LAB REPORT
******On Numbers and Display******

Author 洪晨辉 3220101111*

****Date 2024.12.3****

1. Problem Description

1）Design a 4-to-2 bit binary to BCD digital circuit using the certain structure provided in the FPGA guidance. Display it through two seven-segment display.

2）Make a 2 digit BCD adder.

3）There's a specific ingenious way to implement a 2 digit BCD adder with logic provided below. Try it out.

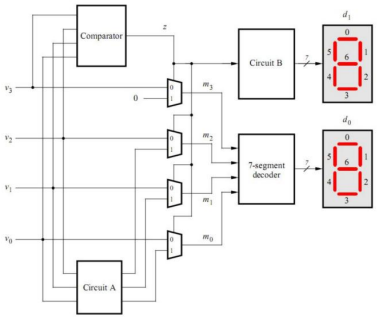
1 T0 = A0 + B0 2if(T0 > 9) then 3 Z0 =10; 4 c1 =1; 5 else 6 Z0 =0; 7 c1 =0; 8 end if 9 S0 = T0 − Z0 10 T1 = A1 + B1 + c1 11 if (T1 > 9) then 12 Z1 =10; 13 c2 =1; 14 else 15 Z1 =0; 16 c2 =0; 17 end if 18 S1 = T1 − Z1 19 S2 = c2

4）Upgrade the 4-to-2 BCD conversion circuit into 6 bit one.

2. Design Formulation

Part II

As much as I love to, the guidance has pose restrictions on circuit design. It must consist of the following structure,



Which means the circuit must consists of these stuff,

Comparator: determines if the 4-bit binary input is greater than 3.

Circuit A and B: Adjust value for Ones or Tens.

Seven-segment decoder: You know the drill.

Part V

Theoretically, one should implement many adders and such. But for us lazy-bones, we just use some plus operand tips-and-tricks provided by IEEE standard library anyway. Not a big deal.

Part VI

Yay, we have if-else block now. Just copy and paste it and we'll be fine.

Part VII

Nah, the method provided by Part II is dumb. Why do those structural stuff when you could easily pull out the truth table implementation in your sleeve, and completely wretch the well-organized design into a deep abyss of horribly strangled logic gates? But hey, what do you expect? It's convenient! So don't judge me.

Truth table as follows,

Truth Table for 6-Bit Binary to BCD

Binary Input (6-bit)	Decimal Value	BCD Tens (MSB)	BCD Ones (LSB)
000000	0	0000	0000
000001	1	0000	0001
000010	2	0000	0010
000011	3	0000	0011
000100	4	0000	0100
000101	5	0000	0101
000110	6	0000	0110
000111	7	0000	0111
001000	8	0000	1000
001001	9	0000	1001
001010	10	0001	0000
001011	11	0001	0001
...	...	...	...
111110	62	0110	0010
111111	63	0110	0011

This table directly maps every 6-bit binary input to its equivalent BCD output.

3. Design Entry

Part II

Part2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity part2 is
    Port (
        sw : in STD_LOGIC_VECTOR(3 downto 0); -- 4-bit binary input
  hex1,hex0 : out STD_LOGIC_VECTOR(7 downto 0)
    );
end part2;

architecture Structural of part2 is
    -- Signals
    signal z          : STD_LOGIC; -- Comparator result
    signal bcd_tens   : STD_LOGIC; -- BCD tens digit
    signal bcd_ones   : STD_LOGIC_VECTOR(3 downto 0);  -- BCD ones digit

begin
    -- Instantiate Comparator
    COMP: entity work.Comparator4
        Port Map (
            binary_in => sw,
            z => z
        );

    -- Instantiate Circuit A (Ones)
    CIRCA: entity work.CircuitA4
        Port Map (
            binary_in => sw,
            z => z,
            bcd_ones => bcd_ones
        );

    -- Instantiate Circuit B (Tens)
    CIRCB: entity work.CircuitB4
        Port Map (
            z => z,
            bcd_tens => bcd_tens
        );

    HEX_1: entity work.char_7segB
        Port Map (
            c => bcd_tens,
            display => hex1
        );

