=>seg_data1<="1111011";
  when "1010"=>seg_data1<="1110111";
  when "1011"=>seg_data1<="0011111";
  when "1100"=>seg_data1<="0001101";
  when "1101"=>seg_data1<="0111101";
```

```
  when "1110"=>seg_data1<="1001111";
  when "1111"=>seg_data1<="1000111";
  when others=>seg_data1<="0000001";
end case;
end process;

  process(sum16(7 downto 4)) is
    begin
  case sum16(7 downto 4) is

  when "0000"=>seg_data2<="1111110";
  when "0001"=>seg_data2<="0110000";
  when "0010"=>seg_data2<="1101101";
  when "0011"=>seg_data2<="1111001";
  when "0100"=>seg_data2<="0110011";
  when "0101"=>seg_data2<="1011011";
  when "0110"=>seg_data2<="1011111";
  when "0111"=>seg_data2<="1110000";
  when "1000"=>seg_data2<="1111111";
  when "1001"=>seg_data2<="1111011";
  when "1010"=>seg_data2<="1110111";
  when "1011"=>seg_data2<="0011111";
  when "1100"=>seg_data2<="0001101";
```

```
  when "1101"=>seg_data2<="0111101";
  when "1110"=>seg_data2<="1001111";
  when "1111"=>seg_data2<="1000111";
  when others=>seg_data2<="0000001";
end case;
end process;

  process(sum16(11 downto 8)) is
    begin
  case sum16(11 downto 8) is

  when "0000"=>seg_data3<="1111110";
  when "0001"=>seg_data3<="0110000";
  when "0010"=>seg_data3<="1101101";
  when "0011"=>seg_data3<="1111001";
  when "0100"=>seg_data3<="0110011";
  when "0101"=>seg_data3<="1011011";
  when "0110"=>seg_data3<="1011111";
  when "0111"=>seg_data3<="1110000";
  when "1000"=>seg_data3<="1111111";
  when "1001"=>seg_data3<="1111011";
  when "1010"=>seg_data3<="1110111";
  when "1011"=>seg_data3<="0011111";
```

```
  when "1100"=>seg_data3<="0001101";
  when "1101"=>seg_data3<="0111101";
  when "1110"=>seg_data3<="1001111";
  when "1111"=>seg_data3<="1000111";
  when others=>seg_data3<="0000001";
end case;
end process;

  process(sum16(15 downto 12)) is
    begin
  case sum16(15 downto 12) is

  when "0000"=>seg_data4<="1111110";
  when "0001"=>seg_data4<="0110000";
  when "0010"=>seg_data4<="1101101";
  when "0011"=>seg_data4<="1111001";
  when "0100"=>seg_data4<="0110011";
  when "0101"=>seg_data4<="1011011";
  when "0110"=>seg_data4<="1011111";
  when "0111"=>seg_data4<="1110000";
  when "1000"=>seg_data4<="1111111";
  when "1001"=>seg_data4<="1111011";
  when "1010"=>seg_data4<="1110111";
```

```
  when "1011"=>seg_data4<="0011111";
  when "1100"=>seg_data4<="0001101";
  when "1101"=>seg_data4<="0111101";
  when "1110"=>seg_data4<="1001111";
  when "1111"=>seg_data4<="1000111";
  when others=>seg_data4<="0000001";
end case;
end process;



  process(inputa(3 downto 0)) is
    begin
  case inputa(3 downto 0) is

  when "0000"=>seg_data5<="1111110";
  when "0001"=>seg_data5<="0110000";
  when "0010"=>seg_data5<="1101101";
  when "0011"=>seg_data5<="1111001";
  when "0100"=>seg_data5<="0110011";
  when "0101"=>seg_data5<="1011011";
  when "0110"=>seg_data5<="1011111";
  when "0111"=>seg_data5<="1110000";
```

```
  when "1000"=>seg_data5<="1111111";
  when "1001"=>seg_data5<="1111011";
  when "1010"=>seg_data5<="1110111";
  when "1011"=>seg_data5<="0011111";
  when "1100"=>seg_data5<="0001101";
  when "1101"=>seg_data5<="0111101";
  when "1110"=>seg_data5<="1001111";
  when "1111"=>seg_data5<="1000111";
  when others=>seg_data5<="0000001";
end case;
end process;

process(inputa(7 downto 4)) is
    begin
  case inputa(7 downto 4) is

  when "0000"=>seg_data6<="1111110";
  when "0001"=>seg_data6<="0110000";
  when "0010"=>seg_data6<="1101101";
  when "0011"=>seg_data6<="1111001";
  when "0100"=>seg_data6<="0110011";
  when "0101"=>seg_data6<="1011011";
  when "0110"=>seg_data6<="1011111";
```

