LAB REPORT
Laboratory Exercise 1: Switches, Lights and Multiplexers

3220104688 杨佳昕

28 Nov, 2024

1. Problem Description

Part I

The DE2 board provides 18 toggle switches called SW17-0, that can be used as inputs to a circuit, and 18 red lights LEDR17-0 used to display. A code has been given, showing a simple VHDL entity that uses these switches and their states on the LEDS, i.e., LEDR <= SW. Pin location assignments are given in the csv document.

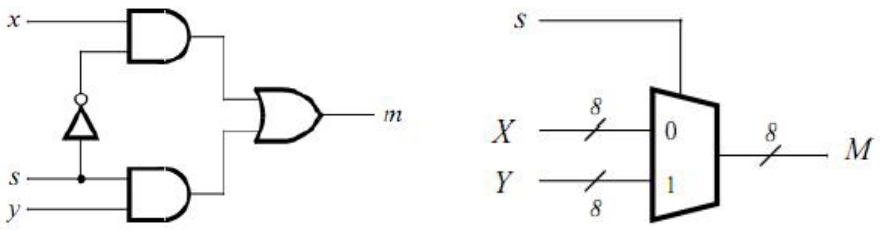
1. Create a project and compile the VHDL entity.
2. Include the project in the required pin assignments for the DE2 board.
3. Download the compiled circuit into the FPGA chip. Test the functionality.

Part II

A sum-of-products circuit and its symbol has been shown, in which when s=0, m=x, else, s=1, m=y. The multiplexer can be described by the VHDL statement: m <= (not(s) and x) or (s and y). Now extend the project into an 8-bit 2-to-1 multiplexer, where x, y and m are 8-bit inputs and output, resembling the symbol as follows.

1. Create a project and compile the VHDL entity.
2. Include the project in the required pin assignments for the DE2 board.
3. Download the compiled circuit into the FPGA chip. Test the functionality.

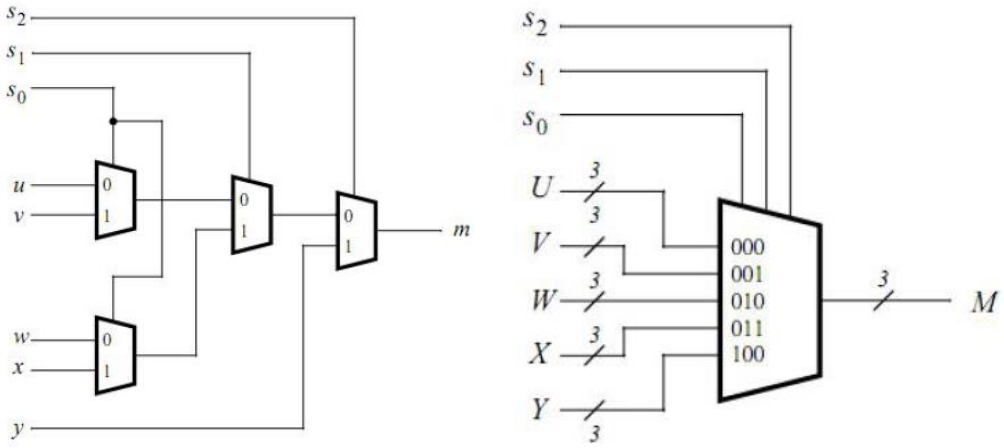
NB: SW17 as s input, SW7-0 as x input and SW15-8 as y input.



Part III

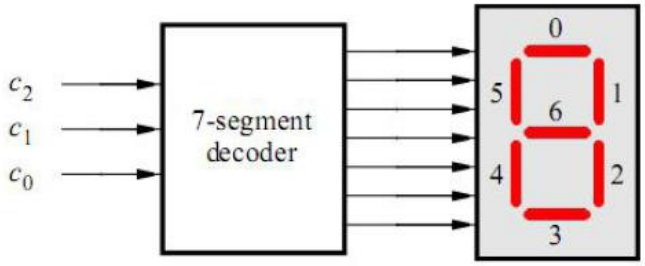
A 5-to-1 multiplexer by using 2-to-1 multiplexers is shown as follows. Now extend it to an 8-bit 5-to-1 multiplexer, where SW17-15 as s input and SW14-0 as the five 3-bit inputs u to y. Moreover, connect SW switches to the red lights LEDR and connect the output m to green lights LEDG2-0.

1. Create a project and compile the VHDL entity.
2. Include the project in the required pin assignments for the DE2 board.
3. Download the compiled circuit into the FPGA chip. Test the functionality.



Part IV

The graph in the following shows a 7-segment decoder module that has the 3-bit input and the 7-bit output. Table below lists the characters that should be displayed for each valuation of input. To keep the design simple, only 4 characters are included in the table.



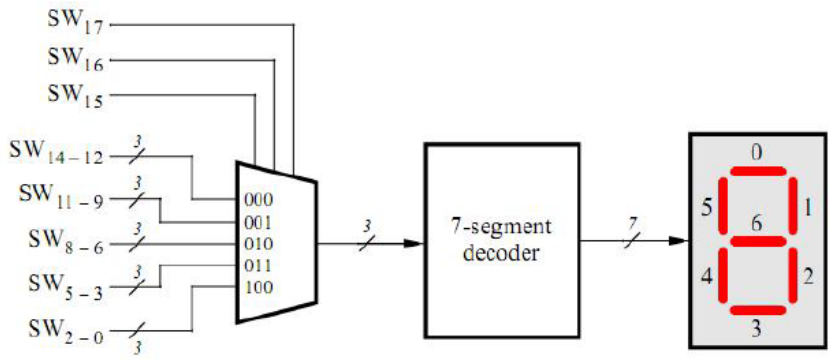
c2c1c0	000	001	010	011	100	101	110	111
character	H	E	L	O				

1. Create a project and compile the VHDL entity, in which a Boolean expression is used.
2. Include the project in the required pin assignments for the DE2 board.
3. Download the compiled circuit into the FPGA chip. Test the functionality.

Part V

We need to build a circuit that can display five characters as the graph below. Note that we have used the circuits from Parts III and IV as subcircuits in this code. The purpose is to display any word on the five displays that is composed of the characters in the table as follows. Input s, i.e.,

SW17-15 will determine the output, while SW2-0 as u, SW5-3 as v, SW8-6 as w, SW11-9 as x and SW14-12 as y are the inputs. Parts of the code shown, fill the whole project.



SW17SW16SW15	Character pattern
000	H E L L O
001	E L L O H
010	L L O H E
011	L O H E L
100	O H E L L

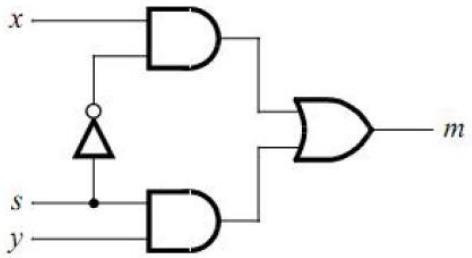
1. Create a project and compile the VHDL entity.
2. Include the project in the required pin assignments for the DE2 board.
3. Download the compiled circuit into the FPGA chip. Test the functionality.

2. Design Formulation

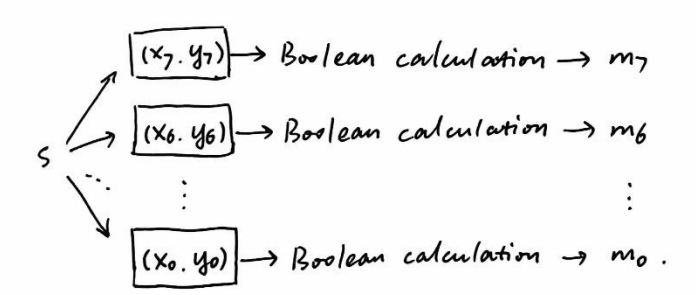
Part I

This part is easy to solve, just ledr <= sw.