 HEX_0: entity work.char_7seg
 Port Map (
  c => bcd_ones,
  display => hex0
 );

end Structural;
```

Comparator4.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Comparator4 is
    Port (
        binary_in : in STD_LOGIC_VECTOR(3 downto 0); -- 4-bit binary input
        z         : out STD_LOGIC -- High if binary_in >= 10
    );
end Comparator4;
```

```
10.
11. architecture PureLogic of Comparator
    4 is
12. begin
13.     -- z = 1 when binary_in >= 10
14.     z <= binary_in(3) and (binary_in
    (2) or binary_in(1));
15. end PureLogic;
```

CircuitA4.vhd

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity CircuitA4 is
5.      Port (
6.          binary_in : in STD_LOGIC_VEC
    TOR(3 downto 0); --4-bit binary input
7.          z         : in STD_LOGIC; --
     Comparator result (1 if tens digit
    is 1)
8.          bcd_ones  : out STD_LOGIC_VE
    CTOR(3 downto 0) -- Adjusted BCD one
    s digit
9.      );
10. end CircuitA4;
11.
12. architecture PureLogic of CircuitA4
    is
13.     signal subtract_10 : STD_LOGIC_V
    ECTOR(3 downto 0) := "1001"; -- Bina
    ry 9
14.     signal result      : STD_LOGIC_V
    ECTOR(3 downto 0); -- Result after s
    ubtraction
15.     signal borrow      : STD_LOGIC_V
    ECTOR(4 downto 0); -- Borrow propaga
    tion
16. begin
17.     -- Borrow propagation and subtra
    ction
18.     borrow(0) <= z; -- Borrow starts
     only when z = 1
19.     result(0) <= binary_in(0) xor su
    btract_10(0) xor borrow(0);
20.     borrow(1) <= (not binary_in(0) a
    nd subtract_10(0)) or
21.                  (not binary_in(0) a
    nd borrow(0)) or
22.                  (subtract_10(0) and
     borrow(0));
23.
24.     result(1) <= binary_in(1) xor su
    btract_10(1) xor borrow(1);
25.     borrow(2) <= (not binary_in(1) a
    nd subtract_10(1)) or
26.                  (not binary_in(1) a
    nd borrow(1)) or
27.                  (subtract_10(1) and
     borrow(1));
28.
29.     result(2) <= binary_in(2) xor su
    btract_10(2) xor borrow(2);
30.     borrow(3) <= (not binary_in(2) a
    nd subtract_10(2)) or
31.                  (not binary_in(2) a
    nd borrow(2)) or
32.                  (subtract_10(2) and
     borrow(2));
33.
34.     result(3) <= binary_in(3) xor su
    btract_10(3) xor borrow(3);
35.
36.     -- MUX to select adjusted or una
    djusted value
37.     bcd_ones(0) <= (not z and binary
    _in(0)) or (z and result(0));
38.     bcd_ones(1) <= (not z and binary
    _in(1)) or (z and result(1));
39.     bcd_ones(2) <= (not z and binary
    _in(2)) or (z and result(2));
40.     bcd_ones(3) <= (not z and binary
    _in(3)) or (z and result(3));
41. end PureLogic;
```

CircuitB4.vhd

```
1.  library IEEE;
2.  use IEEE.STD_LOGIC_1164.ALL;
3.
4.  entity CircuitB4 is
```

```
    Port (
        z        : in STD_LOGIC; -- Comparator result
        bcd_tens : out STD_LOGIC -- Tens digit
    );
end CircuitB4;

architecture PureLogic of CircuitB4 is
begin
    -- Tens digit logic
    bcd_tens <= z;
end PureLogic;
```

Char_7seg.vhd

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

entity char_7seg is
    port (
        c : in std_logic_vector(3 downto 0); -- 4-bit input character code
        display : out std_logic_vector(7 downto 0) -- 7-segment output
    );
end char_7seg;

architecture behavior of char_7seg is
begin
    process(c)
    begin
        case c is
            when "0000" => display <= "11000000"; -- 0
            when "0001" => display <= "11111001"; -- 1
            when "0010" => display <= "10100100"; -- 2
            when "0011" => display <= "10110000"; -- 3
            when "0100" => display <= "10011001"; -- 4
            when "0101" => display <= "10010010"; -- 5
            when "0110" => display <= "10000010"; -- 6
            when "0111" => display <= "11111000"; -- 7
            when "1000" => display <= "10000000"; -- 8
            when "1001" => display <= "10010000"; -- 9
            when others => display <= "11111111"; -- Default to blank
        end case;
    end process;
end behavior;
```

