LAB REPORT
Laboratory exercise 2: Numbers and Displays

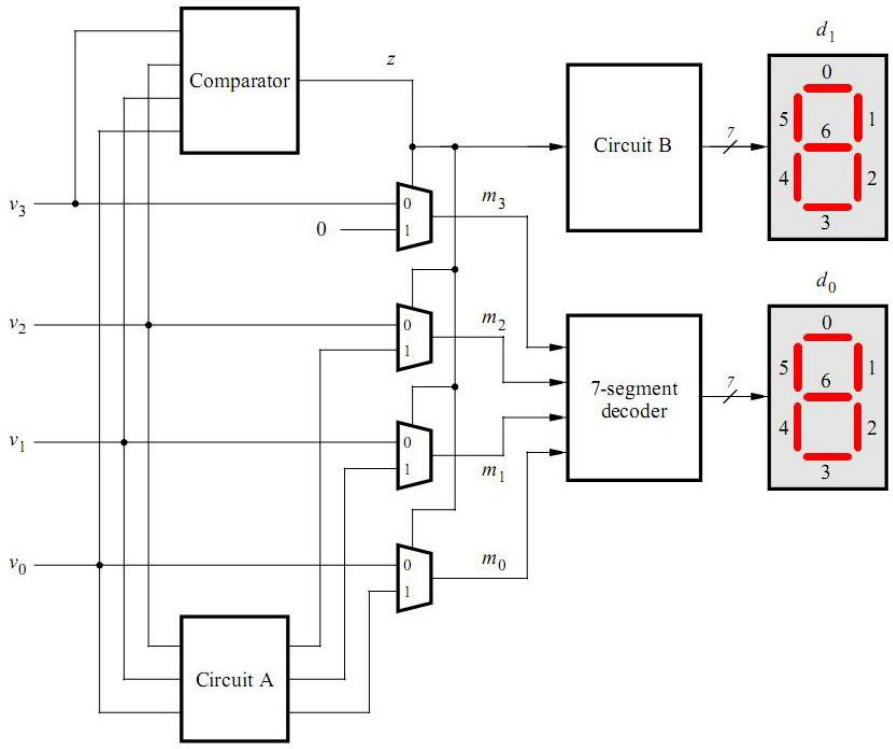
3220104688 杨佳昕

5 Dec, 2024

1. Problem Description

Part II

Convert a 4-bit binary input $v_3v_2v_1v_0$ to a 2-bit output d_1d_0. A partial design of this circuit is given. Use Boolean expressions and do not use if-else, case or similar statements.



Part V

BCD inputs A_1A_0 and B_1B_0 are controlled by SW15-12, 11-8, 7-4, 3-0 respectively. Display A_1A_0, B_1B_0 and the sum of the two numbers in HEX7-6, 5-4, and 2-0.

Part VI

An algorithm is given:

$T_0 = A_0 + B_0$;

if ($T_0 > 9$) then $Z_0 = 10$; $c_1 = 1$;

else $Z_0 = 0$; $c_1 = 0$; end if;

$S_0 = T_0 – Z_0;$

$T_1 = A_1 + B_1 + c_1;$

if $(T_1 > 9)$ *then* $Z_1 = 10; c_2 = 1;$

else $Z_1 = 0; c_2 = 0;$ *end if;*

$S_1 = T_1 – Z_1;$

$S_2 = c_2;$

The input and output is the same as that in Part V. Design the vhdl code.

Part VII

Design a combinational circuit that converts a 6-bit binary number into a 2-bit decimal number. Use SW5-0 as input and HEX1-0 as output.

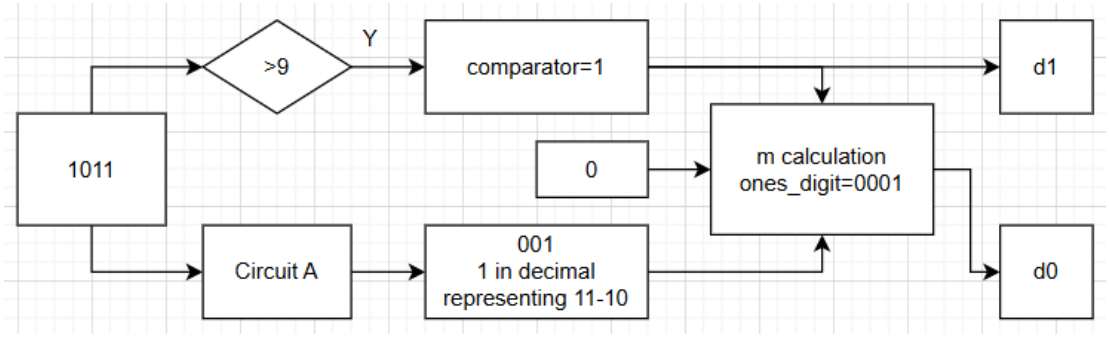
2. Design Formulation

Part II

When $v_3v_2v_1v_0$ is over 9 in binary, **comparator** = 1. We can use a Boolean expression like *sw(3) and (not sw(2) and sw(1) and not sw(0))* to evaluate **comparator** value. HEX1 is related to the variable **comparator**.

Circuit A is used when input is over 9 in binary. For example, when input is 1011, the ideal output after Circuit A is 001 (0001 in ones digit), and thus 010 (*not sw(2) and sw(1) and sw(0)*, representing 1011) must be in the list of circuit(0), and cannot be in the list of circuit(1), and so on.

Then, calculate m and use Boolean expressions similar as above to calculate HEX0(i).



Part V

Use *to_integer* to convert binary input into decimal form, add A_1A_0 and B_1B_0, and separate the hundreds, tens and ones digit. Display the digits in HEX2-0 with *case* sentences.

Part VI

Use the algorithm given to replace the *separation* part in Part V, and other parts are the same.

