Programable Logic Circuts
Lab 01

Sølve Kjelseth

September 5, 2025

0. Introduction

This is the second report in this course, detailing the completion of the second lab exercise. All boolean expressions was found from given or self made truth tables and converted using this website. Note: The VHDL and LaTeX code is open source github link.

0.0. Table of Contents

0.1. List of Figures

1. Part 1

This tasks purpose was to use only structural/assignment statements to make a decoder for two 7 segment displays and display the decimal value of 0-9 on each based on switch inputs corresponding to a binary-coded-decimal (BCD) number. Inputs corresponding to decimal numbers 10-15 was treated as don't care as it was deemed invalid input. Interesting enough putting in decimal values 10 and above did still make some systematical sense as it treated all inputs as if the MSB was 0 instead of 1, example of this shown in figure 1.3d.

1.0. Solving

I started by making table 1.0 based on figure 1.0 and desired output. `0` as output will light up the segment and `1` will make it turn off, `*` represents don´t care and as those values are out of bounds they may be either `0` or `1`.

Table 1.0: Desired output from decoder

NR	SW				HEX						
	a	b	c	d	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	1	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	1	1	0	0
N/A	1	0	1	0	*	*	*	*	*	*	*
N/A	1	0	1	1	*	*	*	*	*	*	*
N/A	1	1	0	0	*	*	*	*	*	*	*
N/A	1	1	0	1	*	*	*	*	*	*	*
N/A	1	1	1	0	*	*	*	*	*	*	*
N/A	1	1	1	1	*	*	*	*	*	*	*

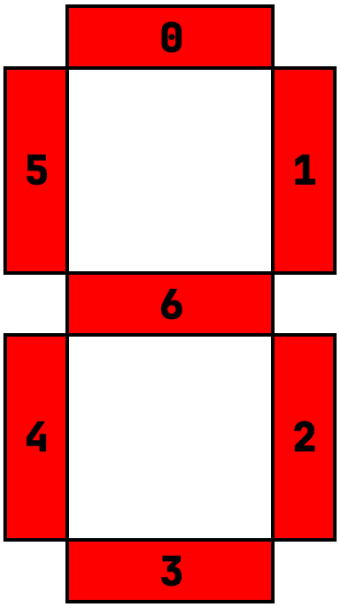


Figure 1.0: 7-segment display ports

Then solving the boolean algebra I got the following result in table 1.1. Formatting note: `!` before a letter means logical `NOT` and the operation only applies to the first letter directly after `!`, letters beside each other means logical `AND`, and `+` means logical `OR`. The expressions are always structured as SOP (Sum of products). `1.1.`

Table 1.1: Minimized solutions

HEX	Expression
0	!a!b!cd + b!d
1	b!cd + bc!d
2	!bc!d
3	!b!cd + b!c!d + bcd
4	b!c + d
5	!a!bd + !bc + cd
6	!a!b!c + bcd

1.1. Code

Code 1.1: TLE as described in the task

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

entity L02P01 is
    port(SW   : in  std_logic_vector(7 downto 0);
         HEX0 : out std_logic_vector(6 downto 0);
         HEX1 : out std_logic_vector(6 downto 0));
end L02P01;


architecture strucural of L02P01 is

    signal a0 : std_logic;
    signal b0 : std_logic;
    signal c0 : std_logic;
    signal d0 : std_logic;
    signal a1 : std_logic;
    signal b1 : std_logic;
    signal c1 : std_logic;
    signal d1 : std_logic;

begin

    a0 <= SW(3);
    b0 <= SW(2);
    c0 <= SW(1);
    d0 <= SW(0);
    a1 <= SW(7);
    b1 <= SW(6);
    c1 <= SW(5);
    d1 <= SW(4);

    HEX0(0) <= (((not a0) and (not b0) and (not c0) and d0) or
                (b0 and (not d0)));
    HEX0(1) <= ((b0 and (not c0) and d0) or
                (b0 and c0 and (not d0)));
    HEX0(2) <= ((not b0) and c0 and (not d0));
    HEX0(3) <= (((not b0) and (not c0) and d0) or
                (b0 and (not c0) and (not d0)) or
                (b0 and c0 and d0));
    HEX0(4) <= ((b0 and (not c0)) or d0);
    HEX0(5) <= (((not a0) and (not b0) and d0) or
                ((not b0) and c0) or
                (c0 and d0));
    HEX0(6) <= (((not a0) and (not b0) and (not c0)) or
                (b0 and c0 and d0));

    HEX1(0) <= (((not a1) and (not b1) and (not c1) and d1) or
                (b1 and (not d1)));
    HEX1(1) <= ((b1 and (not c1) and d1) or
                (b1 and c1 and (not d1)));
    HEX1(2) <= ((not b1) and c1 and (not d1));
    HEX1(3) <= (((not b1) and (not c1) and d1) or
                (b1 and (not c1) and (not d1)) or
                (b1 and c1 and d1));
```