Char_7segB.vhd

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

entity char_7segB is
    port (
        c : in std_logic; -- 1-bit input character code
        display : out std_logic_vector(7 downto 0) -- 7-segment output
    );
end char_7segB;

architecture behavior of char_7segB is
begin
    process(c)
    begin
        case c is
            when '0' => display <= "11000000"; -- 0
            when '1' => display <= "11111001"; -- 1
            when others => display <= "11111111"; -- Default to blank
        end case;
    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(7 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
 hex_0: entity work.char_7seg
  Port Map(
   c => sum_ones_digit,
   display => hex0
  );

 hex_1: entity work.char_7seg
  Port Map(
   c => sum_tens_digit,
   display => hex1
  );

 hex_2: entity work.char_7seg
  Port Map(
   c => sum_hundreds_digit,
   display => hex2
  );

 hex_4: entity work.char_7seg
  Port Map(
   c => num_two_ones_digit,
   display => hex4
  );

 hex_5: entity work.char_7seg
 Port Map(
  c => num_two_tens_digit,
  display => hex5
  );

 hex_6: entity work.char_7seg
  Port Map(
   c => num_one_ones_digit,
   display => hex6
  );

 hex_7: entity work.char_7seg
  Port Map(
   c => num_one_tens_digit,
   display => hex7
  );

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

```
1.
```

```
73.  -- num_one display --
74.  num_one_tens_digit <= num_one / 10;
75.  num_one_ones_digit <= num_one mod 10;
76.
77.
78.  -- num_two display --
79.  num_two_tens_digit <= num_two / 10;
80.  num_two_ones_digit <= num_two mod 10;
81.
82.
83.  -- sum display --
84.  sum_hundreds_digit <= sum / 100;
85.  sum_tens_digit <= ((sum - 100 * sum_hundreds_digit) - (sum - 100 * sum_hundreds_digit) mod 10) / 10;
86.  sum_ones_digit <= sum mod 10;
87.
88. end process;
89. end behavior;
```

Char_7seg.vhd

```
1.  library ieee;
2.  use ieee.std_logic_1164.all;
3.
4.  entity char_7seg is
5.      port (
6.          c : in integer; -- 4-bit input character code
7.          display : out std_logic_vector(7 downto 0) -- 7-segment output
8.      );
9.  end char_7seg;
10.
11. architecture behavior of char_7seg is
12. begin
13.     process(c)
14.     begin
15.         case c is
16.             when 0 => display <= "11000000"; -- 0
17.             when 1 => display <= "11111001"; -- 1
18.             when 2 => display <= "10100100"; -- 2
19.             when 3 => display <= "10110000"; -- 3
20.             when 4 => display <= "10011001"; -- 4
21.             when 5 => display <= "10010010"; -- 5
22.             when 6 => display <= "10000010"; -- 6
23.             when 7 => display <= "11111000"; -- 7
24.             when 8 => display <= "10000000"; -- 8
25.             when 9 => display <= "10010000"; -- 9
26.             when others => display <= "11111111"; -- Default to blank
27.         end case;
28.     end process;
29. end behavior;
```

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(7 downto 0)
9.   );
10. end part6;
11.
12. architecture behavior of part6 is
13.
```

```
 signal a, b, c, t, z : integer := 0;
 signal a_1, a_0, b_1, b_0 : integer range 9 downto 0 := 0;
 signal c_2, c_1, c_0, t_1, t_0, z_1, z_0 : integer := 0;
 signal s_0, s_1, s_2 : integer range 9 downto 0 := 0;

begin
 process(sw, a, b, c, t, z)
 begin
  a_1 <= to_integer(unsigned(sw(15 downto 12)));
  a_0 <=  to_integer(unsigned(sw(11 downto 8)));
  b_1 <= to_integer(unsigned(sw(7 downto 4)));
  b_0 <= to_integer(unsigned(sw(3 downto 0)));