Part II

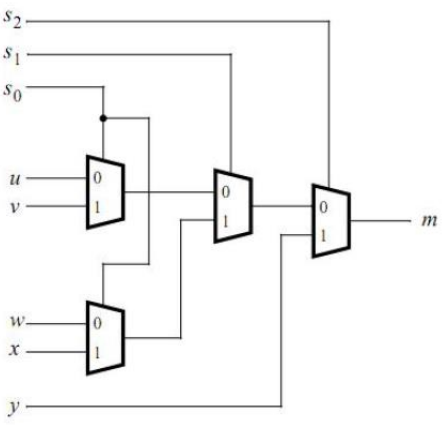


This is the sum-of-products circuit, i.e., m = (x and not(s)) or (s and y). The 8-bit 2-to-1 multiplexer is actually a combination of the circuit above, resembling the graph as follows.



Thus we can use a parallel structure to calculate m7-0(in the code is ledg7-0). And what's more, ledr is asked to be the same as sw, so we add an additional parallel structure to describe variable ledr.

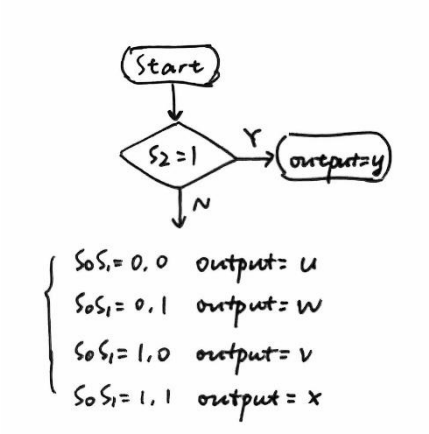
Part III



Truth table

S_0S_1 \ S_2	0	1
00	u	y
01	w	y
10	v	y
11	x	y

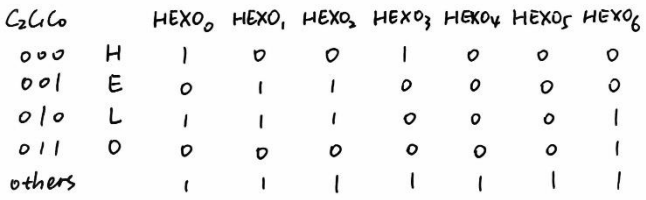
Let's consider the 1-dimension 5-to-1 multiplexer first. The truth table is shown above. From the truth table, we can list the flow chart as follows.



Just use if-elsif-else to judge whether the output is u, v, w, x or y. Now put the multiplexer into 3-dimention. The method is the same as that in Part II, and remember to define ledr17-0 with a parallel structure.

Part IV

HEX = 0 means the light is on, and 1 means the light is off. Truth table:



$C_2C_1C_0$		$HEX0_0$	$HEX0_1$	$HEX0_2$	$HEX0_3$	$HEX0_4$	$HEX0_5$	$HEX0_6$
000	H	1	0	0	1	0	0	0
001	E	0	1	1	0	0	0	0
010	L	1	1	1	0	0	0	1
011	O	0	0	0	0	0	0	1
others		1	1	1	1	1	1	1

From the truth table above, we can infer the Boolean expression:

HEX00 = not c0;

HEX01 = c0 xor c1, the same as HEX02;

HEX03 = not(c0 or c1);

HEX04 = 0, the same as HEX05;

HEX06 = c1.

And what's more, if considering the variable c2, we should add "or c2"to each Boolean expression because when c2 = 1, the result is always 1.

Thus, we can add the Boolean expression in the code, linking output to input directly.

Part V

From the code given, we know the whole project is divided into 3 parts:

Part A: the main part, combining Part B and C.

Part B: part for choosing one as output(also the input of the decoder following) from the inputs u, v, w, x, y depending on s.

Part C: part for arrange the sequence of five characters depending on s.

First, let's talk about the input SW14-0. Part V has shown that SW14-0 fit the table in Part IV, and thus to be simply, we suppose the input is HELLO, which is:

u is out while s = 000			v is out while s = 001			w is out while s = 010			x is out while s = 011			y is out while s = 100		
H			E			L			L			O		
SW14-12			SW11-9			SW8-6			SW5-3			SW2-0		
0	0	0	0	0	1	0	1	0	0	1	0	0	1	1

The binary 000001010010011(constant values) is the input SW14-0.

When s is given, one input variable (3-bit long) will be selected out as the input of the decoder in the following. Name the variable c.

Now consider s = 000, which means the output should follow the sequence H-E-L-L-O. When s = 0, c = 000, meaning H in the table in Part IV. Therefore, if following the steps below, the requirement will be achieved:

1. s = 000, c = 000. Set that when c = 000, the output (HEX4) is H.

2. s plus 1 to itself, s = 001, then c = 001. Set that when c = 001, the output (HEX3) is E.

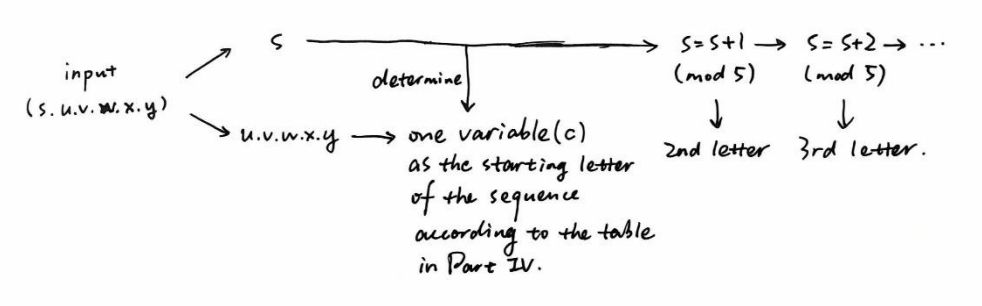
3. s plus 10(2 in decimal) to itself, s = 010, then c = 010. Set that when c = 010, the output (HEX2) is L.

4. s plus 11(3 in decimal) to itself, s = 011, then c = 010. Set that when c = 010, the output (HEX1) is L.

5. s plus 100(4 in decimal) to itself, s = 100, then c = 011. Set that when c = 011, the output (HEX0) is O.

If s is bigger than 100, then minus 100 to itself(i.e., mod 5) to set a rotation of s. Then, the five characters can be arranged in the right sequence.

The whole process can be summarized as a mind map as follows:



3. Design Entry

Part I

part1.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity part1 is
5       port(
6           sw : in std_logic_vector(17 downto 0);
7           ledr : out std_logic_vector(17 downto 0));
8   end part1;
9
10  architecture behavior of part1 is
11      begin
12      ledr <= sw;
13  end behavior;
```

Part II

part2.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity part2 is
5       port(
6           s : in std_logic;
7           x : in std_logic_vector(7 downto 0);
8           y : in std_logic_vector(7 downto 0);
9           ledg : out std_logic_vector(7 downto 0);
10          ledr : out std_logic_vector(17 downto 0));
11  end part2;
12
13  architecture behavior of part2 is
14  begin
```

```
15      ledg(0) <= (not(s) and x(0)) or (s and y(0));
16      ledg(1) <= (not(s) and x(1)) or (s and y(1));
17      ledg(2) <= (not(s) and x(2)) or (s and y(2));
18      ledg(3) <= (not(s) and x(3)) or (s and y(3));
19      ledg(4) <= (not(s) and x(4)) or (s and y(4));
20      ledg(5) <= (not(s) and x(5)) or (s and y(5));
21      ledg(6) <= (not(s) and x(6)) or (s and y(6));
22      ledg(7) <= (not(s) and x(7)) or (s and y(7));
23      ledr(0) <= x(0);
24      ledr(1) <= x(1);
25      ledr(2) <= x(2);
26      ledr(3) <= x(3);
27      ledr(4) <= x(4);
28      ledr(5) <= x(5);
29      ledr(6) <= x(6);
30      ledr(7) <= x(7);
31      ledr(8) <= y(0);
32      ledr(9) <= y(1);
33      ledr(10) <= y(2);
34      ledr(11) <= y(3);
35      ledr(12) <= y(4);
36      ledr(13) <= y(5);
37      ledr(14) <= y(6);
38      ledr(15) <= y(7);
39      ledr(17) <= s;
40  end behavior;
```