Part VII

In order to generate a combinational circuit with RTL viewer, no *to_integer* sentences should be included in the project, i.e., use Boolean expressions. We can classify all the 63 possible inputs with tens digit or ones digit with Boolean judgement. Once a number is input, we will classify it into its ten digit

category and one digit category, and display them in HEX1 and HEX0.

3. Design Entry

Part II

```
library ieee;
use ieee.std_logic_1164.all;

entity part2 is
    port(
        sw : in std_logic_vector(3 downto 0);
        hex1, hex0 : out std_logic_vector(6 downto 0)
    );
end part2;

architecture behavior of part2 is
    signal comparator : std_logic := '1';
    signal circuit : std_logic_vector(2 downto 0);
    signal m : std_logic_vector(3 downto 0) := "1111";

begin
    process(sw, comparator, circuit, m)
    begin
        -- hex1 --
        comparator <= sw(3) and ((not sw(2) and sw(1) and not sw(0))
                              or (not sw(2) and sw(1) and sw(0))
                              or (sw(2) and not sw(1) and not sw(0))
                              or (sw(2) and not sw(1) and sw(0))
                              or (sw(2) and sw(1) and not sw(0))
                              or (sw(2) and sw(1) and sw(0)));
        hex1(0) <= comparator;
        hex1(1) <= '0';
        hex1(2) <= '0';
        hex1(3) <= comparator;
        hex1(4) <= comparator;
        hex1(5) <= comparator;
        hex1(6) <= '1';

        -- circuit calculation --
        circuit(0) <= (not sw(2) and sw(1) and sw(0))
                   or (sw(2) and not sw(1) and sw(0))
                   or (sw(2) and sw(1) and sw(0));
        circuit(1) <= (sw(2) and not sw(1) and not sw(0))
                   or (sw(2) and not sw(1) and sw(0));
        circuit(2) <= (sw(2) and sw(1) and not sw(0))
                   or (sw(2) and sw(1) and sw(0));

        -- m calculation --
        m(0) <= ((not comparator) and sw(0)) or (comparator and circuit(0));
        m(1) <= ((not comparator) and sw(1)) or (comparator and circuit(1));
        m(2) <= ((not comparator) and sw(2)) or (comparator and circuit(2));
        m(3) <= ((not comparator) and sw(3)) or (comparator and '0');

        -- hex0 --
        -- hex0(0) = 1 when 0001, 0100
        hex0(0) <= (not m(3) and not m(2) and not m(1) and m(0))
                or (not m(3) and m(2) and not m(1) and not m(0));
        -- hex0(1) = 1 when 0101, 0110
        hex0(1) <= (not m(3) and m(2) and not m(1) and m(0))
                or (not m(3) and m(2) and m(1) and not m(0));
        -- hex0(2) = 1 when 0010
        hex0(2) <= not m(3) and not m(2) and m(1) and not m(0);
        -- hex0(3) = 1 when 0001, 0100, 0111
        hex0(3) <= (not m(3) and not m(2) and not m(1) and m(0))
                or (not m(3) and m(2) and not m(1) and not m(0))
                or (not m(3) and m(2) and m(1) and m(0));
        -- hex0(4) = 1 when 0001, 0011, 0100, 0101, 0111, 1001
        hex0(4) <= (not m(3) and not m(2) and not m(1) and m(0))
                or (not m(3) and not m(2) and m(1) and m(0))
                or (not m(3) and m(2) and not m(1) and not m(0))
                or (not m(3) and m(2) and not m(1) and m(0))
                or (not m(3) and m(2) and m(1) and m(0))
                or (m(3) and not m(2) and not m(1) and m(0));
        -- hex0(5) = 1 when 0001, 0010, 0011, 0111
        hex0(5) <= (not m(3) and not m(2) and not m(1) and m(0))
                or (not m(3) and not m(2) and m(1) and not m(0))
                or (not m(3) and not m(2) and m(1) and m(0))
                or (not m(3) and m(2) and m(1) and m(0));
```

```
        -- hex0(6) = 1 when 0000, 0001, 0111
        hex0(6) <= (not m(3) and not m(2) and not m(1) and not m(0))
                or (not m(3) and not m(2) and not m(1) and m(0))
                or (not m(3) and m(2) and m(1) and m(0));
    end process;
end behavior;
```

Part V

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity part5 is
    port(
        sw : in std_logic_vector(15 downto 0);
        hex7, hex6, hex5, hex4, hex2, hex1, hex0 : out std_logic_vector(6 downto 0)
    );
end part5;

architecture behavior of part5 is
    signal num_one : integer := 0;
    signal num_two : integer := 0;
    signal num_one_tens_digit : integer range 9 downto 0 := 0;
    signal num_one_ones_digit : integer range 9 downto 0 := 0;
    signal num_two_tens_digit : integer range 9 downto 0 := 0;
    signal num_two_ones_digit : integer range 9 downto 0 := 0;

    signal sum : integer := 0;
    signal sum_hundreds_digit : integer range 9 downto 0 := 0;
    signal sum_tens_digit : integer range 9 downto 0 := 0;
    signal sum_ones_digit : integer range 9 downto 0 := 0;

begin
    process(sw)
    begin
        num_one_tens_digit <= to_integer(unsigned(sw(15 downto 12)));
        num_one_ones_digit <= to_integer(unsigned(sw(11 downto 8)));
        num_two_tens_digit <= to_integer(unsigned(sw(7 downto 4)));
        num_two_ones_digit <= to_integer(unsigned(sw(3 downto 0)));
        num_one <= num_one_tens_digit * 10 + num_one_ones_digit;
        num_two <= num_two_tens_digit * 10 + num_two_ones_digit;
        sum <= num_one + num_two;

