Programable Logic Circuts
Lab 01

Sølve Kjelseth

September 5, 2025

0. Introduction

This is the first report in this course, detailing the completion of the first lab exerciser, I hope to get some feedback on this. Note: The LaTeX code is open source, github link.

0.0. Table of Contents

0.1. List of Figures

1. Task number 1

This task was a introductory task, where the intent was to copy the provided VHDL instructions into a ModelSim project and make a test bench to simulate it. So that is what I did, there was chosen 8 different combinations for the simulations, I included both all 0 and all 1 for completeness and some other variants. I have one comment/question to the supplied instructions, it names it's architecture "behavioral" while I think I would have called this "structural" as it is describing the structural connection of the components.

1.0. Code

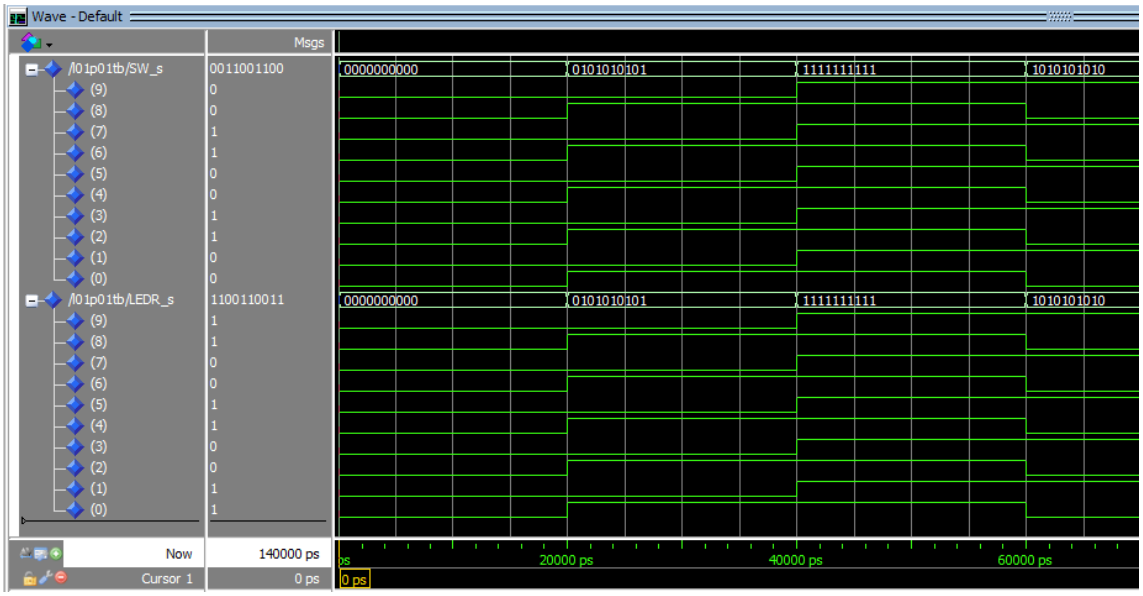
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity L01P01 is
    port(SW   : in  std_logic_vector(9 downto 0);
         LEDR : out std_logic_vector(9 downto 0));
end L01P01;