  -- algorithm 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;

 -- a display --
 case a_1 is
  when 0 => hex7 <= "11111111";
  when 1 => hex7 <= "11111001";
  when 2 => hex7 <= "10100100";
  when 3 => hex7 <= "10110000";
  when 4 => hex7 <= "10011001";
  when 5 => hex7 <= "10010010";
  when 6 => hex7 <= "10000010";
  when 7 => hex7 <= "11111000";
  when 8 => hex7 <= "10000000";
  when 9 => hex7 <= "10010000";
 end case;

 case a_0 is
  when 0 => hex6 <= "11000000";
  when 1 => hex6 <= "11111001";
  when 2 => hex6 <= "10100100";
  when 3 => hex6 <= "10110000";
  when 4 => hex6 <= "10011001";
  when 5 => hex6 <= "10010010";
  when 6 => hex6 <= "10000010";
  when 7 => hex6 <= "11111000";
  when 8 => hex6 <= "10000000";
  when 9 => hex6 <= "10010000";
 end case;

 -- b display --

 case b_1 is
  when 0 => hex5 <= "11111111";
  when 1 => hex5 <= "11111001";
  when 2 => hex5 <= "10100100";
  when 3 => hex5 <= "10110000";
  when 4 => hex5 <= "10011001";
  when 5 => hex5 <= "10010010";
  when 6 => hex5 <= "10000010";
  when 7 => hex5 <= "11111000";
  when 8 => hex5 <= "10000000";
  when 9 => hex5 <= "10010000";
 end case;

 case b_0 is
```

```
    when 0 => hex4 <= "11000000";
    when 1 => hex4 <= "11111001";
    when 2 => hex4 <= "10100100";
    when 3 => hex4 <= "10110000";
    when 4 => hex4 <= "10011001";
    when 5 => hex4 <= "10010010";
     when 6 => hex4 <= "10000010";
     when 7 => hex4 <= "11111000";
     when 8 => hex4 <= "10000000";
     when 9 => hex4 <= "10010000";
   end case;

   -- sum display --

   case s_2 is
     when 0 => hex2 <= "11111111";
    when 1 => hex2 <= "11111001";
    when 2 => hex2 <= "10100100";
    when 3 => hex2 <= "10110000";
    when 4 => hex2 <= "10011001";
    when 5 => hex2 <= "10010010";
    when 6 => hex2 <= "10000010";
    when 7 => hex2 <= "11111000";
    when 8 => hex2 <= "10000000";
    when 9 => hex2 <= "10010000";
  end case;

   case s_1 is
     when 0 => hex1 <= "11000000";
     when 1 => hex1 <= "11111001";
     when 2 => hex1 <= "10100100";
     when 3 => hex1 <= "10110000";
     when 4 => hex1 <= "10011001";
     when 5 => hex1 <= "10010010";
     when 6 => hex1 <= "10000010";
     when 7 => hex1 <= "11111000";
     when 8 => hex1 <= "10000000";
     when 9 => hex1 <= "10010000";
   end case;

   case s_0 is
     when 0 => hex0 <= "11000000";
     when 1 => hex0 <= "11111001";
     when 2 => hex0 <= "10100100";
     when 3 => hex0 <= "10110000";
     when 4 => hex0 <= "10011001";
     when 5 => hex0 <= "10010010";
     when 6 => hex0 <= "10000010";
     when 7 => hex0 <= "11111000";
     when 8 => hex0 <= "10000000";
     when 9 => hex0 <= "10010000";
   end case;
 end process;
end behavior;
```

Part VII

Char_7seg the same as PartII

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity part7 is
    Port (
        binary_in : in STD_LOGIC_VECTOR(5 downto 0); -- 6-bit binary input
     hex1,hex0 : out STD_LOGIC_VECTOR(7 downto 0)
    );
end part7;

architecture TruthTable of part7 is

 -- Component declarations
    component char_7seg
        port (
            c : in std_logic_vector(3 downto 0);   -- 3-bit input for the character
            display : out std_logic_vector(7 downto 0) -- 7-segment output
        );
    end component;

 signal bcd_tens : STD_LOGIC_VECTOR(3 downto 0);
 signal bcd_ones  : STD_LOGIC_VECTOR(3 downto 0);
```