        -- num_one display --
        case num_one_tens_digit is
            when 0 => hex7 <= "1111111";
            when 1 => hex7 <= "1111001";
            when 2 => hex7 <= "0100100";
            when 3 => hex7 <= "0110000";
            when 4 => hex7 <= "0011001";
            when 5 => hex7 <= "0010010";
            when 6 => hex7 <= "0000010";
            when 7 => hex7 <= "1111000";
            when 8 => hex7 <= "0000000";
            when 9 => hex7 <= "0010000";
        end case;

        case num_one_ones_digit is
            when 0 => hex6 <= "1000000";
            when 1 => hex6 <= "1111001";
            when 2 => hex6 <= "0100100";
            when 3 => hex6 <= "0110000";
            when 4 => hex6 <= "0011001";
            when 5 => hex6 <= "0010010";
            when 6 => hex6 <= "0000010";
            when 7 => hex6 <= "1111000";
            when 8 => hex6 <= "0000000";
            when 9 => hex6 <= "0010000";
        end case;

        -- num_two display --
        case num_two_tens_digit is

        case num_two_ones_digit is

```

```
90          -- sum display --
91          sum_hundreds_digit <= (sum - sum mod 100) / 100;
92          sum_tens_digit <= ((sum - 100 * sum_hundreds_digit) - (sum - 100 * sum_hundreds_digit) mod 10) / 10;
93          sum_ones_digit <= sum mod 10;
94
95          case sum_hundreds_digit is
107
108         case sum_tens_digit is
120
121         case sum_ones_digit is
133     end process;
134 end behavior;
```

NB: Codes with similar structures are omitted.

Part VI

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity part6 is
6       port(
7           sw : in std_logic_vector(15 downto 0);
8           hex7, hex6, hex5, hex4, hex2, hex1, hex0 : out std_logic_vector(6 downto 0)
9       );
10  end part6;
11
12  architecture behavior of part6 is
13      signal a_1, a_0, b_1, b_0 : integer range 9 downto 0 := 0;
14      signal c_2, c_1, c_0, t_1, t_0, z_1, z_0 : integer := 0;
15      signal s_0, s_1, s_2 : integer range 9 downto 0 := 0;
16  begin
17      process(sw)
18      begin
19          a_1 <= to_integer(unsigned(sw(15 downto 12)));
20          a_0 <= to_integer(unsigned(sw(11 downto 8)));
21          b_1 <= to_integer(unsigned(sw(7 downto 4)));
22          b_0 <= to_integer(unsigned(sw(3 downto 0)));
23
24          -- algorithm given --
25          t_0 <= a_0 + b_0;
26          if t_0 > 9 then
27              z_0 <= 10;
28              c_1 <= 1;
29          else
30              z_0 <= 0;
31              c_1 <= 0;
32          end if;
33
34          s_0 <= t_0 - z_0;
35          t_1 <= a_1 + b_1 + c_1;
36
37          if t_1 > 9 then
38              z_1 <= 10;
39              c_2 <= 1;
40          else
41              z_1 <= 0;
42              c_2 <= 0;
43          end if;
44
45           s_1 <= t_1 - z_1;
46           s_2 <= c_2;
47
48           -- a display --
49           case a_1 is
50               when 0 => hex7 <= "1111111";
51               when 1 => hex7 <= "1111001";
52               when 2 => hex7 <= "0100100";
53               when 3 => hex7 <= "0110000";
54               when 4 => hex7 <= "0011001";
55               when 5 => hex7 <= "0010010";
56               when 6 => hex7 <= "0000010";
```

```
57              when 7 => hex7 <= "1111000";
58              when 8 => hex7 <= "0000000";
59              when 9 => hex7 <= "0010000";
60          end case;
61
62          case a_0 is
63              when 0 => hex6 <= "1000000";
64              when 1 => hex6 <= "1111001";
65              when 2 => hex6 <= "0100100";
66              when 3 => hex6 <= "0110000";
67              when 4 => hex6 <= "0011001";
68              when 5 => hex6 <= "0010010";
69              when 6 => hex6 <= "0000010";
70              when 7 => hex6 <= "1111000";
71              when 8 => hex6 <= "0000000";
72              when 9 => hex6 <= "0010000";
73          end case;
74
75          -- b display --
76
77          case b_1 is
89
90          case b_0 is
102
103         -- sum display --
104
105         case s_2 is
117
118         case s_1 is
119             when 0 => hex1 <= "1000000";
120             when 1 => hex1 <= "1111001";
121             when 2 => hex1 <= "0100100";
122             when 3 => hex1 <= "0110000";
123             when 4 => hex1 <= "0011001";
124             when 5 => hex1 <= "0010010";
125             when 6 => hex1 <= "0000010";
126             when 7 => hex1 <= "1111000";
127             when 8 => hex1 <= "0000000";
128             when 9 => hex1 <= "0010000";
129         end case;
130
131         case s_0 is
143     end process;
144 end behavior;
```

NB: Codes with similar structures are omitted.

Part VII

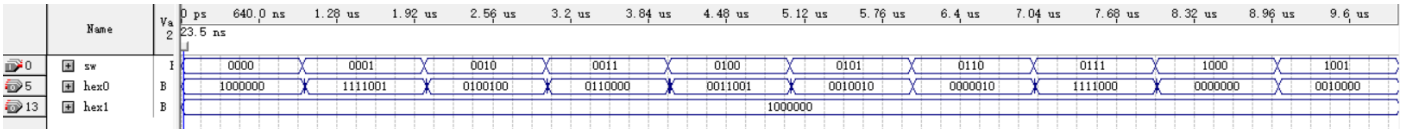
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity part7 is
6      port(
7          sw : in std_logic_vector(5 downto 0);
8          hex1, hex0 : out std_logic_vector(6 downto 0)
9      );
10 end part7;
11
12 architecture behavior of part7 is
13     signal hex_tens_digit : std_logic_vector(3 downto 0);
14     signal hex_ones_digit : std_logic_vector(3 downto 0);
```