```
56      HEX1(4) <= ((b1 and (not c1)) or d1);
57      HEX1(5) <= (((not a1) and (not b1) and d1) or
58                  ((not b1) and c1) or
59                  (c1 and d1));
60      HEX1(6) <= (((not a1) and (not b1) and (not c1)) or
61                  (b1 and c1 and d1));
62
63  end strucural;
```

1.2. RTL

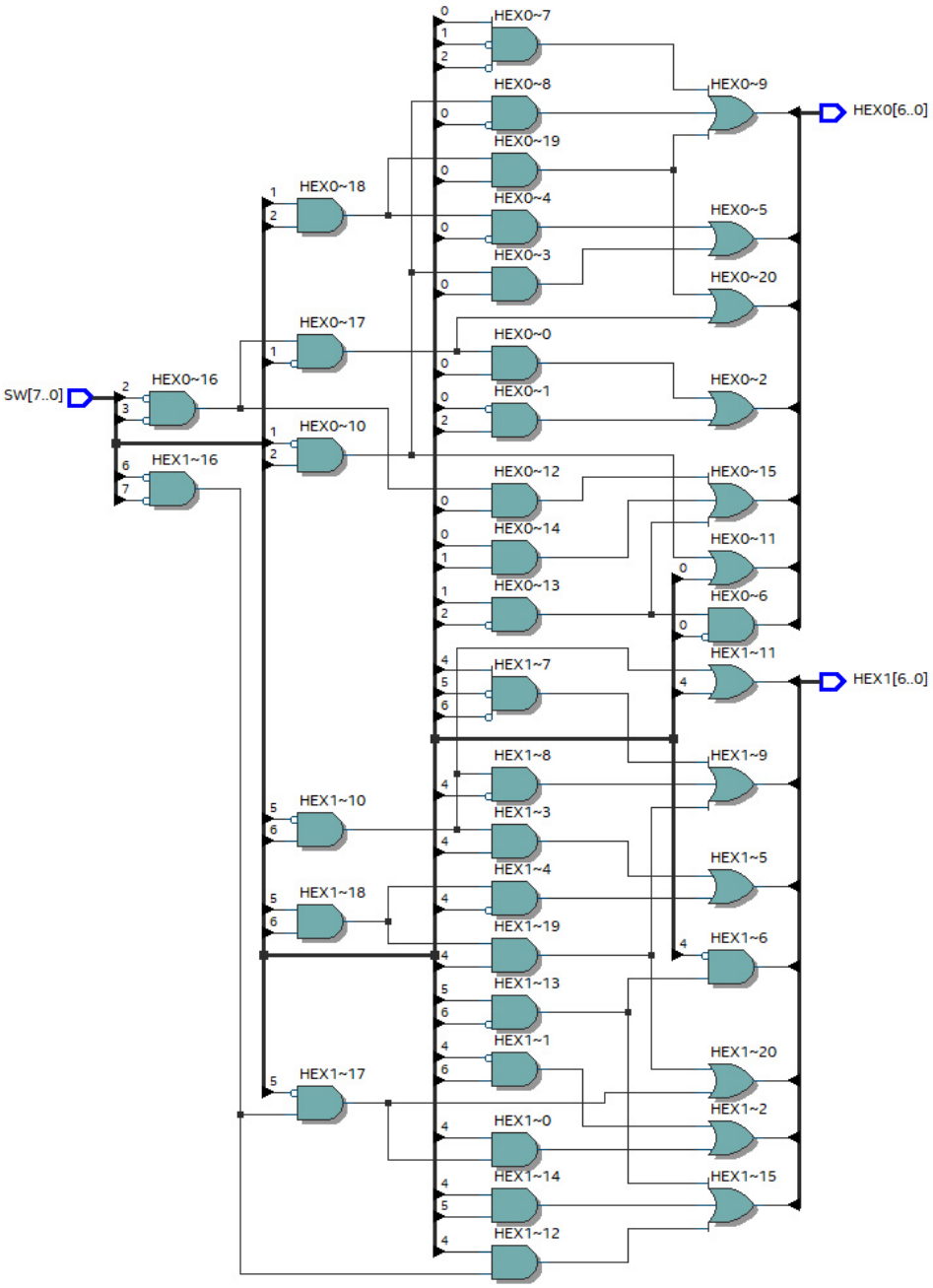
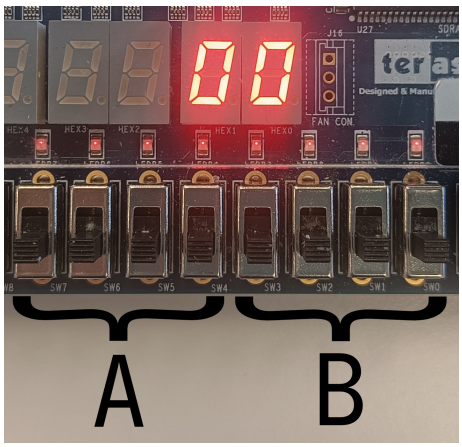
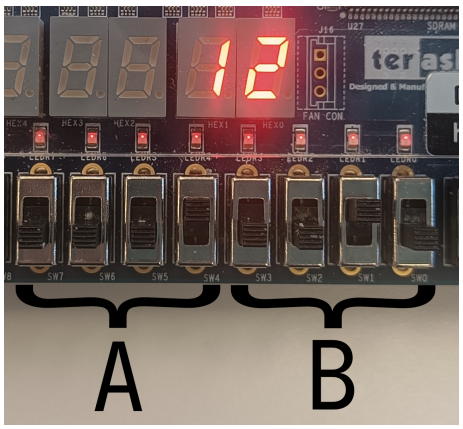


Figure 1.2: RTL synthesizing of the TLE

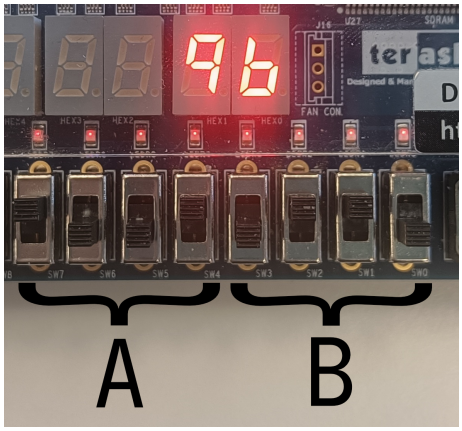
1.3. Results



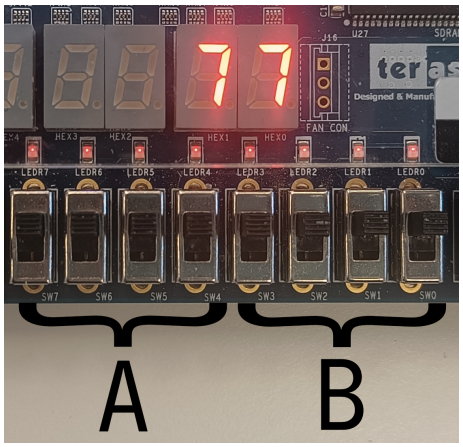
(a) BCD 0 0 as expected



(b) BCD 1 2 as expected



(c) BCD 9 6 as expected



(d) Out of bounds result

Figure 1.3: Test results

2. Part 2

Expanding on the BCD values from part 1, part 2 is about making a 4 bit input BCD number to appear as a two digit decimal number. As the task suggests, this means making a comparator that checks whether above 9 or not and sets the 10´s digit to 1 if above and 0 if not. At the same time this comparator can signify when the 1´s digit should just display the current BCD value and when it should be modified with a multiplexer. When 9 or below, the BCD should be displayed as is, and when above 9 it should modify the input such that it starts from 0 and goes to 5 for the numbers 10 to 15, this is marked circuit A, both in the tasks diagram and in my comment.

2.0. Solving

I started with making table 2.0 and table 2.1 using the mapping from table 2.2.

Table 2.0: Desired output from Z

NR	SW				Z			
	a	b	c	d	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	0	0	0
5	0	1	0	1	0	0	0	0
6	0	1	1	0	0	0	0	0
7	0	1	1	1	0	0	0	0
8	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0
10	1	0	1	0	0	0	0	1
11	1	0	1	1	0	0	0	1
12	1	1	0	0	0	0	0	1
13	1	1	0	1	0	0	0	1
14	1	1	1	0	0	0	0	1
15	1	1	1	1	0	0	0	1

Table 2.1: Desired output from A

NR	SW				A			
	a	b	c	d	3	2	1	0
0	0	0	0	0	*	*	*	*
1	0	0	0	1	*	*	*	*
2	0	0	1	0	*	*	*	*
3	0	0	1	1	*	*	*	*
4	0	1	0	0	*	*	*	*
5	0	1	0	1	*	*	*	*
6	0	1	1	0	*	*	*	*
7	0	1	1	1	*	*	*	*
8	1	0	0	0	*	*	*	*
9	1	0	0	1	*	*	*	*
10	1	0	1	0	0	0	0	0
11	1	0	1	1	0	0	0	1
12	1	1	0	0	0	0	1	0
13	1	1	0	1	0	0	1	1
14	1	1	1	0	0	1	0	0
15	1	1	1	1	0	1	0	1

Table 2.2: Switch mapping

a	SW(3)
b	SW(2)
c	SW(1)
d	SW(0)

Solving the boolean algebra, I found the following expressions.

Table 2.3: Minimized solutions for Z

Z	Expression
0	ac + ab
1	0
2	0
3	0

Table 2.4: Minimized solutions for A

A	Expression
0	d
1	!c
2	bc
3	0

2.1. Code

Code 2.0: TLE as described in the task

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

entity L02P02 is
    port(SW   : in  std_logic_vector(3 downto 0);
         HEX0 : out std_logic_vector(6 downto 0);
         HEX1 : out std_logic_vector(6 downto 0));
end L02P02;


architecture strucural of L02P02 is

    component decoder7seg
        port(BCD : in  std_logic_vector(3 downto 0);
             HEX : out std_logic_vector(6 downto 0));
    end component;

    component mux2to1
        port(s : in  std_logic;
             U : in  std_logic_vector(3 downto 0);
             V : in  std_logic_vector(3 downto 0);
             M : out std_logic_vector(3 downto 0));
    end component;

    signal d0 : std_logic_vector(3 downto 0);
    signal Z  : std_logic_vector(3 downto 0);
    signal A  : std_logic_vector(3 downto 0);

begin

    -- Comparator number is 4 bit because of decoder
    Z(0) <= ((SW(3) and SW(1)) or (SW(3) and SW(2)));
    Z(3 downto 1) <= "000";

    -- Circut A to convert to correct zeroth digit
    A(0) <= SW(0);
    A(1) <= (not SW(1));
    A(2) <= (SW(2) and SW(1));
    A(3) <= '0';

    M0: mux2to1 port map
        (s => z(0),
         U => SW,
         V => A,
         M => d0);

    H0: decoder7seg port map
        (BCD => d0,
         HEX => HEX0);

    H1: decoder7seg port map
        (BCD => Z,
         HEX => HEX1);

end strucural;
```

Code 2.1: Multiplexer used in TLE

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

entity mux2to1 is
    port(s : in  std_logic;
         U : in  std_logic_vector(3 downto 0);
         V : in  std_logic_vector(3 downto 0);
         M : out std_logic_vector(3 downto 0));
end mux2to1;

architecture strucural of mux2to1 is

begin

    M(0) <= (((not s) and U(0)) or (s and  V(0)));
    M(1) <= (((not s) and U(1)) or (s and  V(1)));
    M(2) <= (((not s) and U(2)) or (s and  V(2)));
    M(3) <= (((not s) and U(3)) or (s and  V(3)));

end strucural;
```

Code 2.2: 7-seg decoder used in TLE

VHDL

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

entity decoder7seg is
    port(BCD : in  std_logic_vector(3 downto 0);
         HEX : out std_logic_vector(6 downto 0));
end decoder7seg;

architecture strucural of decoder7seg is

    signal a : std_logic;
    signal b : std_logic;
    signal c : std_logic;
    signal d : std_logic;

begin

    a <= BCD(3);
    b <= BCD(2);
    c <= BCD(1);
    d <= BCD(0);

    HEX(0) <= (((not a) and (not b) and (not c) and d) or
               (b and (not d)));
    HEX(1) <= ((b and (not c) and d) or
               (b and c and (not d)));
    HEX(2) <= ((not b) and c and (not d));
    HEX(3) <= (((not b) and (not c) and d) or
               (b and (not c) and (not d)) or
               (b and c and d));
    HEX(4) <= ((b and (not c)) or d);
    HEX(5) <= (((not a) and (not b) and d) or
               ((not b) and c) or
               (c and d));
    HEX(6) <= (((not a) and (not b) and (not c)) or
               (b and c and d));

end strucural;
```