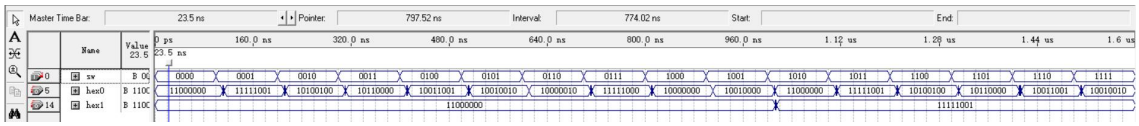
```
23.
24. begin
25.     -- Map binary input to BCD tens
    digit (MSB)
26.     with binary_in select
27.         bcd_tens <=
28.             "0000" when "000000" | "
    000001" | "000010" | "000011" |
29.                         "000100" | "0
    00101" | "000110" | "000111" |
30.                         "001000" | "0
    01001",
31.             "0001" when "001010" | "
    001011" | "001100" | "001101" |
32.                         "001110" | "0
    01111" | "010000" | "010001" |
33.                         "010010" | "0
    10011",
34.             "0010" when "010100" | "
    010101" | "010110" | "010111" |
35.                         "011000" | "0
    11001" | "011010" | "011011" |
36.                         "011100" | "0
    11101" ,
37.             "0011" when "011110" | "
    011111" | "100000" | "100001" |
38.       "100010" | "100011" | "100100"
     | "100101" |
39.       "100110" | "100111" ,
40.             "0100" when "101000" | "
    101001" | "101010" | "101011" |
41.       "101100" | "101101" | "101110"
     | "101111" |
42.       "110000" | "110001" ,
43.             "0101" when "110010" | "
    110011" | "110100" | "110101" |
44.       "110110" | "110111" | "111000"
     | "111001" |
45.       "111010" | "111011" ,
46.             "0110" when "111100" | "
    111101" | "111110" | "111111",
47.             "0000" when others;
48.
```

```
49.     -- Map binary input to BCD ones
    digit (LSB)
50.     with binary_in select
51.         bcd_ones <=
52.             "0000" when "000000" | "
    001010" | "010100" | "011110" |
53.                         "101000" | "1
    10010" | "111100",
54.             "0001" when "000001" | "
    001011" | "010101" | "011111" |
55.                         "101001" | "1
    10011" | "111101",
56.             "0010" when "000010" | "
    001100" | "010110" | "100000" |
57.                         "101010" | "1
    10100" | "111110",
58.             "0011" when "000011" | "
    001101" | "010111" | "100001" |
59.                         "101011" | "1
    10101" | "111111",
60.             "0100" when "000100" | "
    001110" | "011000" | "100010" |
61.                         "101100" | "1
    10110",
62.             "0101" when "000101" | "
    001111" | "011001" | "100011" |
63.                         "101101" | "1
    10111",
64.             "0110" when "000110" | "
    010000" | "011010" | "100100" |
65.                         "101110" | "1
    11000",
66.             "0111" when "000111" | "
    010001" | "011011" | "100101" |
67.                         "101111" | "1
    11001",
68.             "1000" when "001000" | "
    010010" | "011100" | "100110" |
69.                         "110000" | "1
    11010",
70.             "1001" when "001001" | "
    010011" | "011101" | "100111" |
71.                         "110001" | "1
    11011",
```