Part III

part3.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity part3 is
5       port(
6           s2, s1, s0 : in std_logic;
7           u, v, w, x, y : in std_logic_vector(2 downto 0);
8           ledr : out std_logic_vector(17 downto 0);
9           m : out std_logic_vector(2 downto 0));
10
11  end part3;
12
13  architecture behavior of part3 is
14  begin
15      ledr(0) <= u(0);
16      ledr(1) <= u(1);
17      ledr(2) <= u(2);
18      ledr(3) <= v(0);
19      ledr(4) <= v(1);
20      ledr(5) <= v(2);
21      ledr(6) <= w(0);
22      ledr(7) <= w(1);
23      ledr(8) <= w(2);
24      ledr(9) <= x(0);
25      ledr(10) <= x(1);
26      ledr(11) <= x(2);
27      ledr(12) <= y(0);
28      ledr(13) <= y(1);
29      ledr(14) <= y(2);
30      ledr(15) <= s0;
31      ledr(16) <= s1;
32      ledr(17) <= s2;
33
```

```
34      process(u, v, w, x, y)
35      begin
36          if s2 = '1' then
37              m <= y;
38          elsif s2 = '0' and s0 = '0' and s1 = '0' then
39              m <= u;
40          elsif s2 = '0' and s0 = '0' and s1 = '1' then
41              m <= w;
42          elsif s2 = '0' and s0 = '1' and s1 = '0' then
43              m <= v;
44          elsif s2 = '0' and s0 = '1' and s1 = '1' then
45              m <= x;
46          else
47              m <= "000";
48          end if;
49      end process;
50  end behavior;
```

Part IV

part4.vhd

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity part4 is
5       port(
6           c2, c1, c0 : in std_logic;
7           hex0 : out std_logic_vector(6 downto 0));
8   end part4;
9
10  architecture behavior of part4 is
11  begin
12      hex0(0) <= not c0 or c2;
13      hex0(1) <= (c0 xor c1) or c2;
14      hex0(2) <= (c0 xor c1) or c2;
15      hex0(3) <= not(c0 or c1) or c2;
16      hex0(4) <= c2;
17      hex0(5) <= c2;
18      hex0(6) <= c1 or c2;
19  end behavior;
```

Part V

part5.vhd (top-level entity)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity part5 is
6       port(
7           sw : in std_logic_vector(17 downto 0);
8           hex0, hex1, hex2, hex3, hex4 : out std_logic_vector(0 to 6));
9   end part5;
10
11  architecture behavior of part5 is
12      component mux_3bit_5to1
13          port(
14              s, u, v, w, x, y : in std_logic_vector(2 downto 0);
15              m : out std_logic_vector(2 downto 0));
16      end component;
17
18      component char_7seg
19          port(
20              c : in std_logic_vector(2 downto 0);
21              display : out std_logic_vector(6 downto 0));
22      end component;
23
24      signal m0, m1, m2, m3, m4 : std_logic_vector(2 downto 0);
25      signal s_plus_0, s_plus_1, s_plus_2, s_plus_3, s_plus_4 : std_logic_vector(2 downto 0);
26
27      begin
28          process(sw)
29                  variable temp : unsigned(3 downto 0);
30              begin
31                  temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))), 4) mod 5;
32                  s_plus_0 <= std_logic_vector(temp(2 downto 0));
33
34                  temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))) + 1, 4) mod 5;
35                  s_plus_1 <= std_logic_vector(temp(2 downto 0));
36
37                  temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))) + 2, 4) mod 5;
38                  s_plus_2 <= std_logic_vector(temp(2 downto 0));
39
40                  temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))) + 3, 4) mod 5;
41                  s_plus_3 <= std_logic_vector(temp(2 downto 0));
42
43                  temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))) + 4, 4) mod 5;
44                  s_plus_4 <= std_logic_vector(temp(2 downto 0));
45              end process;
```

```
46
47          mux0 : mux_3bit_5to1 port map(
48              s_plus_0, sw(14 downto 12), sw(11 downto 9),
49              sw(8 downto 6), sw(5 downto 3), sw(2 downto 0), m0);
50          mux1 : mux_3bit_5to1 port map(
51              s_plus_1, sw(14 downto 12), sw(11 downto 9),
52              sw(8 downto 6), sw(5 downto 3), sw(2 downto 0), m1);
53          mux2 : mux_3bit_5to1 port map(
54              s_plus_2, sw(14 downto 12), sw(11 downto 9),
55              sw(8 downto 6), sw(5 downto 3), sw(2 downto 0), m2);
56          mux3 : mux_3bit_5to1 port map(
57              s_plus_3, sw(14 downto 12), sw(11 downto 9),
58              sw(8 downto 6), sw(5 downto 3), sw(2 downto 0), m3);
59          mux4 : mux_3bit_5to1 port map(
60              s_plus_4, sw(14 downto 12), sw(11 downto 9),
61              sw(8 downto 6), sw(5 downto 3), sw(2 downto 0), m4);
62
63          h4 : char_7seg port map(m0, hex4);
64          h3 : char_7seg port map(m1, hex3);
65          h2 : char_7seg port map(m2, hex2);
66          h1 : char_7seg port map(m3, hex1);
67          h0 : char_7seg port map(m4, hex0);
68  end behavior;
```

mux_3bit_5to1.vhd (second-top-level entity, used as Part B mentioned above)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity mux_3bit_5to1 is
5       port(
6       s, u, v, w, x, y : in std_logic_vector(2 downto 0);
7       m : out std_logic_vector(2 downto 0));
8   end mux_3bit_5to1;
9
10  architecture behavior of mux_3bit_5to1 is
11  begin
12      process(s, u, v, w, x, y)
13      begin
14          if s = "000" then
15              m <= u;
16          elsif s = "001" then
17              m <= v;
18          elsif s = "010" then
19              m <= w;
20          elsif s = "011" then
21              m <= x;
22          elsif s = "100" then
23              m <= y;
24          else
25              m <= "000";
26          end if;
27      end process;
28  end behavior;
```

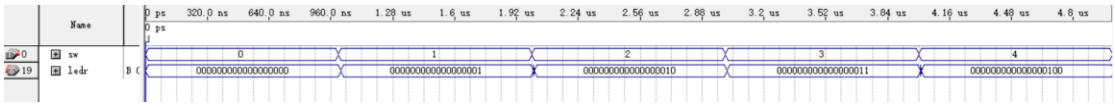
char_7seg.vhd (second-top-level entity, used as Part C mentioned above)

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity char_7seg is
5       port(
6           c : in std_logic_vector(2 downto 0);
7           display : out std_logic_vector(0 to 6));
8   end char_7seg;
9
10  architecture behavior of char_7seg is
11  begin
12      process(c)
13      begin
14          if c = "000" then
15              display <= "0001001";
16          elsif c = "001" then
17              display <= "0000110";
18          elsif c = "010" then
19              display <= "1000111";
20          elsif c = "011" then
21              display <= "1000000";
22          elsif c = "100" then
23              display <= "1111111";
24          else
25              display <= "1111111";
```