```
  when "0111"=>seg_data6<="1110000";
  when "1000"=>seg_data6<="1111111";
  when "1001"=>seg_data6<="1111011";
  when "1010"=>seg_data6<="1110111";
  when "1011"=>seg_data6<="0011111";
  when "1100"=>seg_data6<="0001101";
  when "1101"=>seg_data6<="0111101";
  when "1110"=>seg_data6<="1001111";
  when "1111"=>seg_data6<="1000111";
  when others=>seg_data6<="0000001";
end case;
end process;


process(inputb(3 downto 0)) is
   begin
  case inputb(3 downto 0) is

  when "0000"=>seg_data7<="1111110";
  when "0001"=>seg_data7<="0110000";
  when "0010"=>seg_data7<="1101101";
  when "0011"=>seg_data7<="1111001";
  when "0100"=>seg_data7<="0110011";
```

```
  when "0101"=>seg_data7<="1011011";
  when "0110"=>seg_data7<="1011111";
  when "0111"=>seg_data7<="1110000";
  when "1000"=>seg_data7<="1111111";
  when "1001"=>seg_data7<="1111011";
  when "1010"=>seg_data7<="1110111";
  when "1011"=>seg_data7<="0011111";
  when "1100"=>seg_data7<="0001101";
  when "1101"=>seg_data7<="0111101";
  when "1110"=>seg_data7<="1001111";
  when "1111"=>seg_data7<="1000111";
  when others=>seg_data7<="0000001";
end case;
end process;

process(inputb(7 downto 4)) is
   begin
  case inputb(7 downto 4) is

  when "0000"=>seg_data8<="1111110";
  when "0001"=>seg_data8<="0110000";
  when "0010"=>seg_data8<="1101101";
  when "0011"=>seg_data8<="1111001";
```

```
  when "0100"=>seg_data8<="0110011";
  when "0101"=>seg_data8<="1011011";
  when "0110"=>seg_data8<="1011111";
  when "0111"=>seg_data8<="1110000";
  when "1000"=>seg_data8<="1111111";
  when "1001"=>seg_data8<="1111011";
  when "1010"=>seg_data8<="1110111";
  when "1011"=>seg_data8<="0011111";
  when "1100"=>seg_data8<="0001101";
  when "1101"=>seg_data8<="0111101";
  when "1110"=>seg_data8<="1001111";
  when "1111"=>seg_data8<="1000111";
  when others=>seg_data8<="0000001";
end case;
end process;

seg_a(7) <= NOT seg_data8(6);
seg_b(7) <= NOT seg_data8(5);
seg_c(7) <= NOT seg_data8(4);
seg_d(7) <= NOT seg_data8(3);
seg_e(7) <= NOT seg_data8(2);
seg_f(7) <= NOT seg_data8(1);
seg_g(7) <= NOT seg_data8(0);
```

```
seg_a(6) <= NOT seg_data7(6);
seg_b(6) <= NOT seg_data7(5);
seg_c(6) <= NOT seg_data7(4);
seg_d(6) <= NOT seg_data7(3);
seg_e(6) <= NOT seg_data7(2);
seg_f(6) <= NOT seg_data7(1);
seg_g(6) <= NOT seg_data7(0);

seg_a(5) <= NOT seg_data6(6);
seg_b(5) <= NOT seg_data6(5);
seg_c(5) <= NOT seg_data6(4);
seg_d(5) <= NOT seg_data6(3);
seg_e(5) <= NOT seg_data6(2);
seg_f(5) <= NOT seg_data6(1);
seg_g(5) <= NOT seg_data6(0);

seg_a(4) <= NOT seg_data5(6);
seg_b(4) <= NOT seg_data5(5);
seg_c(4) <= NOT seg_data5(4);
seg_d(4) <= NOT seg_data5(3);
seg_e(4) <= NOT seg_data5(2);
seg_f(4) <= NOT seg_data5(1);
```

```
seg_g(4) <= NOT seg_data5(0);


seg_a(3) <= NOT seg_data4(6);
seg_b(3) <= NOT seg_data4(5);
seg_c(3) <= NOT seg_data4(4);
seg_d(3) <= NOT seg_data4(3);
seg_e(3) <= NOT seg_data4(2);
seg_f(3) <= NOT seg_data4(1);
seg_g(3) <= NOT seg_data4(0);
seg_a(2) <= NOT seg_data3(6);
seg_b(2) <= NOT seg_data3(5);
seg_c(2) <= NOT seg_data3(4);
seg_d(2) <= NOT seg_data3(3);
seg_e(2) <= NOT seg_data3(2);
seg_f(2) <= NOT seg_data3(1);
seg_g(2) <= NOT seg_data3(0);
seg_a(1) <= NOT seg_data2(6);
seg_b(1) <= NOT seg_data2(5);
seg_c(1) <= NOT seg_data2(4);
seg_d(1) <= NOT seg_data2(3);
seg_e(1) <= NOT seg_data2(2);
seg_f(1) <= NOT seg_data2(1);
seg_g(1) <= NOT seg_data2(0);
```

```
seg_a(0) <= NOT seg_data1(6);

seg_b(0) <= NOT seg_data1(5);

seg_c(0) <= NOT seg_data1(4);

seg_d(0) <= NOT seg_data1(3);

seg_e(0) <= NOT seg_data1(2);

seg_f(0) <= NOT seg_data1(1);

seg_g(0) <= NOT seg_data1(0);


END struct;
```