2.2. RTL

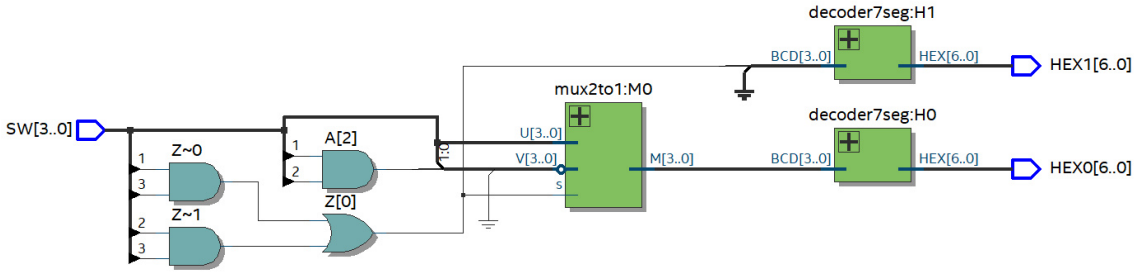


Figure 2.3: RTL synthesizing of the TLE

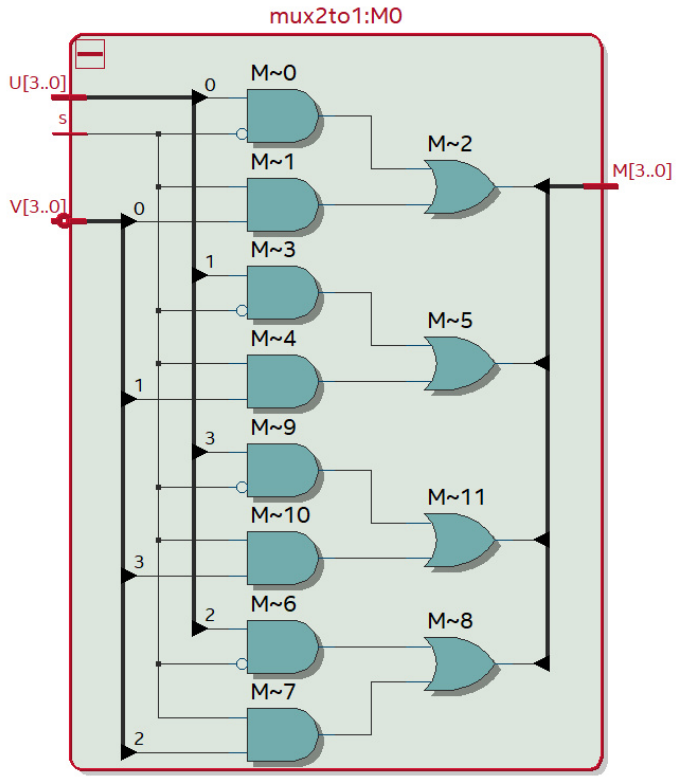


Figure 2.4: RTL synthesizing of the multiplexer

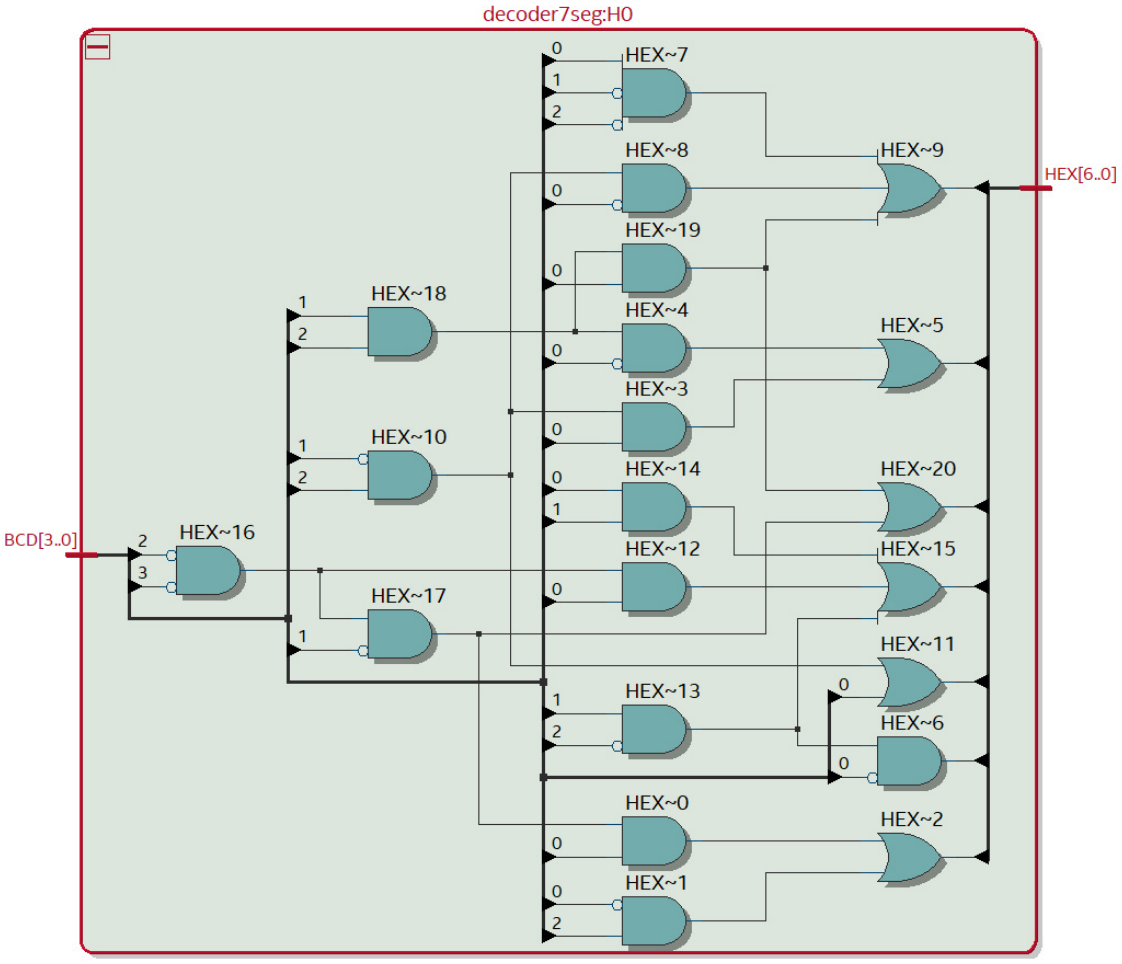
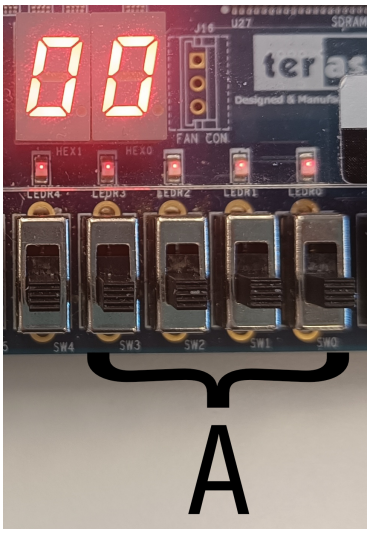
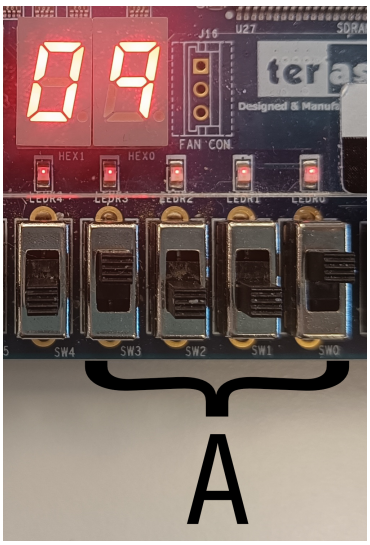


Figure 2.5: RTL synthesizing of the 7-seg decoder

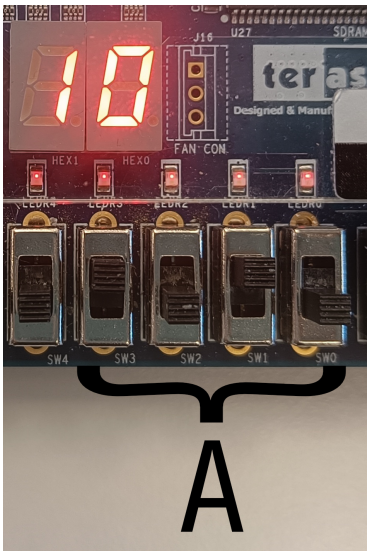
2.3. Results



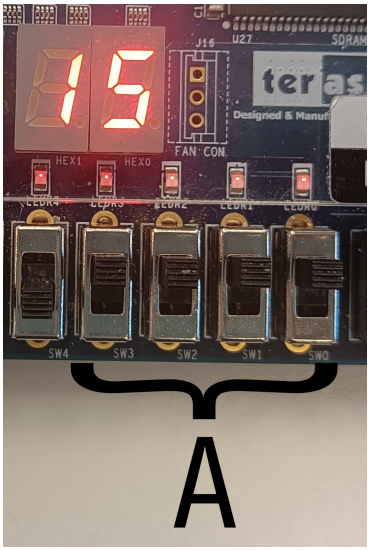
(a) BCD 0 as expected



(b) BCD 9 as expected



(c) BCD 10 as expected



(d) BCD 15 as expected

Figure 2.6: Test results

3. Part 3

This parts is currently unrelated to the two previous parts as here we are making a 4-bit adder using 4 instances of a single bit adder. This task was probably designed to teach the use of multiple instances and how to import other code snippets using components. As I already have done that multiple times for readability and reusability this was a very simple task.

3.0. Solving

As opposed to the previous tasks, this time I used the provided table 3.0 for the single bit adder and found the expressions shown in table 3.1.