```
12  architecture behavior of part7 is
13      signal hex_tens_digit : std_logic_vector(3 downto 0);
14      signal hex_ones_digit : std_logic_vector(3 downto 0);
15  begin
16      process(sw)
17      begin
18          if    sw >= "000000" and sw <= "001001" then hex_tens_digit <= "0000";
19          elsif sw >= "001010" and sw <= "010011" then hex_tens_digit <= "0001";
20          elsif sw >= "010100" and sw <= "011101" then hex_tens_digit <= "0010";
21          elsif sw >= "011110" and sw <= "100111" then hex_tens_digit <= "0011";
22          elsif sw >= "101000" and sw <= "110001" then hex_tens_digit <= "0100";
23          elsif sw >= "110010" and sw <= "111011" then hex_tens_digit <= "0101";
24          elsif sw >= "111100" and sw <= "111111" then hex_tens_digit <= "0110";
25          else hex_tens_digit <= "1111";
26          end if;
27
28          case sw is
29              when "000000" | "001010" | "010100" |
30                   "011110" | "101000" | "111100" => hex_ones_digit <= "0000";
31              when "000001" | "001011" | "010101" |
32                   "011111" | "101001" | "111101" => hex_ones_digit <= "0001";
33              when "000010" | "001100" | "010110" |
34                   "100000" | "101010" | "111110" => hex_ones_digit <= "0010";
35              when "000011" | "001101" | "010111" |
36                   "100001" | "101011" | "111111" => hex_ones_digit <= "0011";
37              when "000100" | "001110" | "011000" |
38                   "100010" | "101100"             => hex_ones_digit <= "0100";
39              when "000101" | "001111" | "011001" |
40                   "100011" | "101101"             => hex_ones_digit <= "0101";
41              when "000110" | "010000" | "011010" |
42                   "100100" | "101110"             => hex_ones_digit <= "0110";