```
        end if;
    end process;
end behavior;
```

4. Simulation and Synthesis Results

Part I

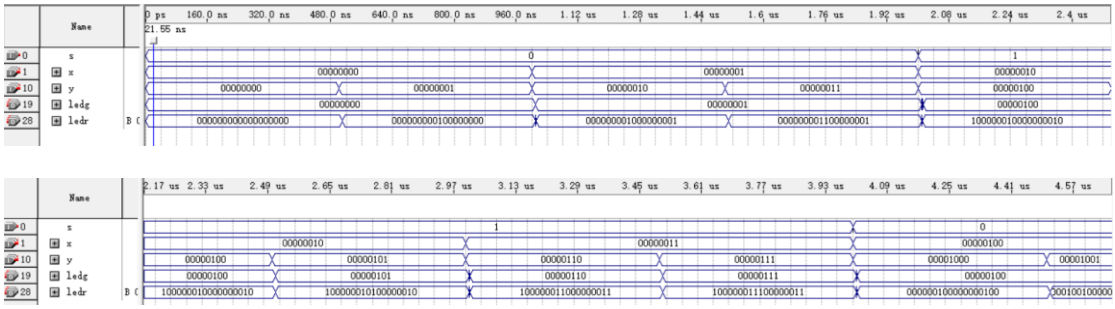


It is clear that the output wave is the same as that of the input. If end time is selected shorter, we can find that there is an around-10ns delay.

NB: All waveform simulations used timing simulation mode.

Part II

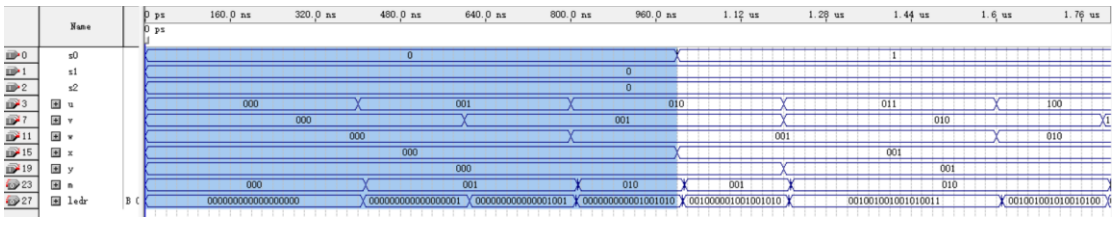
Waveform of s, x, y and ledg, ledr:



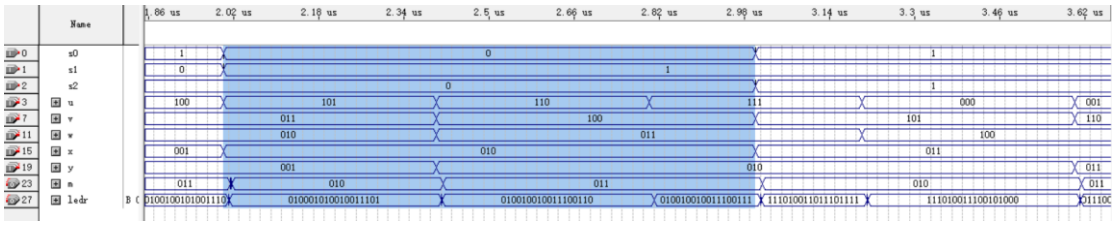
To make the relation(s=0, ledg=x and s=1, ledg=y) more clear, we made the time interval of x and y different. It's clear in the graph that when s=0, ledg is the same as x, while the same as y when s=1.

Variable ledr equals to all the input s, y and x, which cover SW17, SW15-8 and SW7-0 respectively, therefore ledr17 = s, ledr15-8 = y7-0 and ledr7-0 = x7-0. The graph above has well shown this relationship.