Table 3.0: Provided table for adder

In			Out	
Ci	B	A	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 3.1: Minimized solutions

Out	Expression
S	!Ci!BA + !CiB!A + Ci!B!A + CiBA
Co	BA + CiA + CiB

3.1. Code

Code 3.0: TLE as described in the task

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

entity L02P03 is
    port(SW   : in  std_logic_vector(8 downto 0);
         LEDR : out std_logic_vector(4 downto 0));
end L02P03;


architecture strucural of L02P03 is

    component fullAdder
        port(Ci : in  std_logic;
             A  : in  std_logic;
             B  : in  std_logic;
             S  : out std_logic;
             Co : out std_logic);
    end component;

    signal C : std_logic_vector(2 downto 0);

begin

    A0: fullAdder port map
        (Ci => SW(8),
         A => SW(0),
         B => SW(4),
         S => LEDR(0),
         Co => C(0));

    A1: fullAdder port map
        (Ci => C(0),
         A => SW(1),
         B => SW(5),
         S => LEDR(1),
         Co => C(1));

    A2: fullAdder port map
        (Ci => C(1),
         A => SW(2),
         B => SW(6),
         S => LEDR(2),
         Co => C(2));

    A3: fullAdder port map
        (Ci => C(2),
         A => SW(3),
         B => SW(7),
         S => LEDR(3),
         Co => LEDR(4));

end strucural;
```

Code 3.1: Single bit adder used in TLE

VHDL

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

entity fullAdder is
    port(Ci : in  std_logic;
         A  : in  std_logic;
         B  : in  std_logic;
         S  : out std_logic;
         Co : out std_logic);
end fullAdder;

architecture strucural of fullAdder is

begin

    S  <= (((not Ci) and (not B) and A) or
          ((not Ci) and B and (not A)) or
          (Ci and (not B) and (not A)) or
          (Ci and B and A));

    Co <= ((B and A) or
          (Ci and A) or
          (Ci and B));

end strucural;
```

3.2. RTL

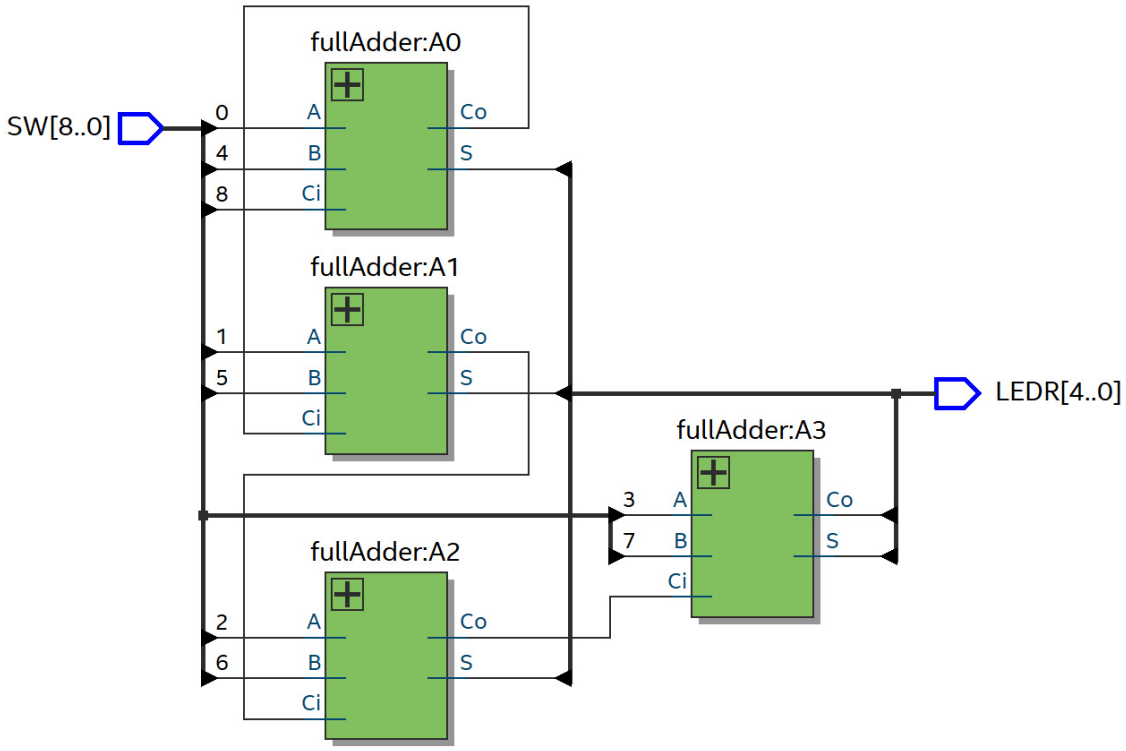


Figure 3.2: RTL synthesizing of the TLE

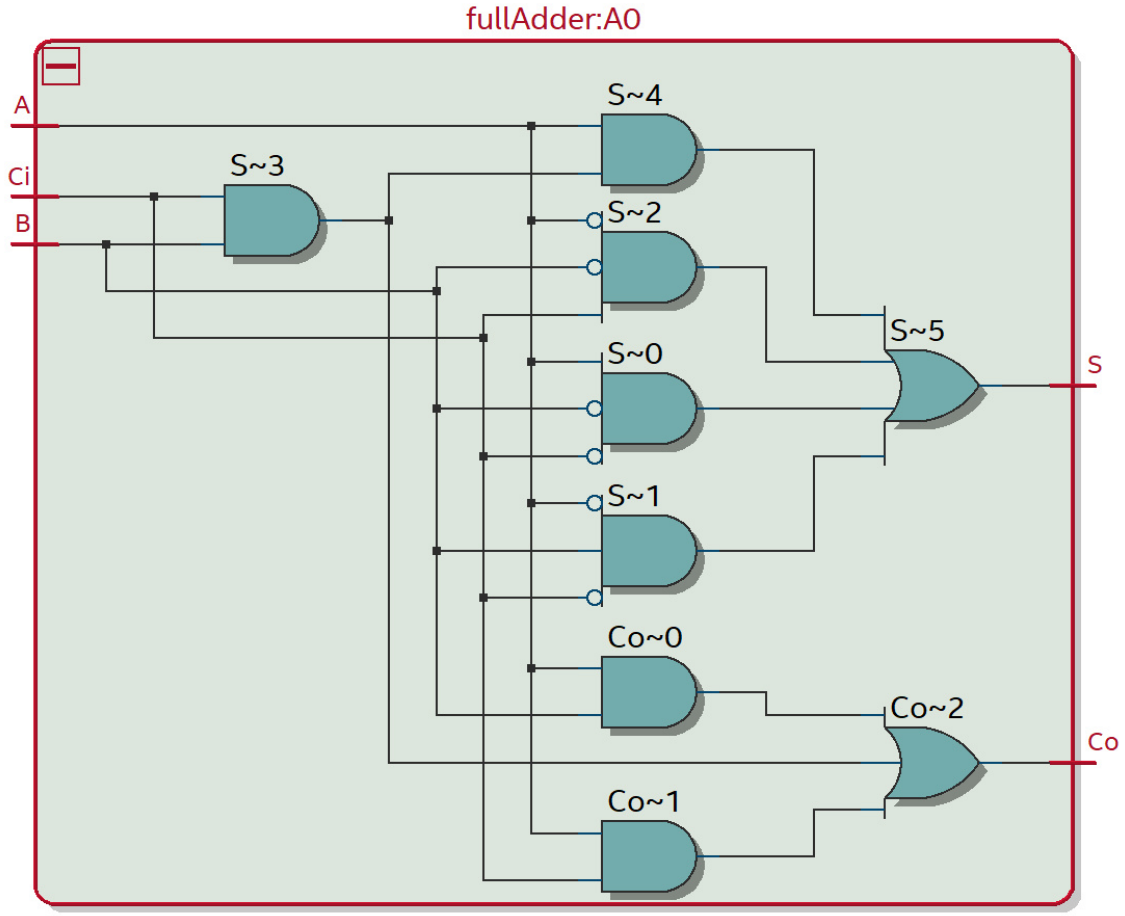
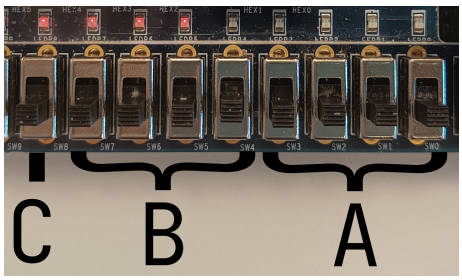
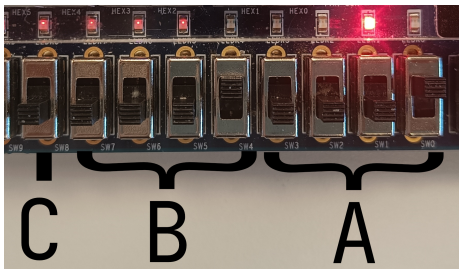


Figure 3.3: RTL synthesizing of a single bit adder instance

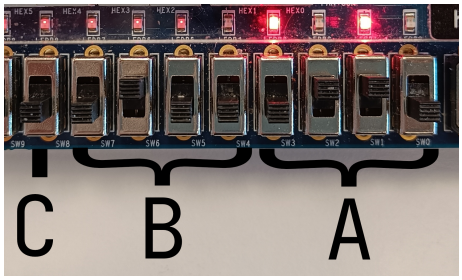
3.3. Results



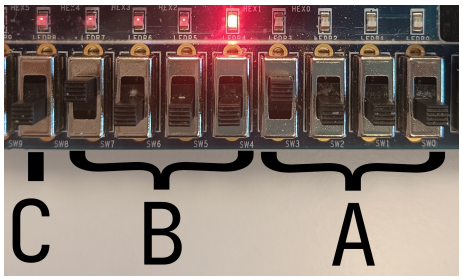
(a) 0 + 0 + 0 = 0



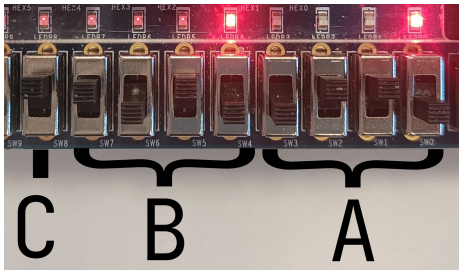
(b) 0 + 1 + 1 = 2



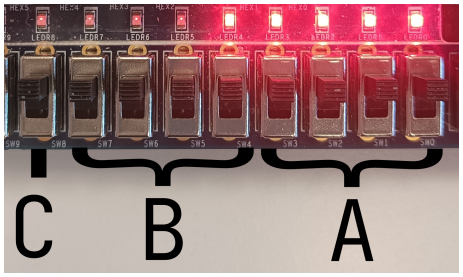
(c) 0 + 4 + 6 = 10



(d) 0 + 8 + 8 = 16



(e) 1 + 10 + 6 = 17



(f) 1 + 15 + 15 = 31

Figure 3.4: Test results

4. Part 4

This part sort of combines part 2 and 3 with two 4 bit inputs and carry in and displays the result as a two digit decimal number. I used the same comparator method as in task 2, but extended for number up to 19 to check wether number is above 9. Then I modified the A circuit, renaming it T for transform to not mix up with the adder inputs and modifying it to convert 10-19 to 0-9.