43              when "000111" | "010001" | "011011" |
44                   "100101" | "101111"             => hex_ones_digit <= "0111";
45              when "001000" | "010010" | "011100" |
46                   "100110" | "110000"             => hex_ones_digit <= "1000";
47              when "001001" | "010011" | "011101" |
48                   "100111" | "110001"             => hex_ones_digit <= "1001";
49              when others                          => hex_ones_digit <= "1111";
50          end case;
51
```

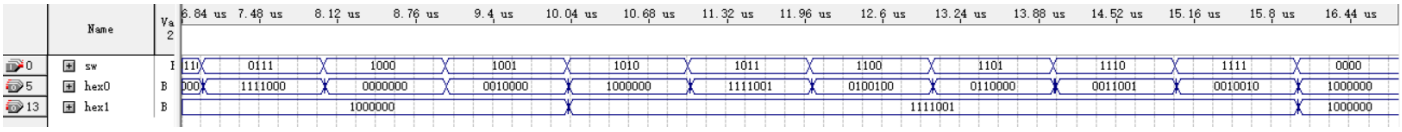
```
52          case hex_tens_digit is
53              when "0000" => hex1 <= "1111111";
54              when "0001" => hex1 <= "1111001";
55              when "0010" => hex1 <= "0100100";
56              when "0011" => hex1 <= "0110000";
57              when "0100" => hex1 <= "0011001";
58              when "0101" => hex1 <= "0010010";
59              when "0110" => hex1 <= "0000010";