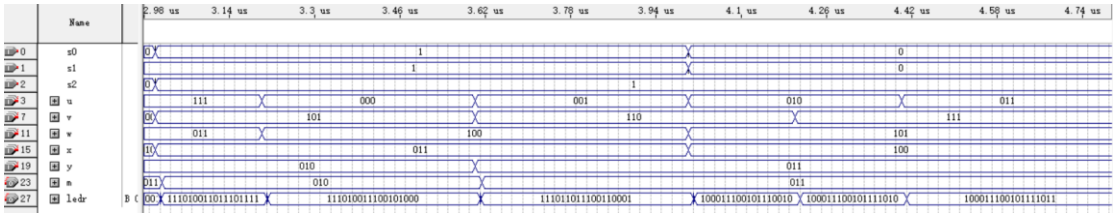
Part III



When s0s1s2 = 000, the waveform of output m is the same as that of input u.



When s0s1s2 = 010, the waveform of output m is the same as that of input w.



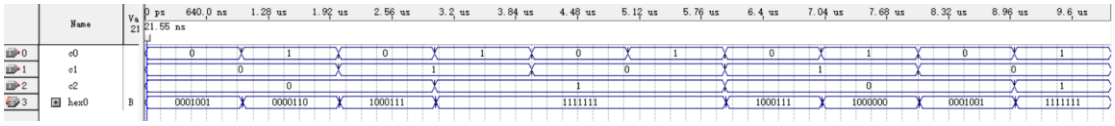
When s2 = 1, the waveform of output m is the same as that of input y.

All of other s0s1s2 situation fits the truth table, and we do not list here.

The waveform of ledr is the same as that of the input variables, i.e., SW17-15 for s2-0, and SW14-0 for all the variables from u to y (although the waveforms of v and x are not shown).

Part IV

H(HEX06-00): 0001001, E: 0000110, L: 1000111, O: 1000000, others: 1111111.



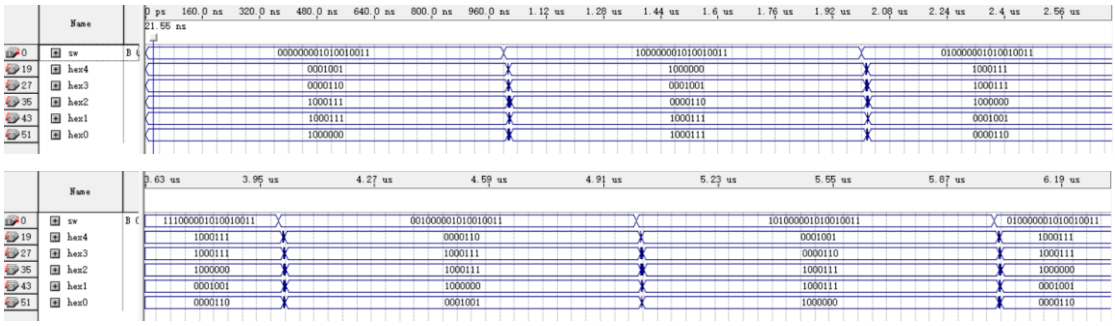
When the input c2c1c0 is 000, the output should be H(0001001), and the output waveform is correct.