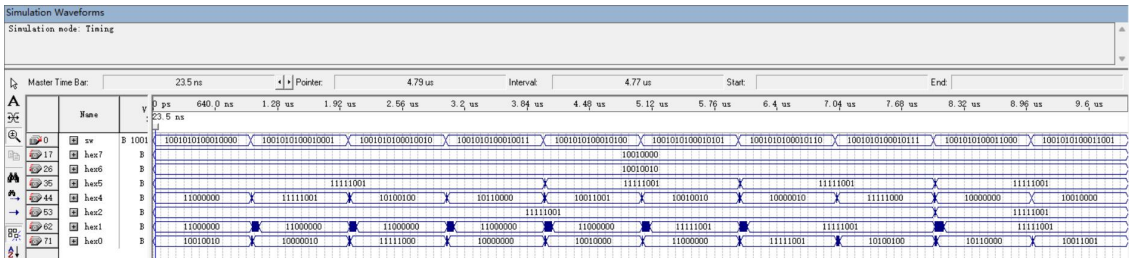
```
            "0000" when others;

    Hex_0: char_7seg
        port map (c => bcd_ones, dis
    play => hex0);

    Hex_1: char_7seg
        port map (c => bcd_tens, dis
    play => hex1);

end TruthTable;
```

4. Simulation and Synthesis Results

PartII

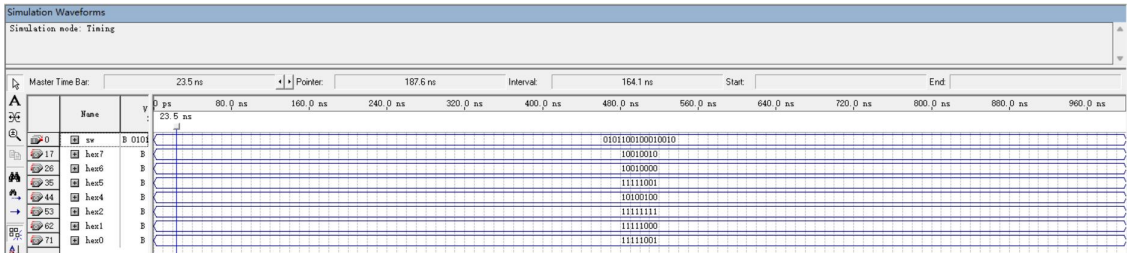


PartV



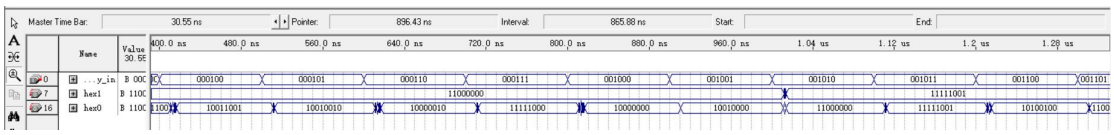
It works! Or at least I think it does...