60              when others => hex1 <= "1111111";
61          end case;
62
63          case hex_ones_digit is
76      end process;
77  end behavior;
```

4. Simulation and Synthesis Results

Part II

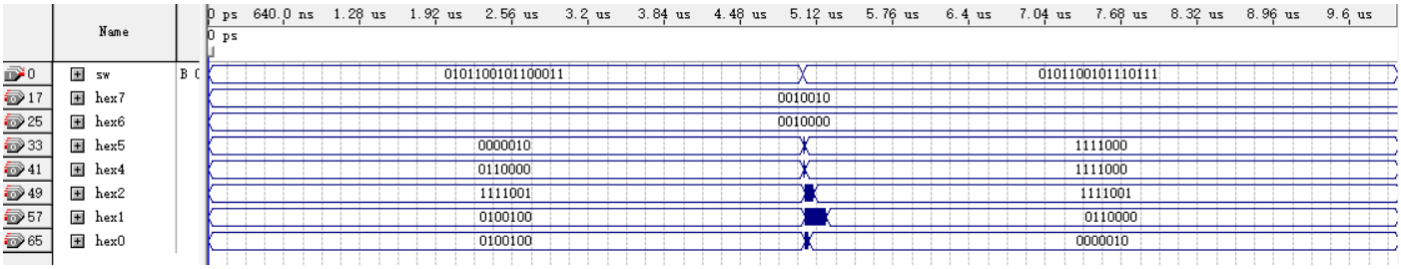


Case:

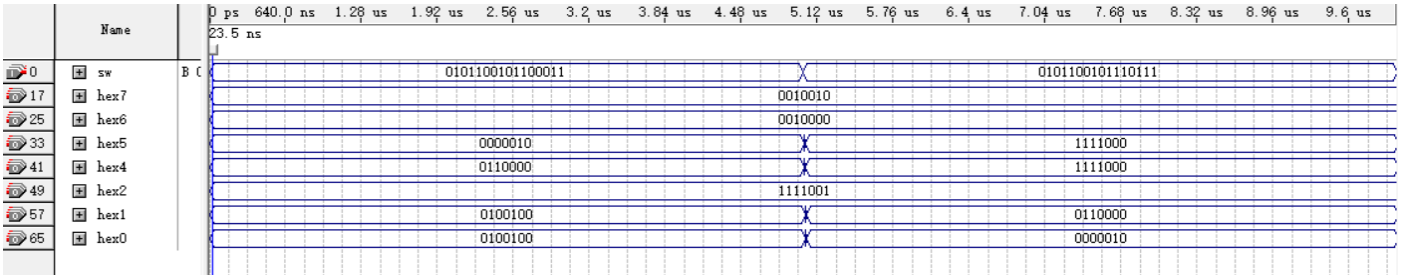


Part V

Take 59+63=122 and 59+77=136 for example:

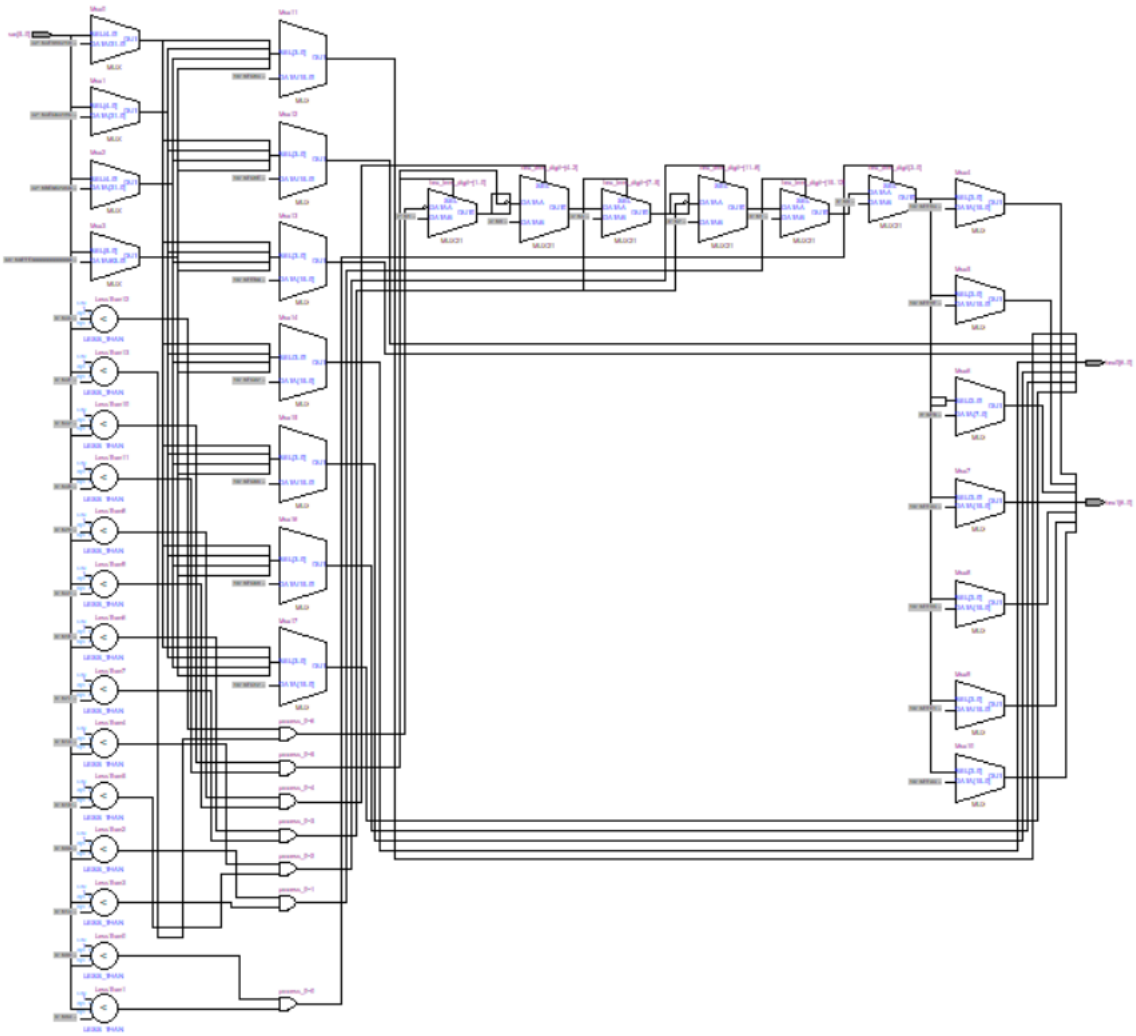


Part VI

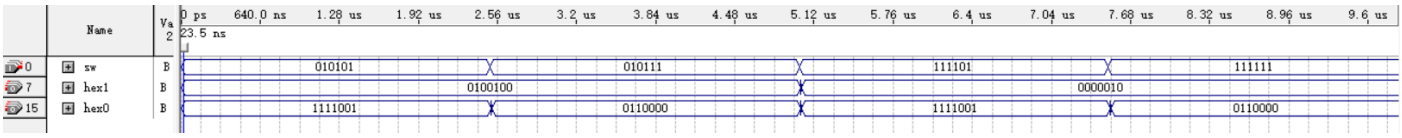


Part VII

RTL Viewer:

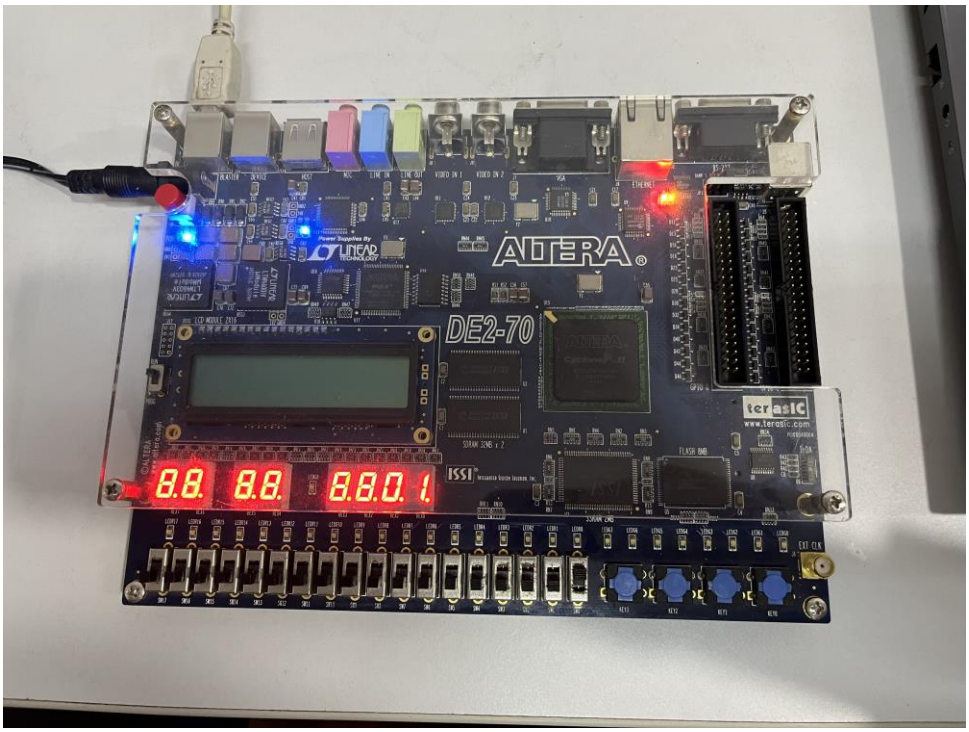


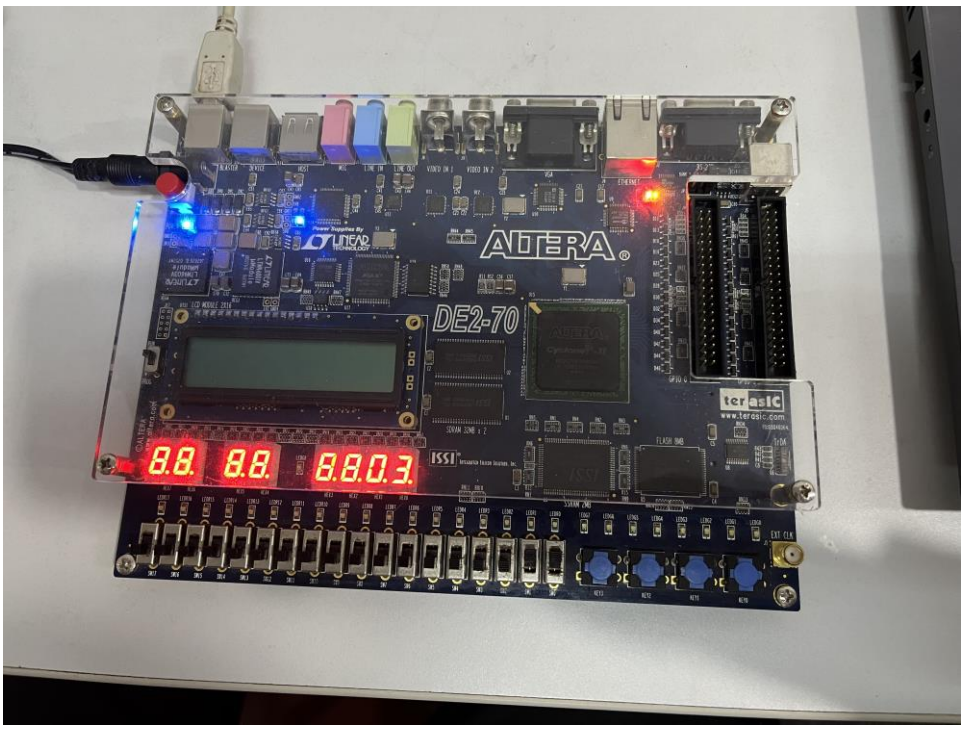
Take 21, 23, 61, 63 for example:

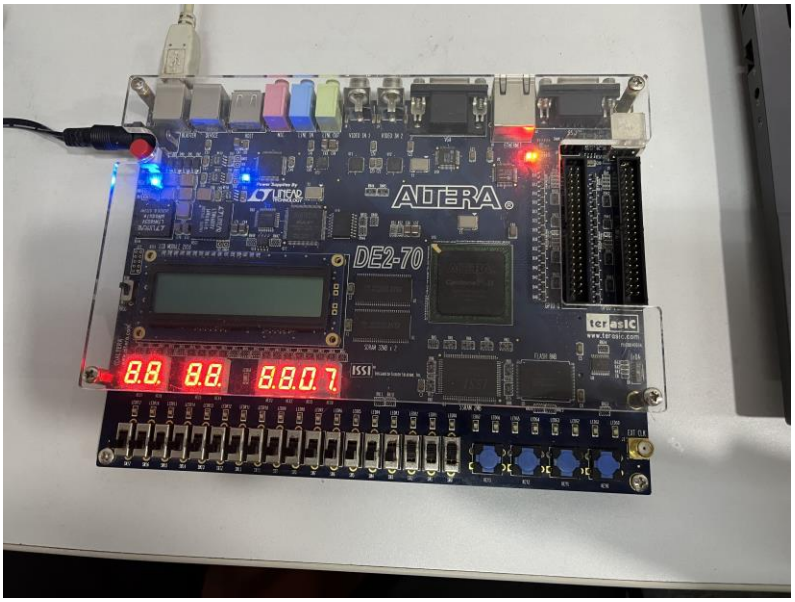


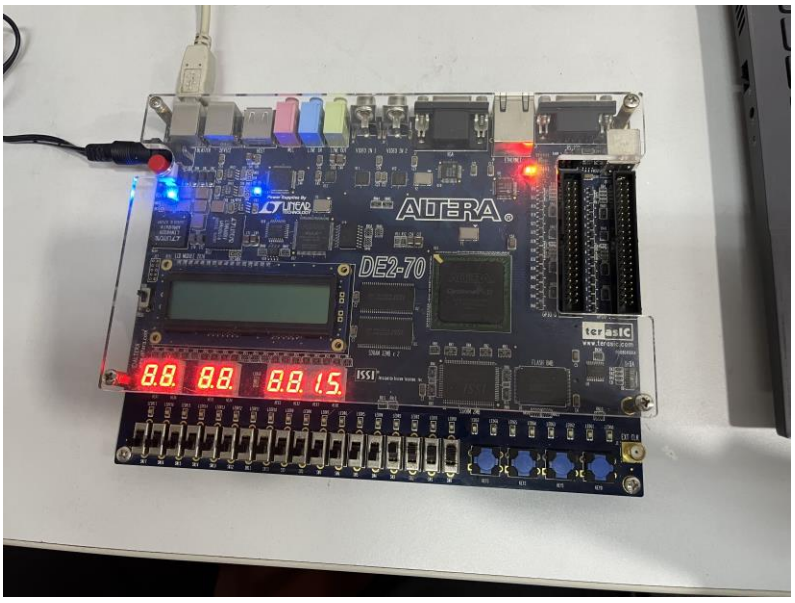
5. Experimental Results:

Part II

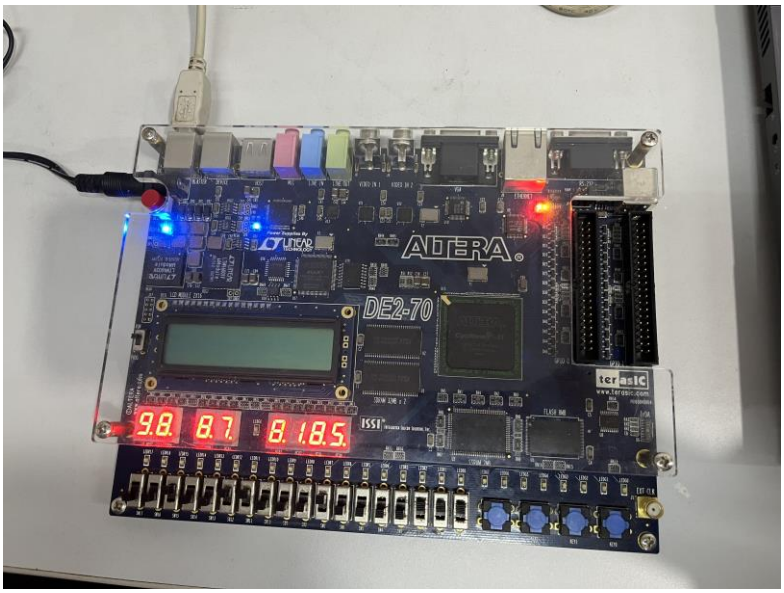


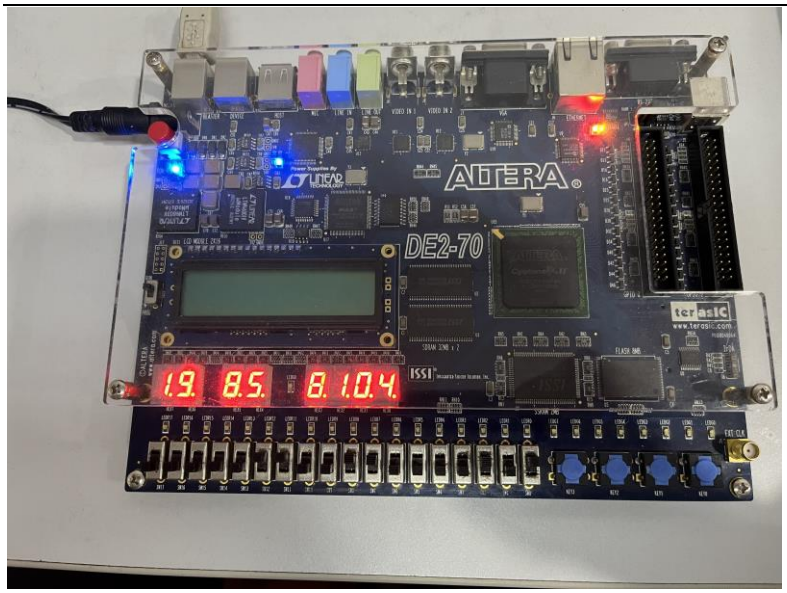




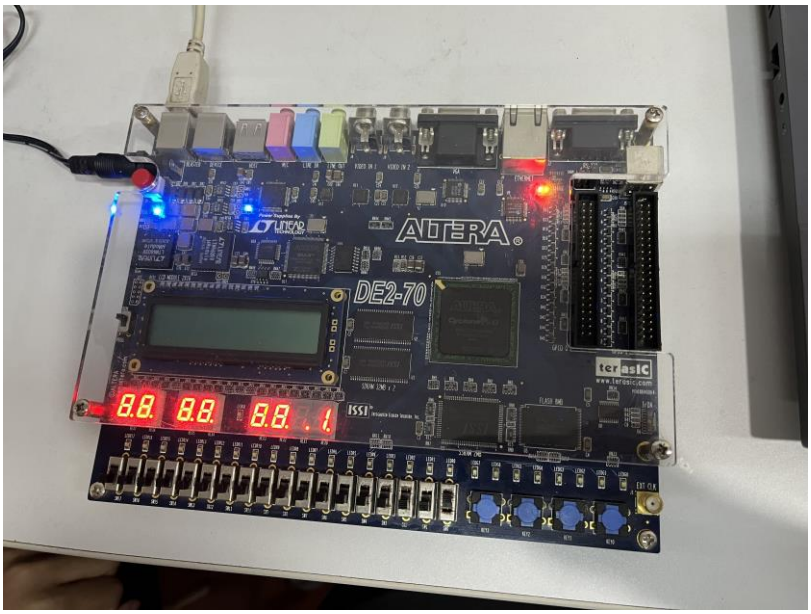


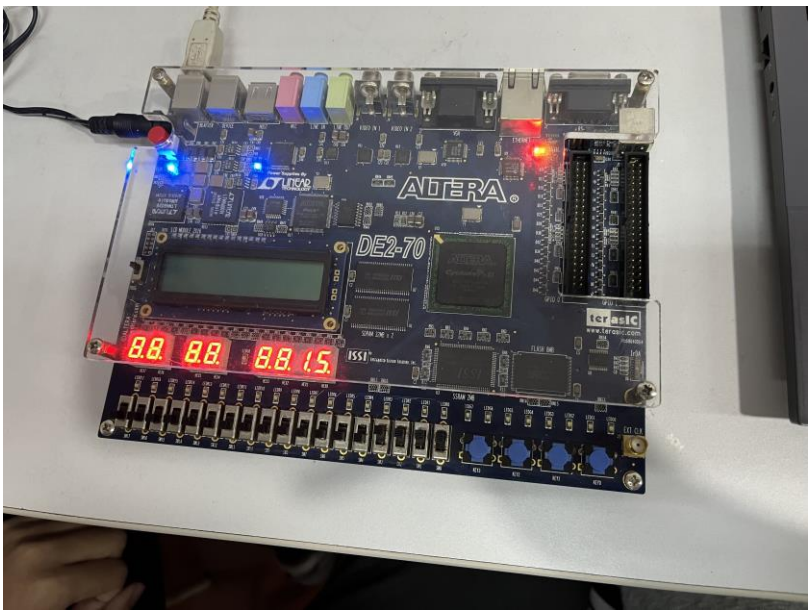
Part V and **Part VI**