The same as 001 for E, 010 for L, 011 for O and 1xx for the others.

Notice: The variable c is in positive sequence, different from c2c1c0.

Part V

Because the inputs SW14-0 are constant values, set these variables constant.



When s = 000, output HELLO (0001001 0000110 1000111 1000111 1000000).

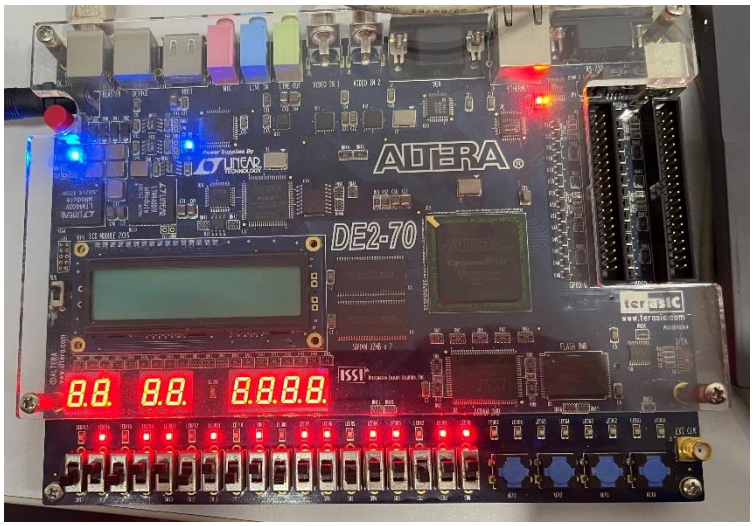
When s = 100, output OHELL (1000000 0001001 0000110 1000111 1000111).

When s = 010, output LLOHE (1000111 1000111 1000000 0001001 0000110).

When s = 001, output ELLOH (0000110 1000111 1000111 1000000 0001001)

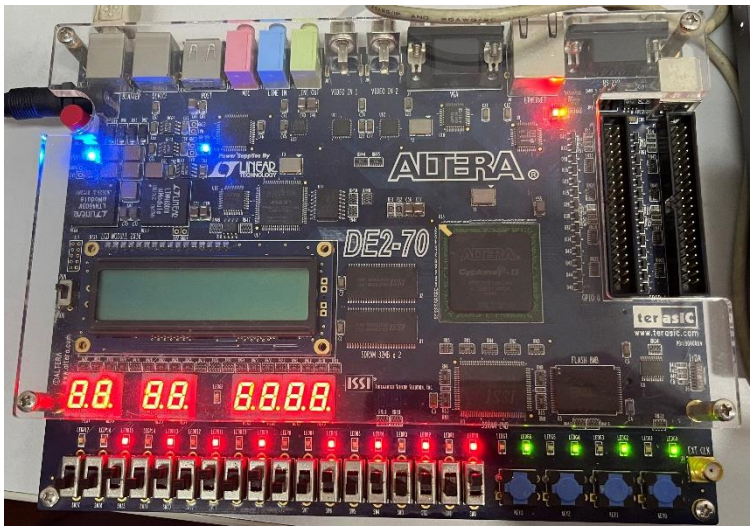
5. Experimental Results

Part I

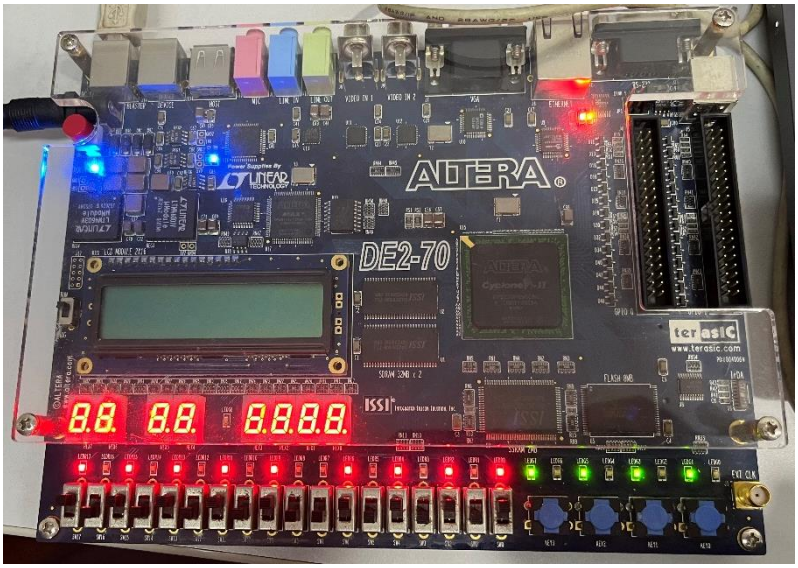


Part II

s = 0, ledg = x:

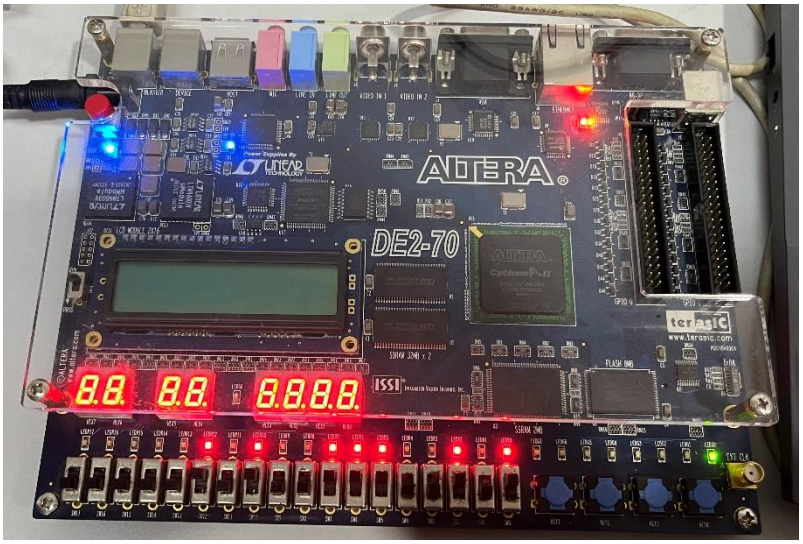


s = 1, ledg = y:

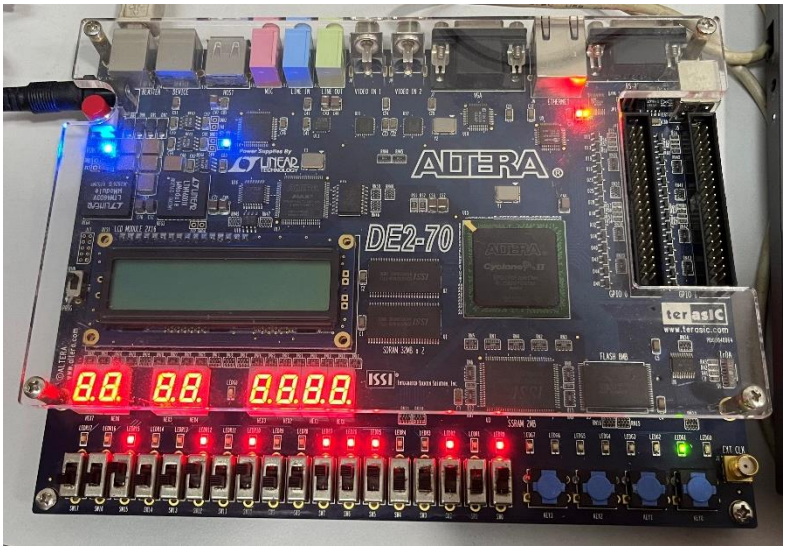


Part III

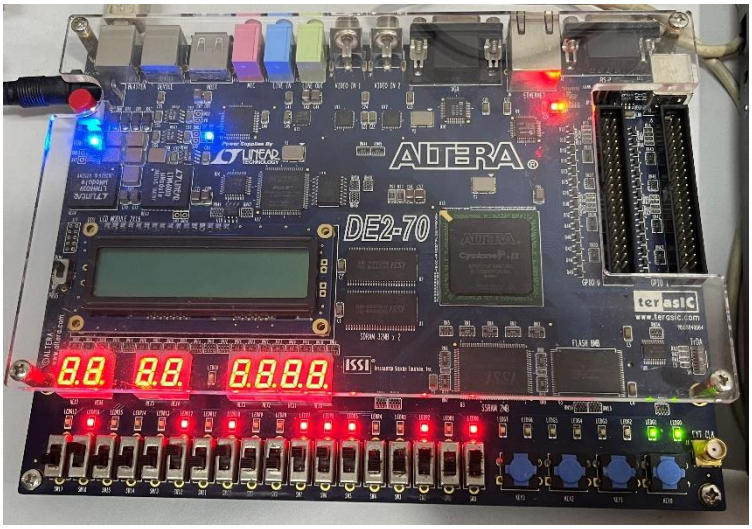
s2s1s0 = 000, m = u:



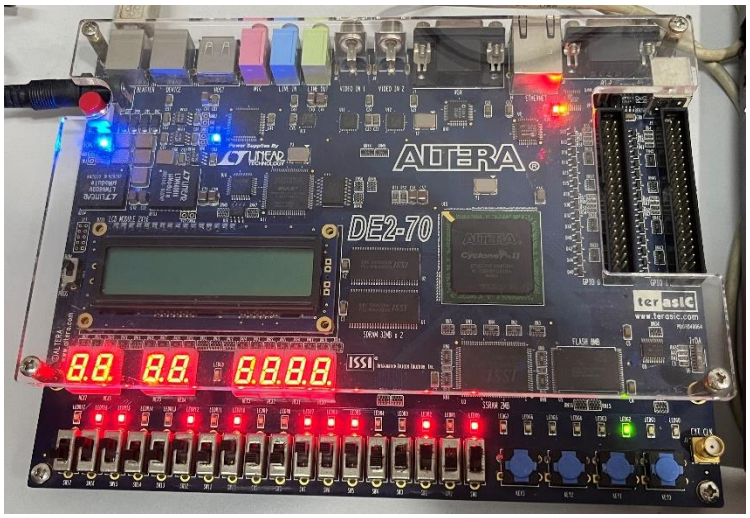