Part VI



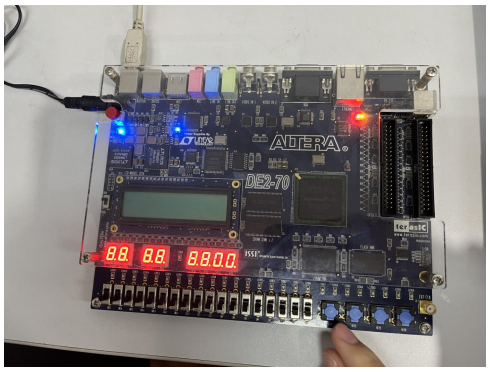
59 + 12 = 71

Part VII

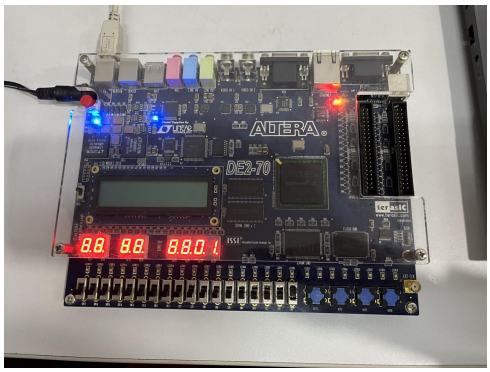


5. Experimental Results

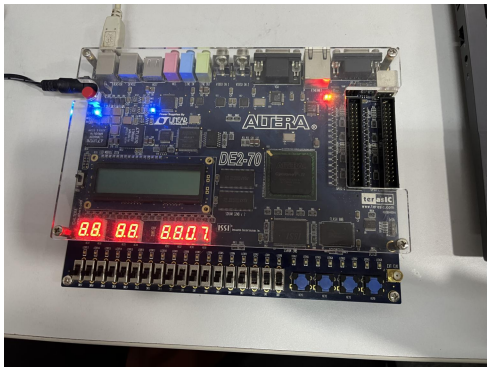
Part II



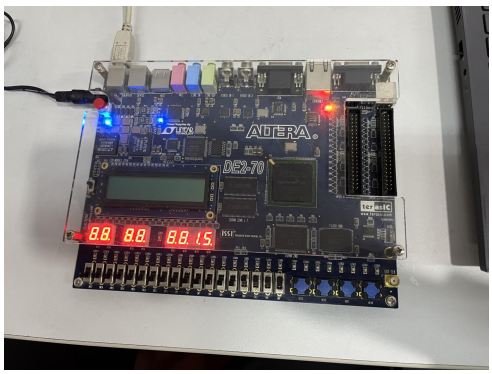
0000



0001

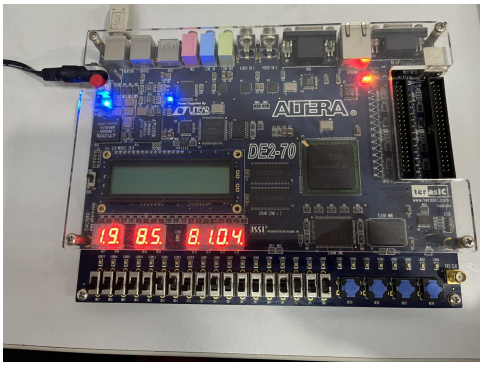


0111



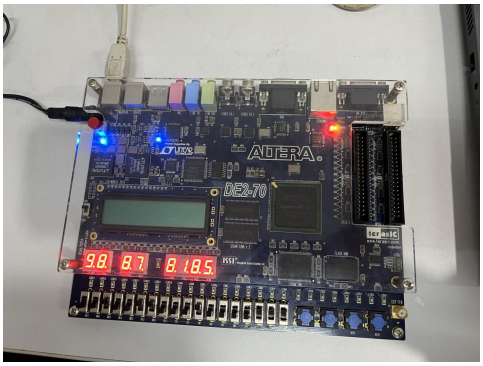
1111

Part V



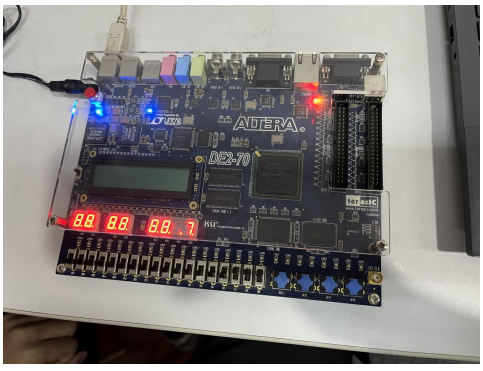
19 + 85 = 104

Part VI

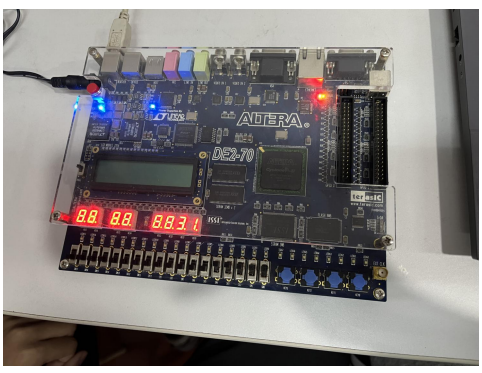


98 + 87 = 185

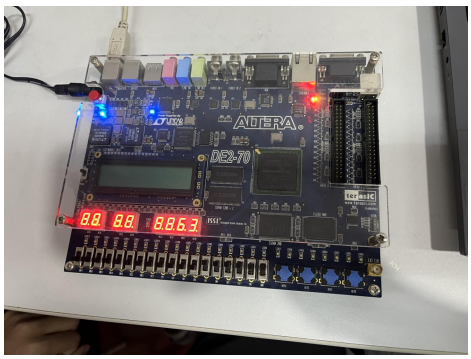
Part VII



000111



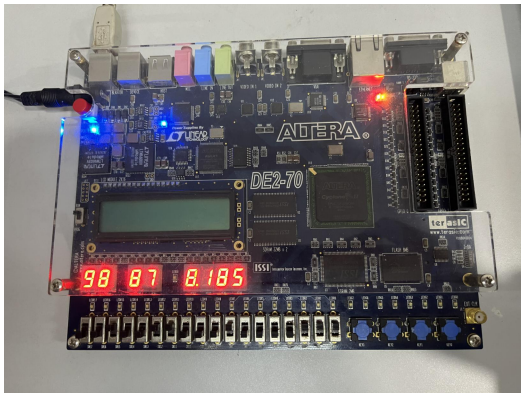
011111



111111

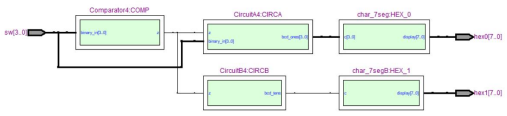
6. Discussion and Conclusion