4.0. Code

Code 4.0: TLE as described in the task

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

entity L02P04 is
    port(SW   : in  std_logic_vector(8 downto 0);
         LEDR : out std_logic_vector(9 downto 0);
         HEX0 : out std_logic_vector(6 downto 0);
         HEX1 : out std_logic_vector(6 downto 0);
         HEX3 : out std_logic_vector(6 downto 0);
         HEX5 : out std_logic_vector(6 downto 0));
end L02P04;


architecture strucural of L02P04 is

    component adder4bit
        port(Ci : in  std_logic;
             a  : in  std_logic_vector(3 downto 0);
             b  : in  std_logic_vector(3 downto 0);
             S  : out std_logic_vector(3 downto 0);
             Co : out std_logic);
    end component;

    component mux2to1
        port(Sel   : in  std_logic;
             In_A  : in  std_logic_vector(3 downto 0);
             In_B  : in  std_logic_vector(3 downto 0);
             Out_M : out std_logic_vector(3 downto 0));
    end component;

    component decoder7seg
        port(BCD : in  std_logic_vector(3 downto 0);
             HEX : out std_logic_vector(6 downto 0));
    end component;

    signal Sum : std_logic_vector(4 downto 0);
    signal S0  : std_logic_vector(3 downto 0);
    signal S1  : std_logic_vector(3 downto 0);
    signal X   : std_logic_vector(3 downto 0);
    signal Y   : std_logic_vector(3 downto 0);
    signal T   : std_logic_vector(3 downto 0);

begin

    X <= SW(7 downto 4);
```

```
46      Y <= SW(3 downto 0);
47      LEDR(8 downto 5) <= "0000";
48      LEDR(4 downto 0) <= Sum;
49
50      -- Checking whether X or Y is above 9
51      LEDR(9) <= ((X(3) and X(1)) or (X(3) and X(2)) or
52                  (Y(3) and Y(1)) or (Y(3) and Y(2)));
53
54      -- Checking whether Sum is above 9
55      S1(0) <= ((Sum(3) and Sum(1)) or (Sum(3) and Sum(2)) or sum(4));
56      S1(3 downto 1) <= "000"; -- S1 only displays 0 or 1
57
58      -- Value to pass to S1 when above 9
59      T(0) <= Sum(0);
60      T(1) <= (not Sum(1));
61      T(2) <= ((Sum(2) and Sum(1)) or (Sum(4) and (not Sum(1))));
62      T(3) <= (Sum(4) and Sum(1));
63
64      A0: adder4bit port map
65          (Ci => SW(8),
66           a => SW(3 downto 0),
67           b => SW(7 downto 4),
68           S => Sum(3 downto 0),
69           Co => Sum(4));
70
71      M0: mux2to1 port map
72          (Sel => S1(0),
73           In_A => Sum(3 downto 0),
74           In_B => T,
75           Out_M => S0);
76
77      H0: decoder7seg port map
78          (BCD => S0,
79           HEX => HEX0);
80
81      H1: decoder7seg port map
82          (BCD => S1,
83           HEX => HEX1);
84
85      H3: decoder7seg port map
86          (BCD => Y,
87           HEX => HEX3);
88
89      H5: decoder7seg port map
90          (BCD => X,
91           HEX => HEX5);
92
93  end strucural;
```

Code 4.1: 4-bit adder used in TLE

VHDL

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

entity adder4bit is
    port(Ci : in  std_logic;
         a  : in  std_logic_vector(3 downto 0);
         b  : in  std_logic_vector(3 downto 0);
         S  : out std_logic_vector(3 downto 0);
         Co : out std_logic);
end adder4bit;


architecture strucural of adder4bit is

    component fullAdder
        port(C_in  : in  std_logic;
             U     : in  std_logic;
             V     : in  std_logic;
             S_out : out std_logic;
             C_out : out std_logic);
    end component;

    signal C : std_logic_vector(2 downto 0);

begin

    A0: fullAdder port map
        (C_in => Ci,
         U => a(0),
         V => b(0),
         S_out => S(0),
         C_out => C(0));

    A1: fullAdder port map
        (C_in => C(0),
         U => a(1),
         V => b(1),
         S_out => S(1),
         C_out => C(1));

    A2: fullAdder port map
        (C_in => C(1),
         U => a(2),
         V => b(2),
         S_out => S(2),
         C_out => C(2));

    A3: fullAdder port map
        (C_in => C(2),
         U => a(3),
         V => b(3),
         S_out => S(3),
         C_out => Co);

end strucural;
```

Code 4.2: Single bit adder used in 4-bit adder

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

entity fullAdder is
    port(C_in : in  std_logic;
         U  : in  std_logic;
         V  : in  std_logic;
         S_out  : out std_logic;
         C_out : out std_logic);
end fullAdder;

architecture strucural of fullAdder is

begin

    S_out <= (((not C_in) and (not V) and U) or
              ((not C_in) and V and (not U)) or
              (C_in and (not V) and (not U)) or
              (C_in and V and U));

    C_out <= ((V and U) or
              (C_in and U) or
              (C_in and V));

end strucural;
```

Code 4.3: Multiplexer used in TLE

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

entity mux2to1 is
    port(Sel   : in  std_logic;
         In_A  : in  std_logic_vector(3 downto 0);
         In_B  : in  std_logic_vector(3 downto 0);
         Out_M : out std_logic_vector(3 downto 0));
end mux2to1;

architecture strucural of mux2to1 is

begin

    Out_M(0) <= (((not Sel) and In_A(0)) or
                 (Sel and  In_B(0)));
    Out_M(1) <= (((not Sel) and In_A(1)) or
                 (Sel and  In_B(1)));
    Out_M(2) <= (((not Sel) and In_A(2)) or
                 (Sel and  In_B(2)));
    Out_M(3) <= (((not Sel) and In_A(3)) or
                 (Sel and  In_B(3)));

end strucural;
```

Code 4.4: 7-seg decoder used in TLE

VHDL

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

entity decoder7seg is
    port(BCD : in  std_logic_vector(3 downto 0);
         HEX : out std_logic_vector(6 downto 0));
end decoder7seg;

architecture strucural of decoder7seg is

    signal a : std_logic;
    signal b : std_logic;
    signal c : std_logic;
    signal d : std_logic;

begin

    a <= BCD(3);
    b <= BCD(2);
    c <= BCD(1);
    d <= BCD(0);

    HEX(0) <= (((not a) and (not b) and (not c) and d) or
               (b and (not d)));
    HEX(1) <= ((b and (not c) and d) or
               (b and c and (not d)));
    HEX(2) <= ((not b) and c and (not d));
    HEX(3) <= (((not b) and (not c) and d) or
               (b and (not c) and (not d)) or (b and c and d));
    HEX(4) <= ((b and (not c)) or d);
    HEX(5) <= (((not a) and (not b) and d) or
               ((not b) and c) or (c and d));
    HEX(6) <= (((not a) and (not b) and (not c)) or
               (b and c and d));