architecture behaviour of L01P01 is
begin
    LEDR <= SW;
end behaviour;
```

Code 1.0: TLE as described in the task

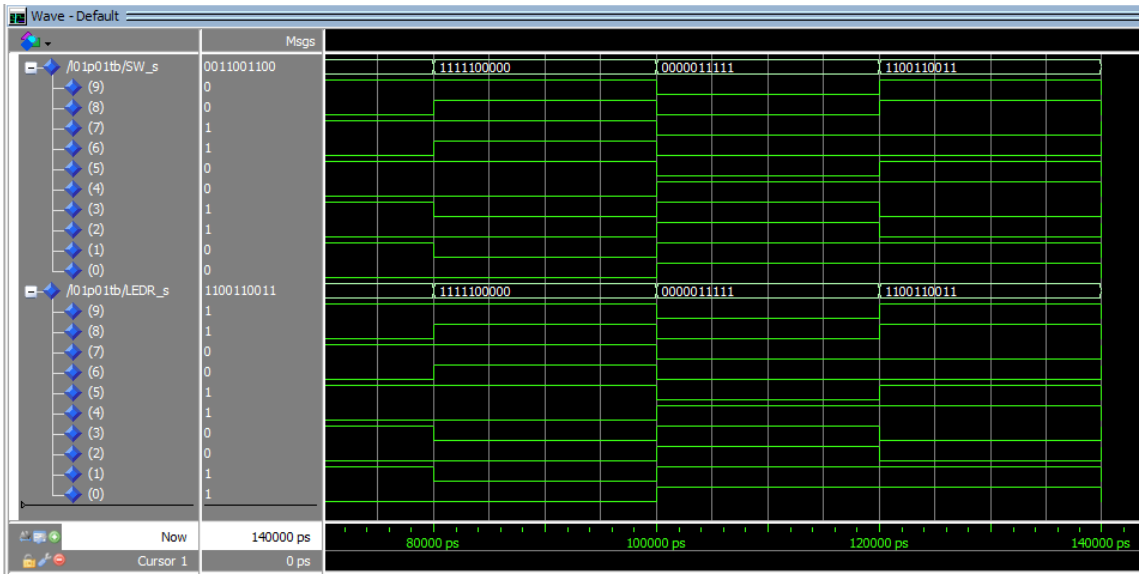
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity L01P01TB is
end L01P01TB;

architecture behaviour of L01P01TB is

    component L01P01
        port(SW   : in  std_logic_vector(9 downto 0);
             LEDR : out std_logic_vector(9 downto 0));
    end component;

    signal SW_s   : std_logic_vector(9 downto 0);
    signal LEDR_s : std_logic_vector(9 downto 0);

begin
    vectors: process begin
        SW_s <= "0000000000";
        wait for 20 ns;
        SW_s <= "0101010101";
        wait for 20 ns;
        SW_s <= "1111111111";
        wait for 20 ns;
        SW_s <= "1010101010";
        wait for 20 ns;
        SW_s <= "1111100000";
        wait for 20 ns;
        SW_s <= "0000011111";
        wait for 20 ns;
        SW_s <= "1100110011";
        wait for 20 ns;
        SW_s <= "0011001100";
        wait;
    end process;

    U1: L01P01 port map (SW_s, LEDR_s);
end behaviour;
```

Code 1.1: TestBench for the TLE

1.1. Simulation results



(a) First four values



(b) Next three values

Figure 1.2: Simulation results

These are the simulation results. A keen eye might notice that the last value of the testbench is not diplayed. The issue is suspected to be that the wait command halts the simulation and the last chosen value will not be seen, for future simulations I will add an extra `wait for 20 ns;` at the end to ensure the last value also gets tested. Due to the low importance of this last simulation result, redoing it has been omitted.

2. Task number 2

This task was to write a simple 2-to-1 multiplexer. With two 4 bit inputs and a single selection bit and one 4 bit output. As the task supplied the statement that would add this functionality it relatively simple to program. Then I made a testbench code to test some inputs and see how it affected the outputs. In total 9 input combinations were tested. As LED nr 9 was used for a flag to see whether or not the selection input was on, the LEDs nr. 4-8 would not be in use and therefore they are assigned an unused signal such that they will not show up in the simulation.

2.0. Code

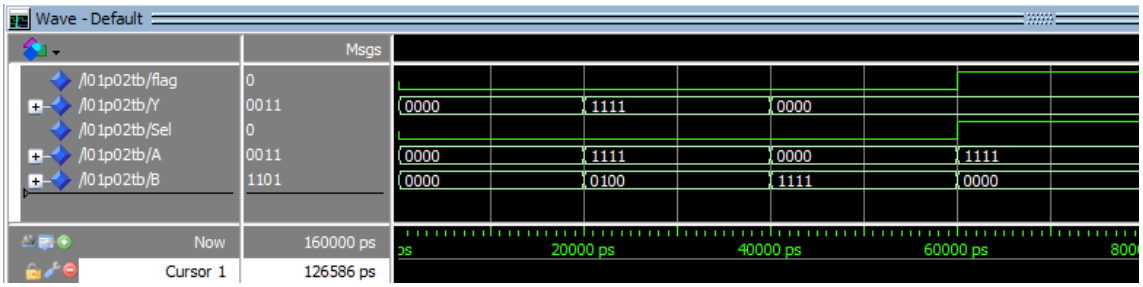
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity L01P02 is
    port(SW   : in  std_logic_vector(9 downto 0);
         LEDR : out std_logic_vector(9 downto 0));
end L01P02;