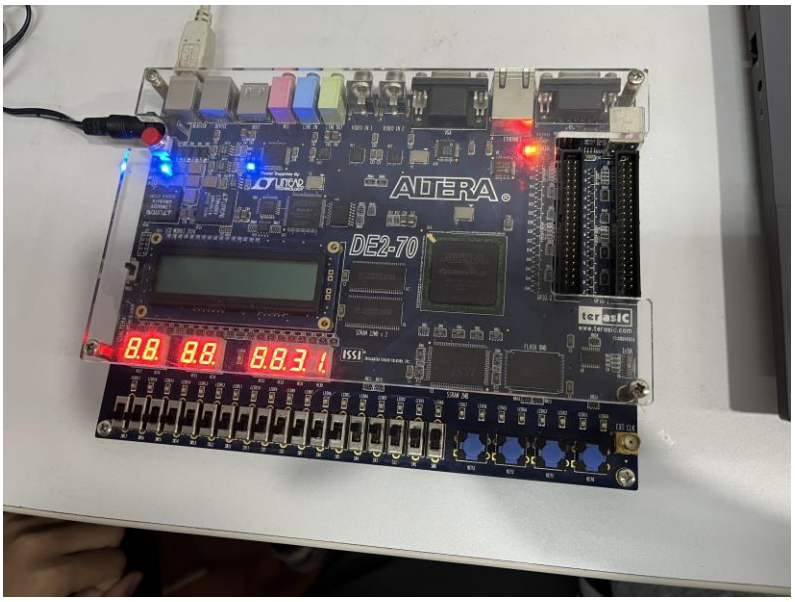


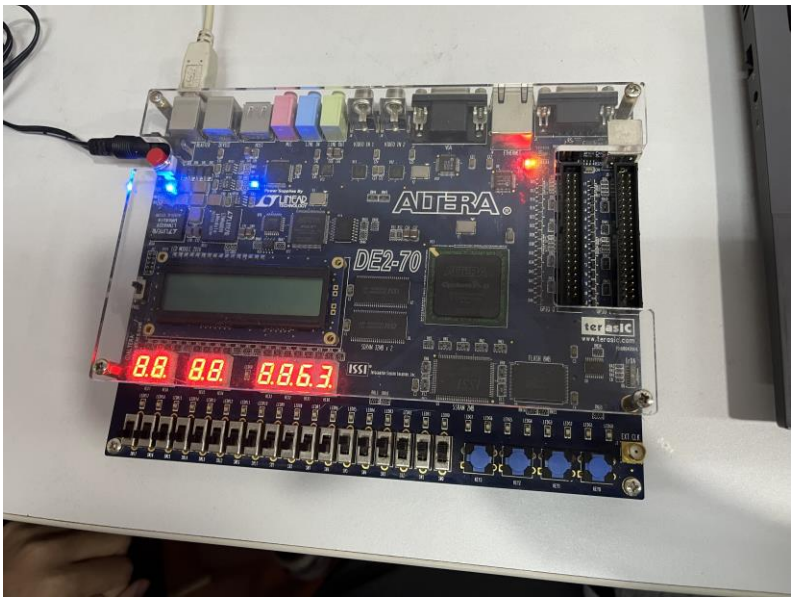

ALTERA
DE2-70
terasIC

Part VII


ALTERA
DE2-70
terasIC


ALTERA
DE2-70
terasIC





6. Discussion and Conclusion:

Discussion I

There's no fault in the codes above. However, using only if-else, case or Boolean expressions is too foolish. Because too many assignment expressions are included in the code, we can add an additional vhdl file char_7seg.vhd, in which we transfer a binary number to an 8-bit HEX display. Thus the code could be more clean and easy to read.

Discussion II

When burning code in Part VI to FPGA board, I noticed that the maximum value of sum was limited at 93. However, when burning again, the problem has been solved. The only difference during burning is that I operated the switch during burning. We infer that the operation influenced the inner structure of the board, and thus some Boolean expressions did not work.