end strucural;
```

4.1. RTL

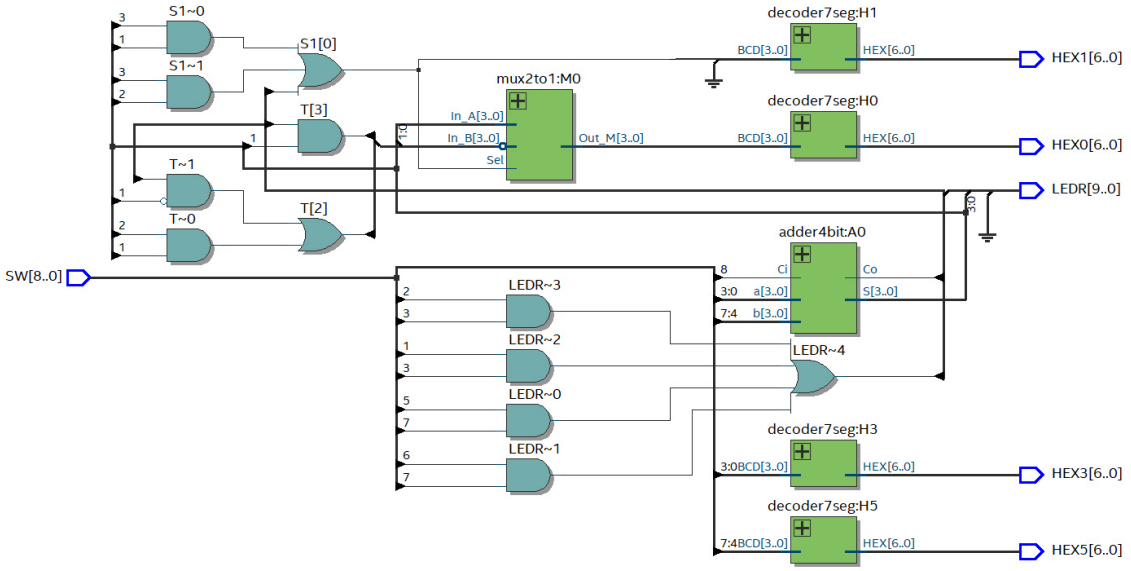


Figure 4.5: RTL synthesizing of the TLE

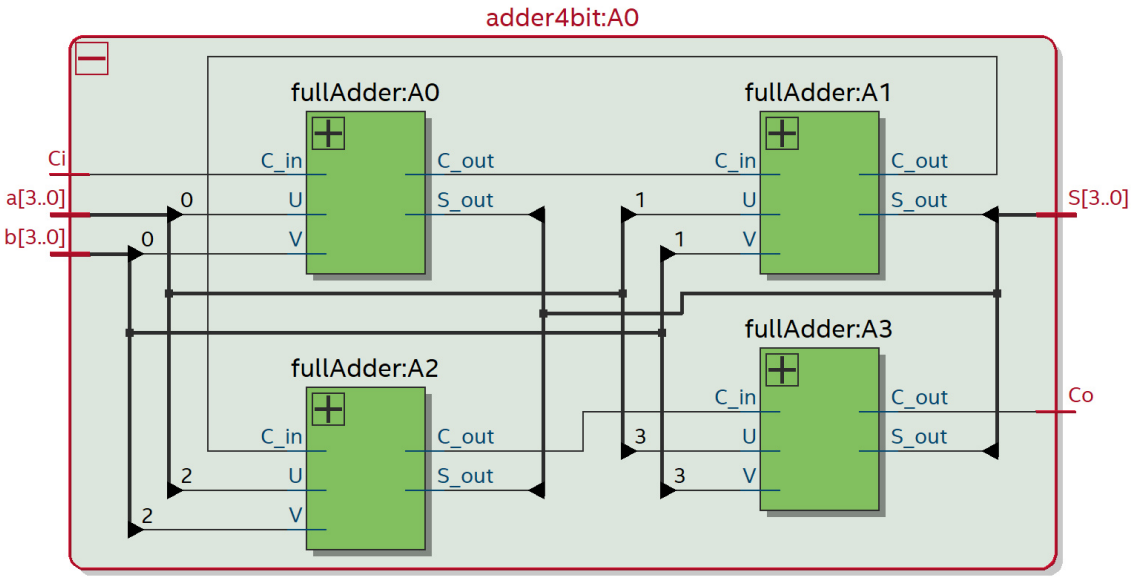


Figure 4.6: RTL synthesizing of the 4 bit adder

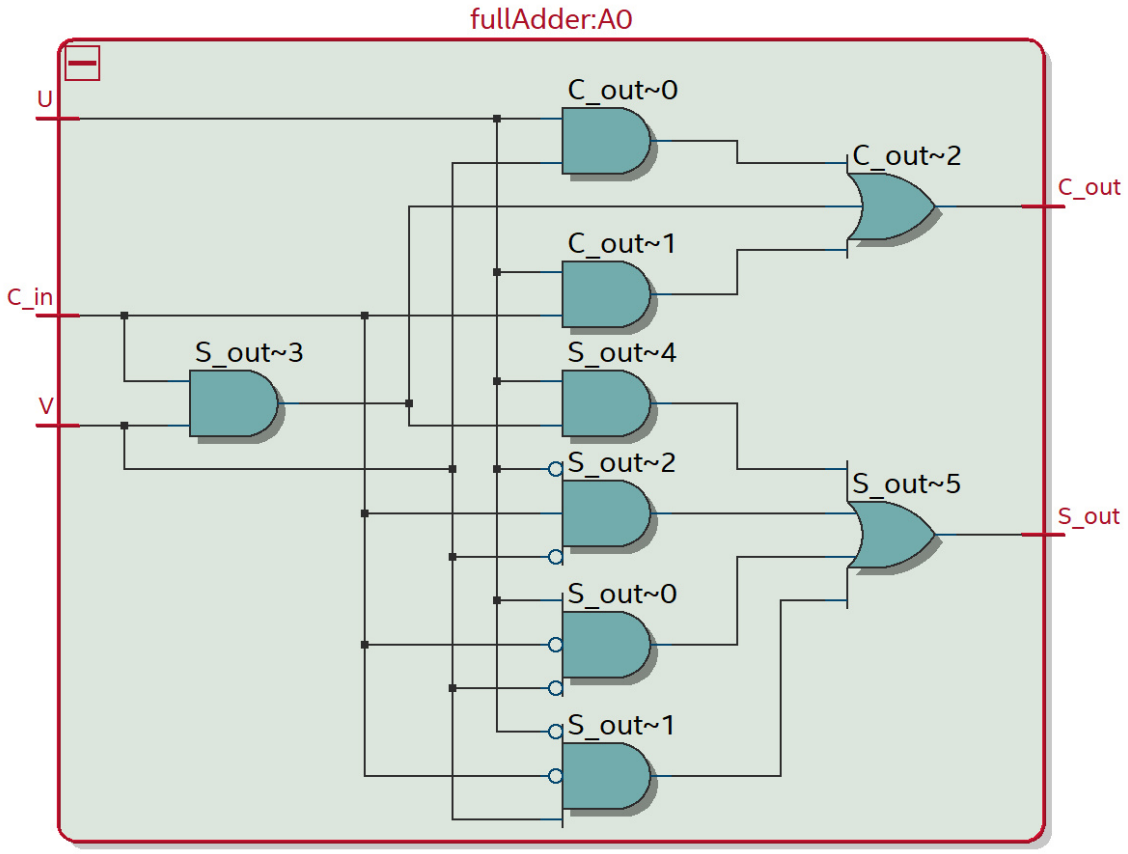


Figure 4.7: RTL synthesizing of a single bit adder instance

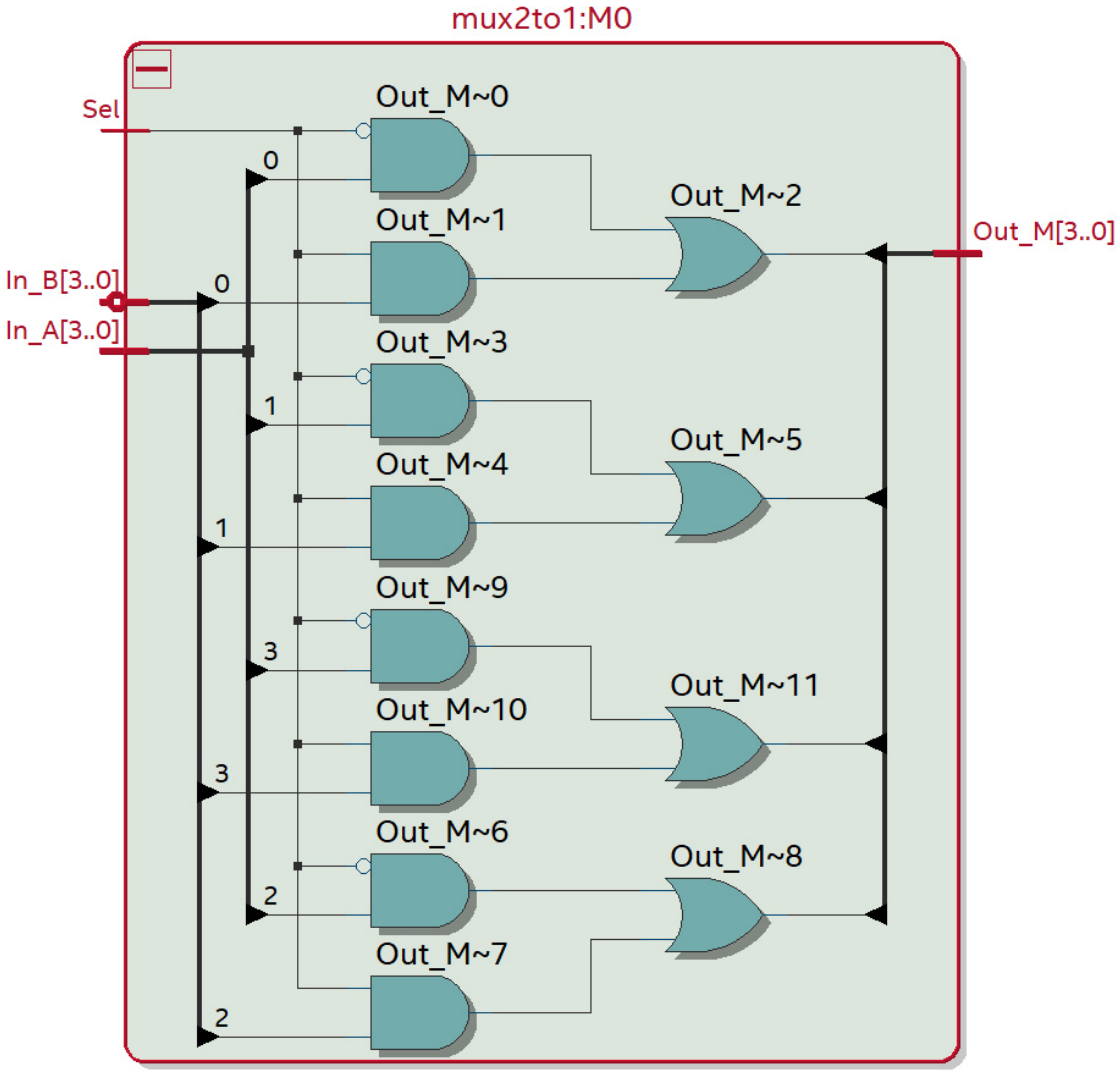


Figure 4.8: RTL synthesizing of the multiplexer

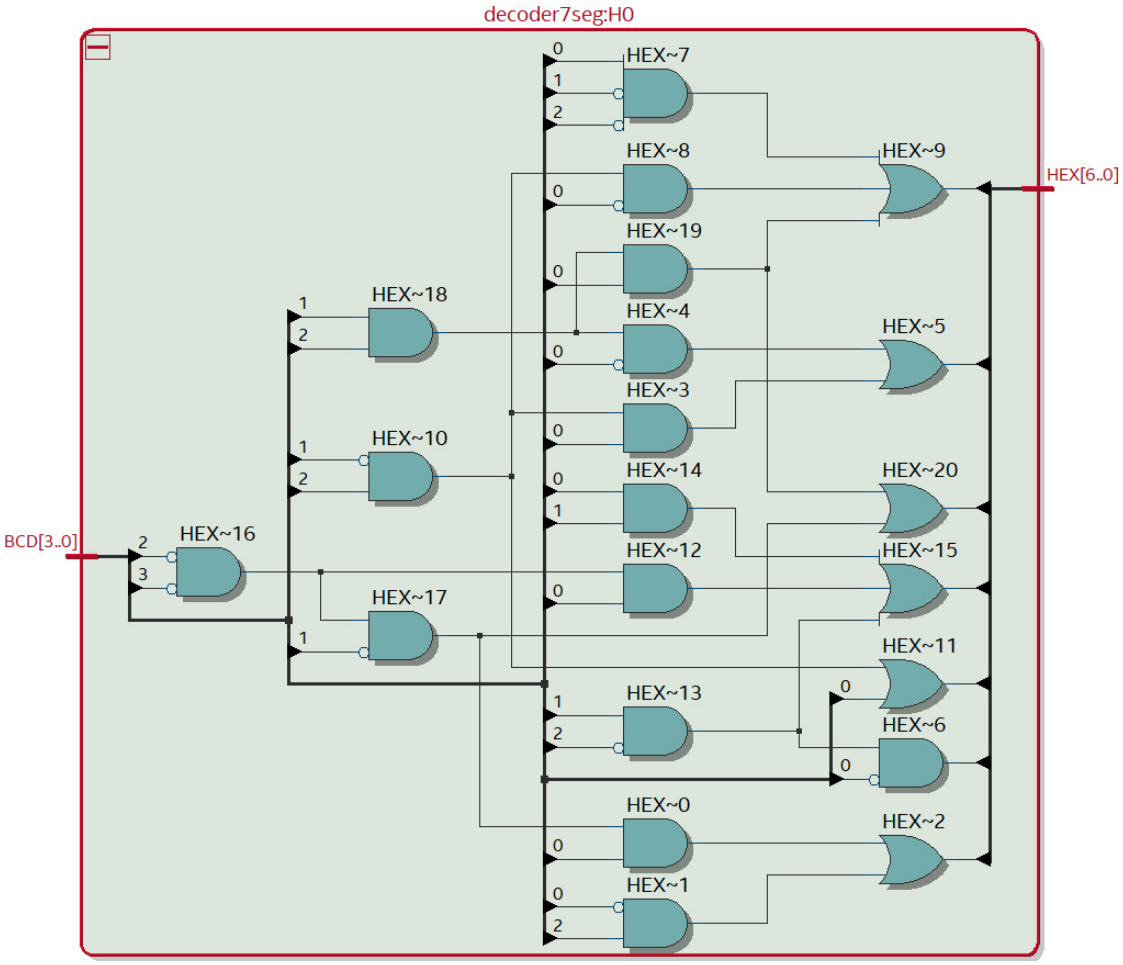
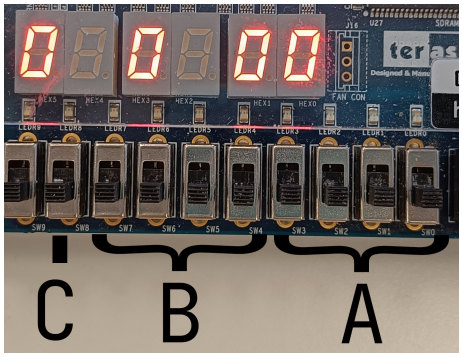
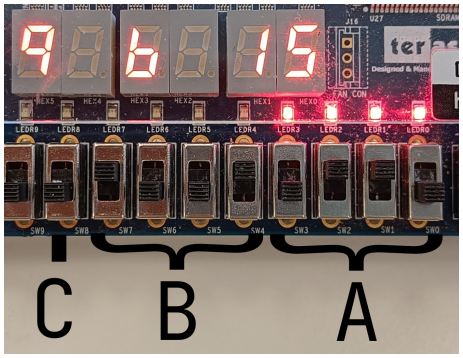


Figure 4.9: RTL synthesizing of a 7-segment decoder instance

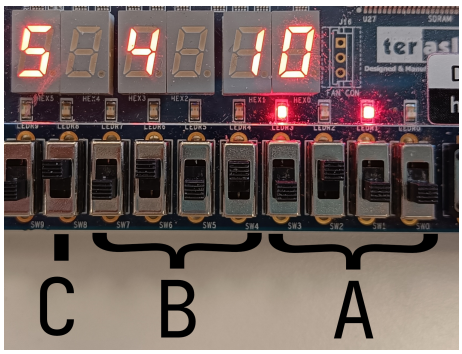
4.2. Results



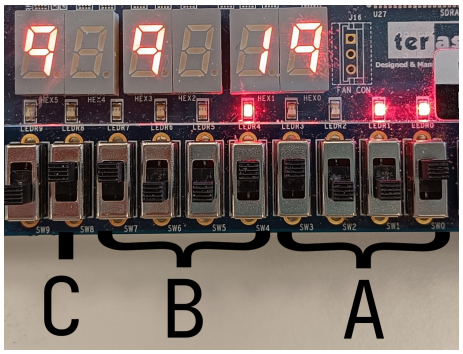
(a) 0 + 0 + 0 = 0



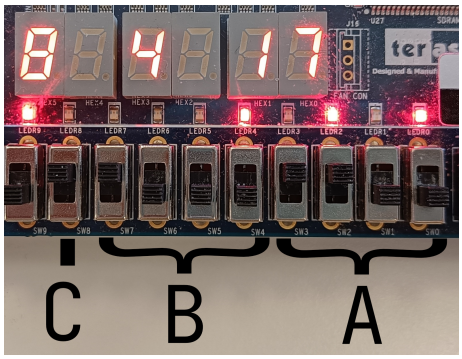
(b) 0 + 9 + 6 = 15



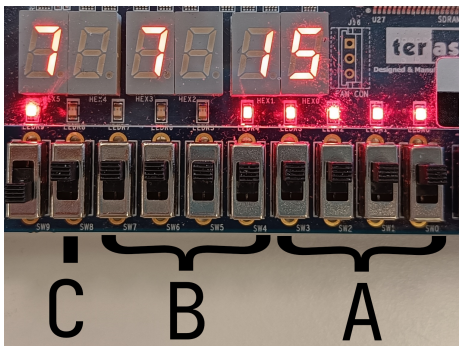
(c) 1 + 5 + 4 = 10



(d) 1 + 9 + 9 = 19



(e) Out of bounds



(f) Out of bounds

Figure 4.10: Test results

5. Part 5

5.0. Code

Code 5.0: TLE as described in the task

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity L02P05 is
    port(SW   : in  std_logic_vector(8 downto 0);
         HEX0 : out std_logic_vector(6 downto 0);
         HEX1 : out std_logic_vector(6 downto 0);
         HEX3 : out std_logic_vector(6 downto 0);
         HEX5 : out std_logic_vector(6 downto 0));