architecture structural of L01P02 is
begin
    LEDR(9) <= SW(9);
    LEDR(8 downto 4) <= "00000";
    LEDR(3) <= (not(SW(9)) and SW(3)) or (SW(9) and SW(7));
    LEDR(2) <= (not(SW(9)) and SW(2)) or (SW(9) and SW(6));
    LEDR(1) <= (not(SW(9)) and SW(1)) or (SW(9) and SW(5));
    LEDR(0) <= (not(SW(9)) and SW(0)) or (SW(9) and SW(4));
end structural;
```

Code 2.0: TLE for a 4 bit 2 to 1 multiplexer

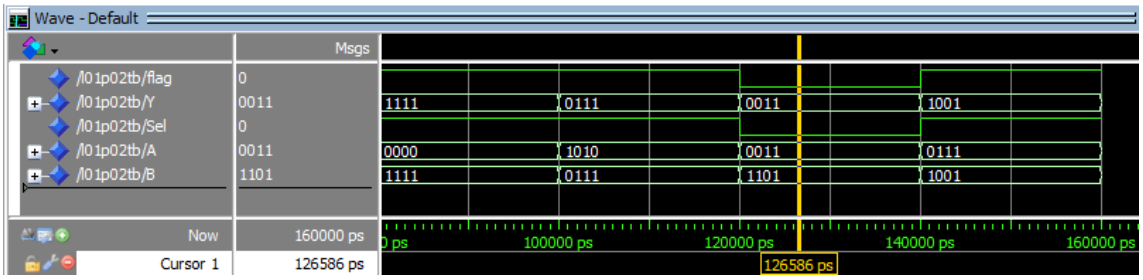
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity L01P02TB is
end L01P02TB;

architecture behaviour of L01P02TB is

    component L01P02
        port(SW   : in  std_logic_vector(9 downto 0);
             LEDR : out std_logic_vector(9 downto 0));
    end component;

    signal A    : std_logic_vector(3 downto 0);
    signal B    : std_logic_vector(3 downto 0);
    signal Sel  : std_logic;
    signal flag : std_logic;
    signal na   : std_logic_vector(4 downto 0);
    signal Y    : std_logic_vector(3 downto 0);

begin
    vectors: process begin
        A <= X"0"; B <= X"0"; Sel <= '0';
        wait for 20 ns;
        A <= X"F"; B <= X"4"; Sel <= '0';
        wait for 20 ns;
        A <= X"0"; B <= X"F"; Sel <= '0';
        wait for 20 ns;
        A <= X"F"; B <= X"0"; Sel <= '1';
        wait for 20 ns;
        A <= X"0"; B <= X"F"; Sel <= '1';
        wait for 20 ns;
        A <= X"A"; B <= X"7"; Sel <= '1';
        wait for 20 ns;
        A <= X"3"; B <= X"D"; Sel <= '0';
        wait for 20 ns;
        A <= X"7"; B <= X"9"; Sel <= '1';
        wait for 20 ns;
        A <= X"B"; B <= X"0"; Sel <= '1';
        wait;
    end process;

    U1: L01P02 port map
        (SW(9) => Sel,
         SW(8) => '0',
         SW(7 downto 4) => B,
         SW(3 downto 0) => A,
         LEDR(9) => flag,
         LEDR(8 downto 4) => na,
         LEDR(3 downto 0) => Y);
end behaviour;
```

Code 2.1: TestBench for the multiplexer

2.1. Simulation results



(a) First four values



(b) Next four values

Figure 2.2: Simulation results

Here you can see that the output is as expected. As long as Sel is set to 0 the flag is set to 0 and the output, Y is equal to input A. Then Sel is switched to 1 and that makes flag also go to 1 and output Y is now equal to input B instead. With the same reasoning as previous task the last line of test values are not shown, and screenshots will not be redone for this task either.