Nauseous as the little red dot bottom right at the HEX display screen seems, upgrading the seven-segment code to eight would help, as the oHEX_DP bit is covered. Here's the one implemented.

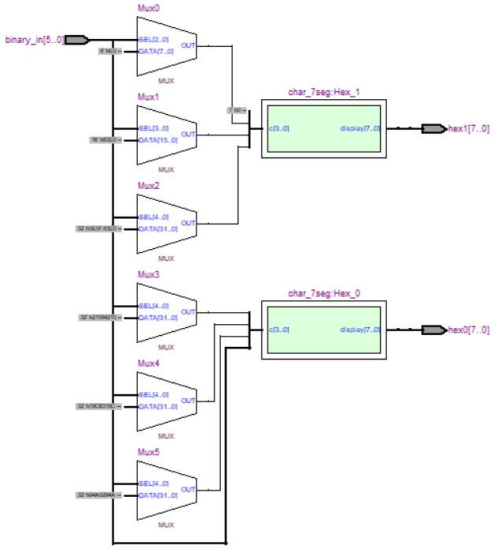


Looks a lot tidier and cleaner than usual. It would be a useful gag when floating-point arithmetic occurs.

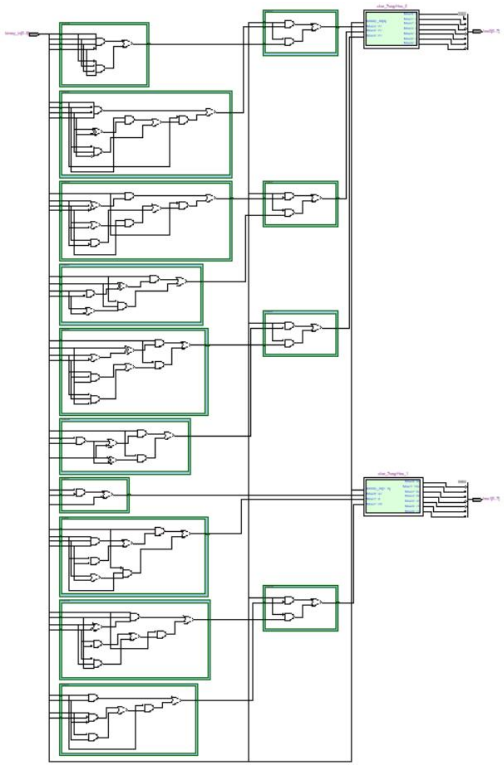
Also, there is inherently some difference between the structural implementation of 4 bit BCD converter and the purely boolean one, though not in the way that would affect hardware processing. One would be more graceful if structuring it closer to the first method for there would be naturally more portable and easily maintained.



The good...



The bad...



And the ugly. Both of them.