end L02P05;


architecture behavioural of L02P05 is

    component decoder7seg
        port(BCD : in  std_logic_vector(3 downto 0);
             HEX : out std_logic_vector(6 downto 0));
    end component;

    signal A   : std_logic_vector(3 downto 0);
    signal B   : std_logic_vector(3 downto 0);
    signal Cin : std_logic;
    signal Sum : std_logic_vector(4 downto 0);
    signal St  : std_logic_vector(3 downto 0);
    signal S0  : std_logic_vector(4 downto 0);
    signal S1  : std_logic_vector(3 downto 0);



begin
    process(A, B, Cin)
    begin
        A <= SW(7 downto 4);
        B <= SW(3 downto 0);
        Cin <= SW(8);

        Sum <= ('0' & A) + ('0' & B) + Cin;
        if Sum > "01001" then
            S0 <= Sum - "01010";
            S1 <= "0001";
        else
            S0 <= Sum(4 downto 0);
            S1 <= "0000";
        end if;
    end process;

    H0: decoder7seg port map
        (BCD => S0(3 downto 0),
         HEX => HEX0);

    H1: decoder7seg port map
```

```
54          (BCD => S1,
55           HEX => HEX1);
56
57      H3: decoder7seg port map
58          (BCD => B,
59           HEX => HEX3);
60
61      H5: decoder7seg port map
62          (BCD => A,
63           HEX => HEX5);
64
65  end behavioural;
```