s2s1s0 = 001, m = v:



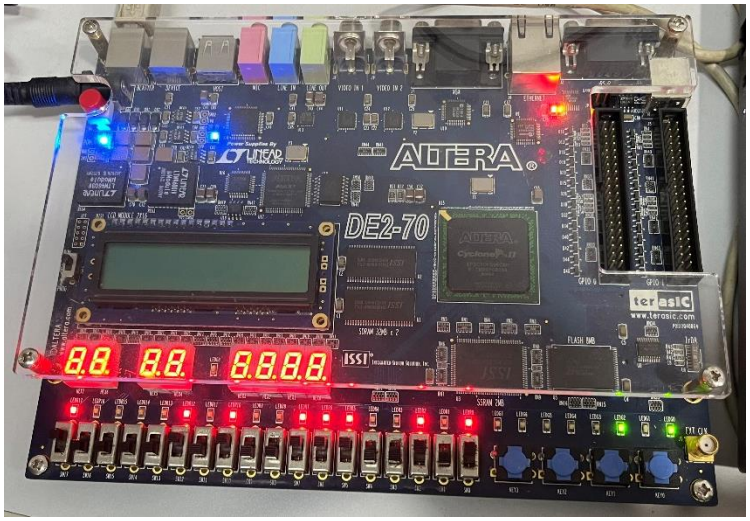
s2s1s0 = 010, m = w:



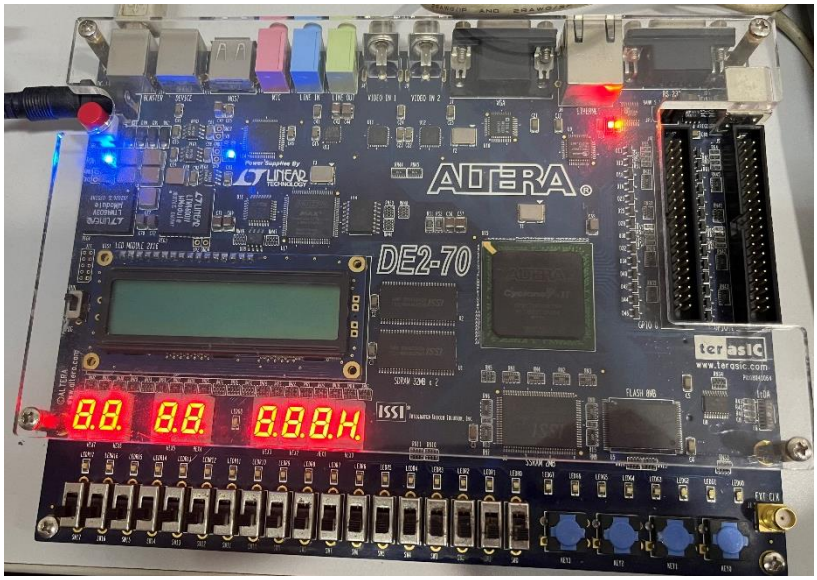
s2s1s0 = 011, m = x:

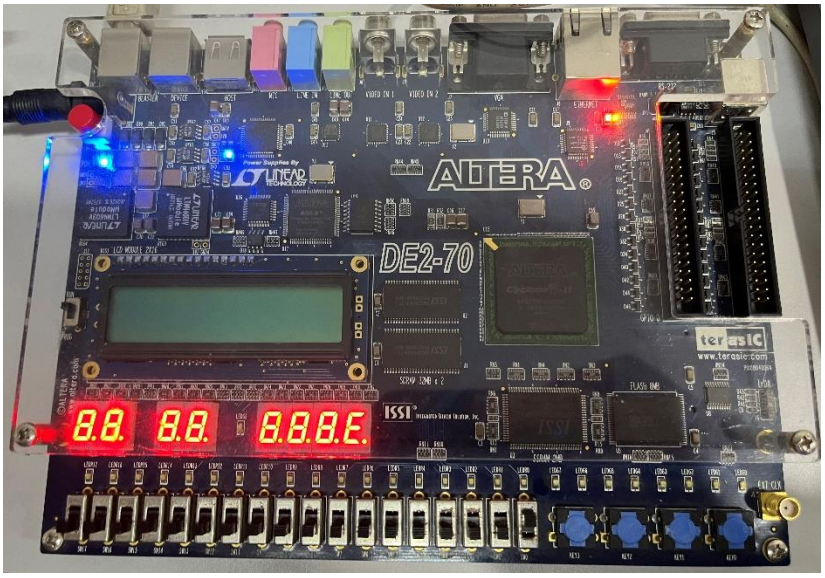


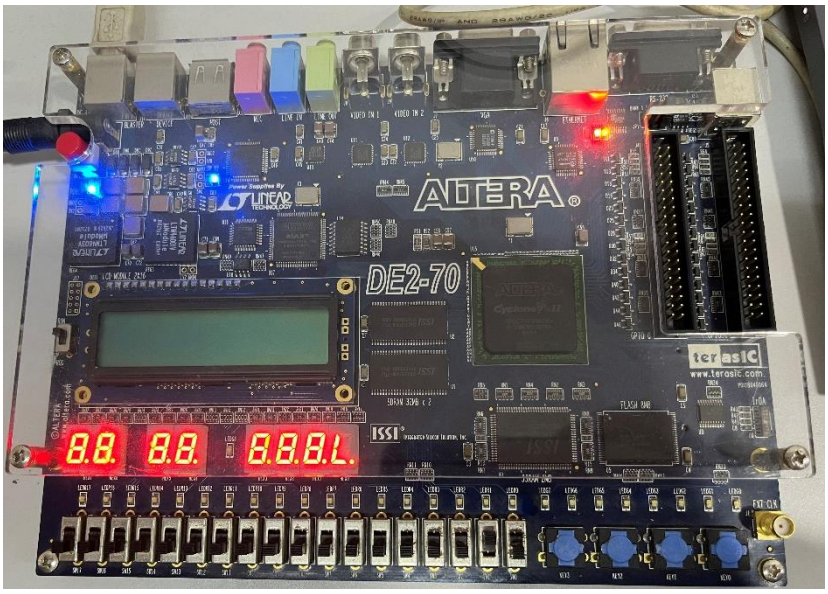
s2 = 1, m = y:

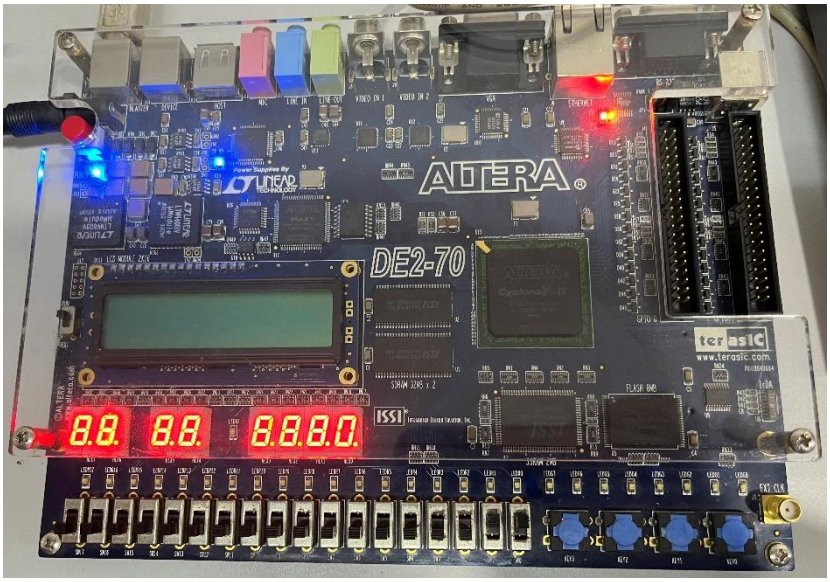


Part IV

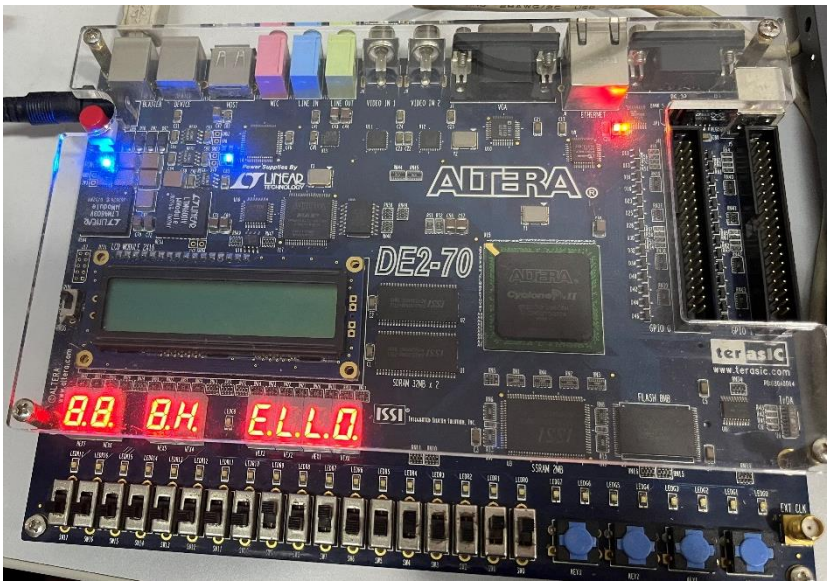


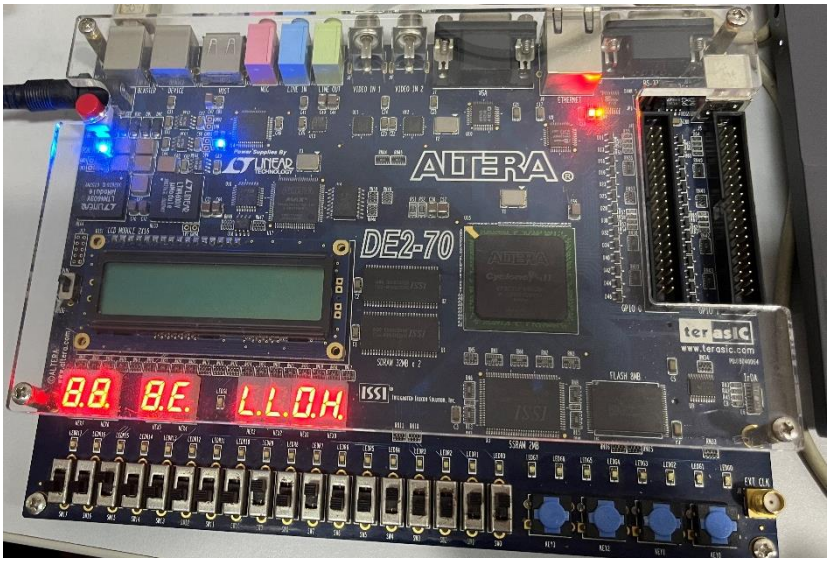

ALTERA
DE2-70
terasIC
www.terasic.com

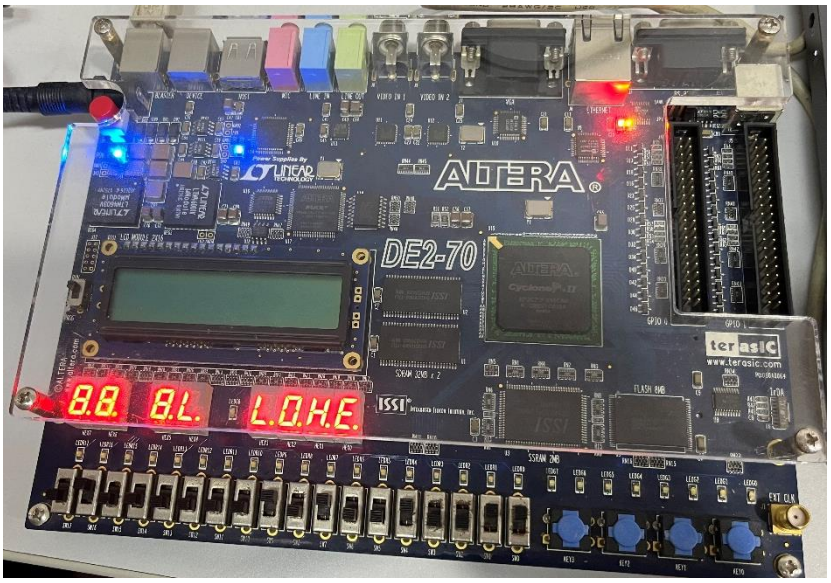

ALTERA
DE2-70
terasIC
www.terasic.com

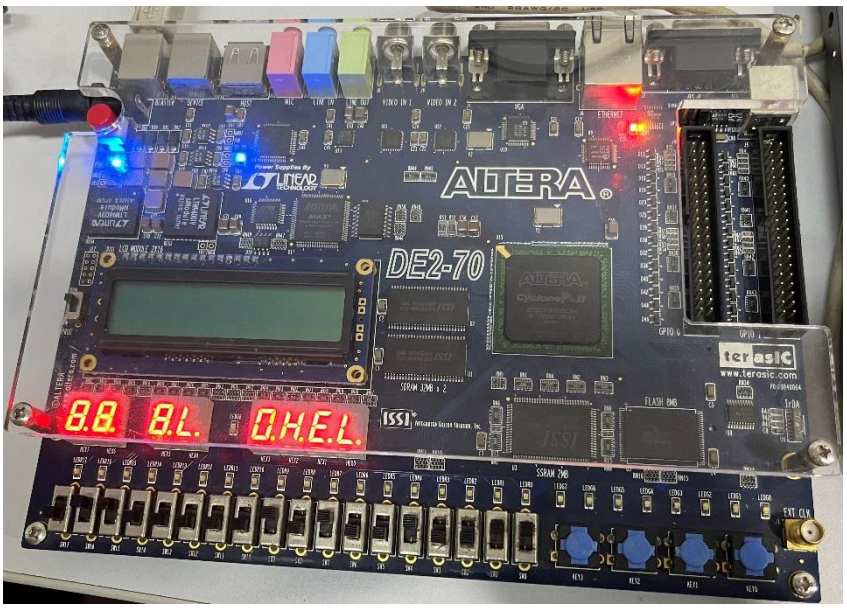

ALTERA
DE2-70
terasIC
www.terasic.com

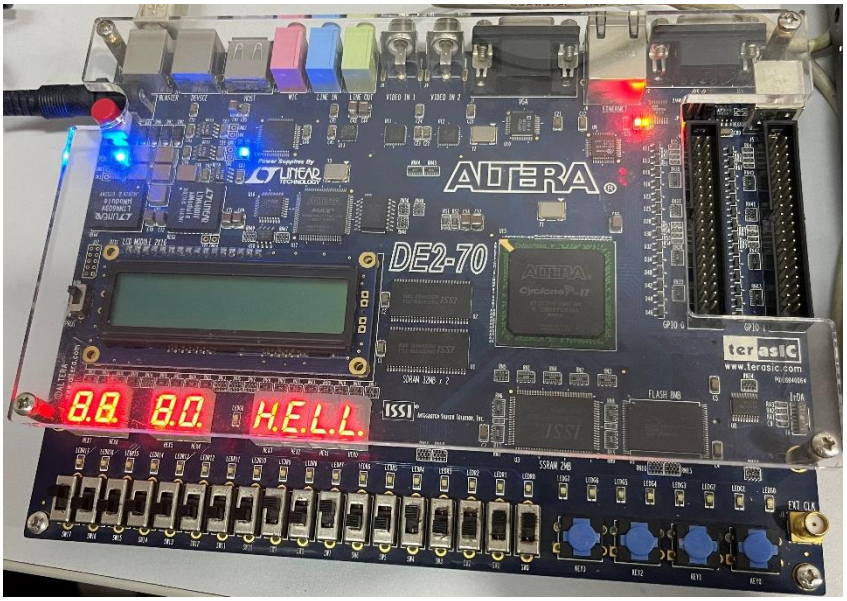
Part V


ALTERA
DE2-70
8.8. 8.H. E.L.L.O.


ALTERA
DE2-70
8.8. 8.E. L.L.O.H.


ALTERA
DE2-70
8.8. 8.L. L.O.H.E.





6. Discussion and Conclusion

Problem I(in Part V):

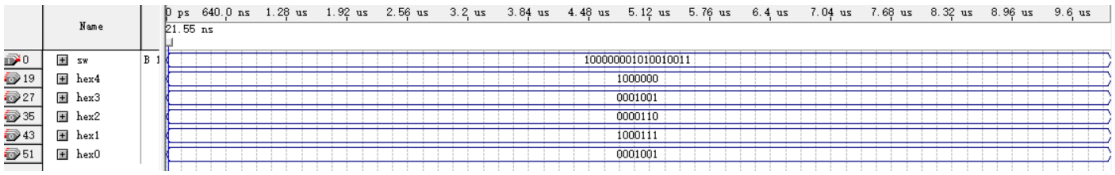
False:

Code:

```
        temp := unsigned(sw(17 downto 15)) mod 5;
        s_plus_0 <= std_logic_vector(temp(2 downto 0));

        temp := (unsigned(sw(17 downto 15)) + 1) mod 5;
        s_plus_1 <= std_logic_vector(temp(2 downto 0));

        temp := (unsigned(sw(17 downto 15)) + 2) mod 5;
        s_plus_2 <= std_logic_vector(temp(2 downto 0));

        temp := (unsigned(sw(17 downto 15)) + 3) mod 5;
        s_plus_3 <= std_logic_vector(temp(2 downto 0));

        temp := (unsigned(sw(17 downto 15)) + 4) mod 5;
        s_plus_4 <= std_logic_vector(temp(2 downto 0));
    end process;
```

Simulation result:

The error happened when s = 100 only.



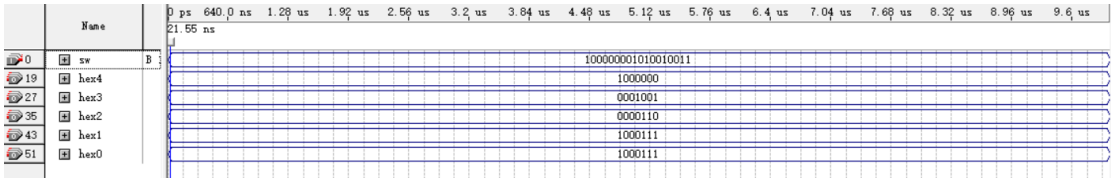
When s = 100, the output is OHELH.

Correct:

Code:

```
process(sw)
        variable temp : unsigned(3 downto 0);
    begin
        temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))), 4) mod 5;
        s_plus_0 <= std_logic_vector(temp(2 downto 0));

        temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))) + 1, 4) mod 5;
        s_plus_1 <= std_logic_vector(temp(2 downto 0));

        temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))) + 2, 4) mod 5;
        s_plus_2 <= std_logic_vector(temp(2 downto 0));

        temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))) + 3, 4) mod 5;
        s_plus_3 <= std_logic_vector(temp(2 downto 0));

        temp := to_unsigned(to_integer(unsigned(sw(17 downto 15))) + 4, 4) mod 5;
        s_plus_4 <= std_logic_vector(temp(2 downto 0));
    end process;
```

Simulation result:



When s = 100, the output is OHELL.

So why does the problem happened?

See the false code “temp := (unsigned(sw(17 downto 15)) + 4) mod 5”:

$$sw(17\ downto\ 15) = 100$$

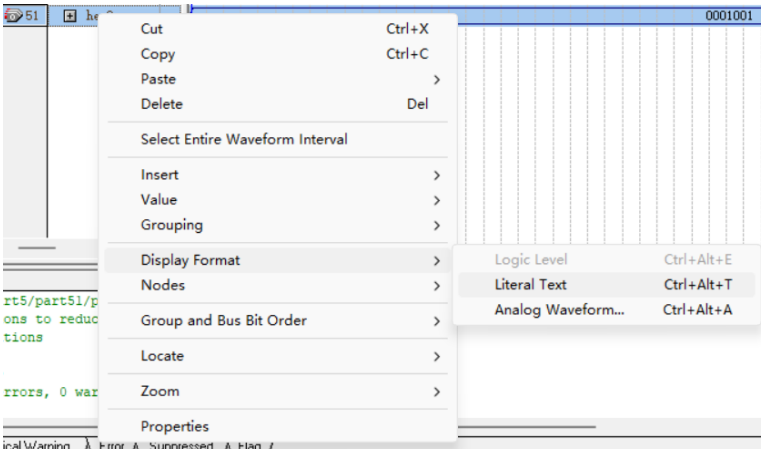
$$unsigned\big(sw(17\ downto\ 15)\big) = 4$$

$$temp = (4+4)\ mod\ 5 = 3 \rightarrow 100\ in\ binary$$

The ideal value for temp here is 000, because the ideal process is to extract the last 3 bits of the binary form of 4+4=8. But actually, the result is 100 due to the calculation in decimal before binary transfer. Therefore, we have to make the binary transfer ahead of mod calculation, just as the correct code shows.

Problem II:

In simulation, TA said I forgot to change the simulation form into binary. But actually, that is not the case. Even I changed the form into binary, the input and output still display in a waveform state. The solution is to change the display format to literal text (ctrl+alt+T in keyboard), just like the graph shown below.



Only with this button pressed would the simulation transferred to the binary form, and this was not noted in the slide given.

Other algorithms concerning the experiment are all noted in Design Formulation.

Conclusion:

This lab exercise covered the implementation of various digital circuits on an FPGA using VHDL. I successfully designed and tested a basic switch-LED interface, an 8-bit 2-to-1 multiplexer, an 8-bit 5-to-1 multiplexer, a 7-segment decoder, and a system to display words on multiple 7-segment displays. Each part was verified through simulation and physical testing, confirming the correct functionality of the circuits. The exercises provided hands-on experience with VHDL and FPGA programming, reinforcing my understanding of digital logic design.

-- End --