Code 5.1: 7-seg decoder used in TLE

```
VHDL
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity decoder7seg is
5       port(BCD : in  std_logic_vector(3 downto 0);
6            HEX : out std_logic_vector(6 downto 0));
7   end decoder7seg;
8
9   architecture strucural of decoder7seg is
10
11      signal a : std_logic;
12      signal b : std_logic;
13      signal c : std_logic;
14      signal d : std_logic;
15
16  begin
17
18      a <= BCD(3);
19      b <= BCD(2);
20      c <= BCD(1);
21      d <= BCD(0);
22
23      HEX(0) <= (((not a) and (not b) and (not c) and d) or
24                 (b and (not d)));
25      HEX(1) <= ((b and (not c) and d) or
26                 (b and c and (not d)));
27      HEX(2) <= ((not b) and c and (not d));
28      HEX(3) <= (((not b) and (not c) and d) or
29                 (b and (not c) and (not d)) or (b and c and d));
30      HEX(4) <= ((b and (not c)) or d);
31      HEX(5) <= (((not a) and (not b) and d) or
32                 ((not b) and c) or (c and d));
33      HEX(6) <= (((not a) and (not b) and (not c)) or
34                 (b and c and d));
35
36  end strucural;
```

5.1. RTL

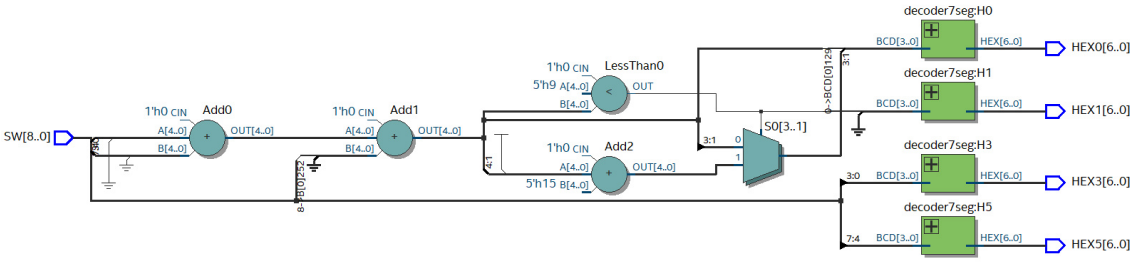


Figure 5.2: RTL synthesizing of the TLE

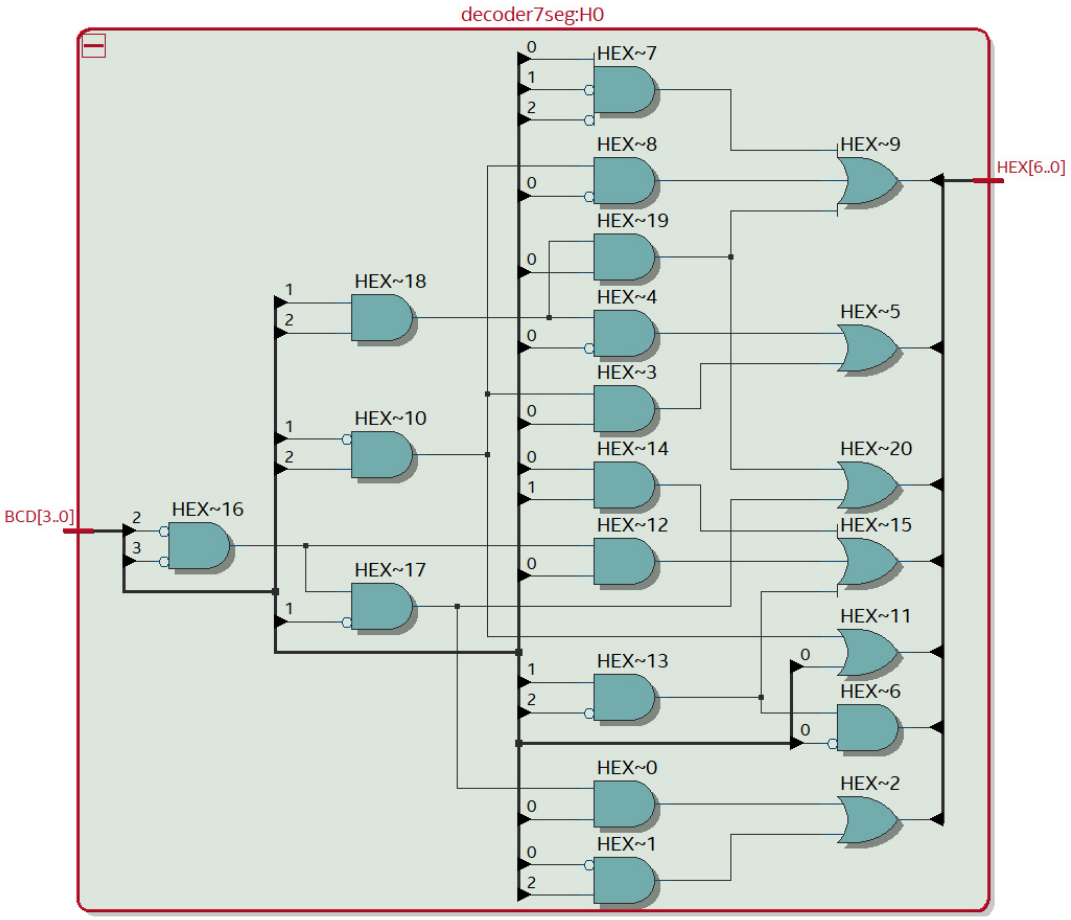
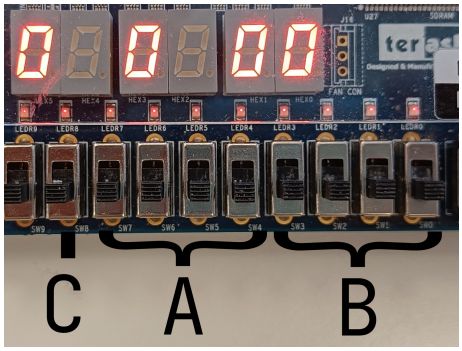
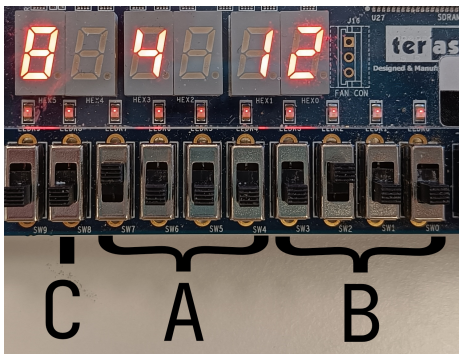


Figure 5.3: RTL synthesizing of the 7-segment decoder

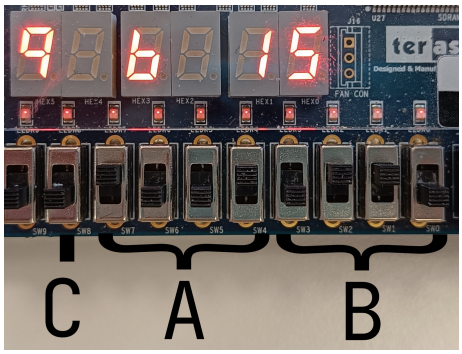
5.2. Results



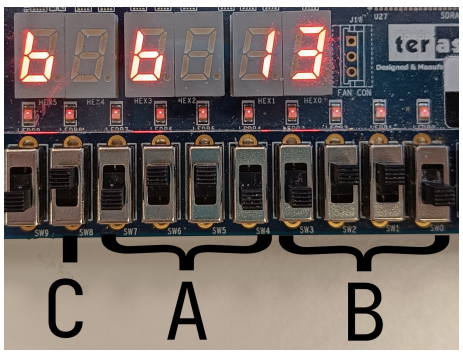
(a) 0 + 0 + 0 = 0



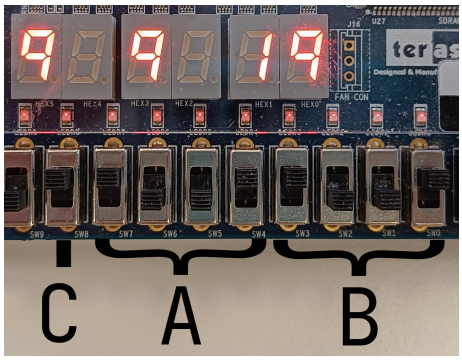
(b) 0 + 8 + 4 = 12



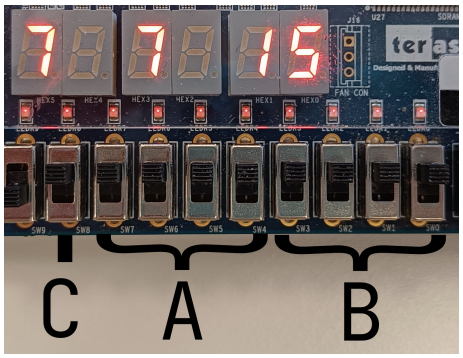
(c) 0 + 9 + 6 = 15



(d) 1 + 6 + 6 = 13



(e) 1 + 9 + 9 = 19



(f) Out of bounds

Figure 5.4: Test results