3. Task number 3

This task was similar to the previous but this time a 4-to-1 multiplexer. With four 2 bit inputs and a 2 bit selection and one 2 bit output. The logic statements from the previous task were used on two separate 2-to-1 multiplexers and results stored in a signal, where the final output was a 2-to-1 multiplexing of the signals. This then made up the 4-to-1 multiplexer. Then I made a testbench code to test some inputs and see how it affected the outputs. In total 8 input combinations were tested.

3.0. Code

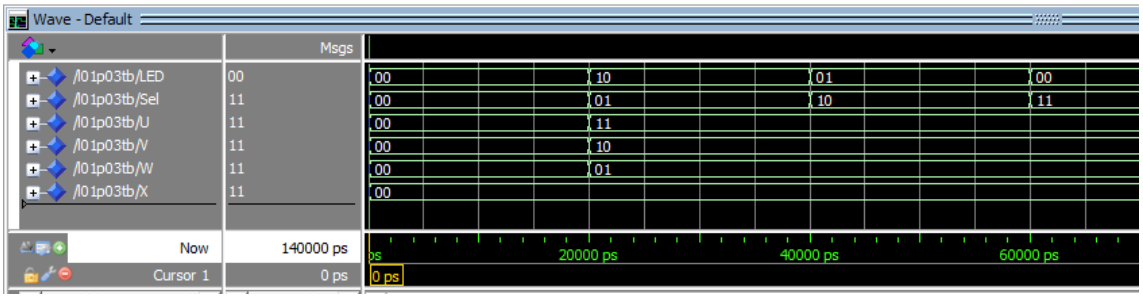
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity L01P03 is
    port(SW   : in  std_logic_vector(9 downto 0);
         LEDR : out std_logic_vector(1 downto 0));
end L01P03;

architecture structural of L01P03 is
    signal s : std_logic_vector(1 downto 0);
    signal u : std_logic_vector(1 downto 0);
    signal v : std_logic_vector(1 downto 0);
    signal w : std_logic_vector(1 downto 0);
    signal x : std_logic_vector(1 downto 0);
    signal a : std_logic_vector(1 downto 0);
    signal b : std_logic_vector(1 downto 0);
    signal m : std_logic_vector(1 downto 0);

begin
    s <= SW(9 downto 8);
    u <= SW(7 downto 6);
    v <= SW(5 downto 4);
    w <= SW(3 downto 2);
    x <= SW(1 downto 0);

    a(0) <= (not s(0) and u(0)) or (s(0) and v(0));
    a(1) <= (not s(0) and u(1)) or (s(0) and v(1));
    b(0) <= (not s(0) and w(0)) or (s(0) and x(0));
    b(1) <= (not s(0) and w(1)) or (s(0) and x(1));

    m(0) <= (not s(1) and a(0)) or (s(1) and b(0));
    m(1) <= (not s(1) and a(1)) or (s(1) and b(1));

    LEDR <= m;
end structural;
```

Code 3.0: TLE for a 2 bit 4 to 1 multiplexer

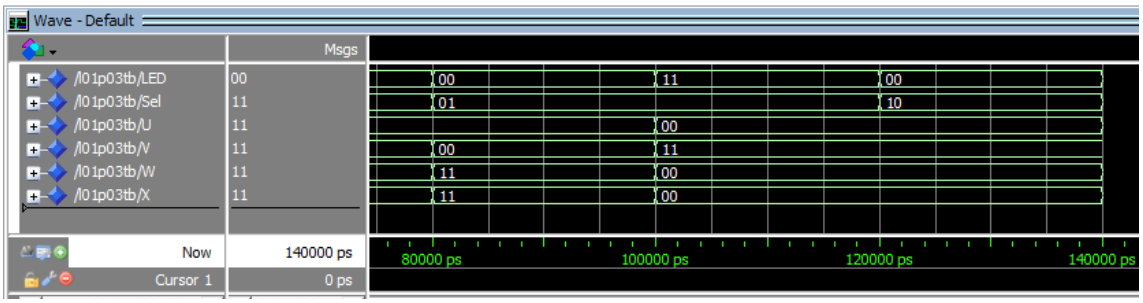
```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity L01P03TB is
end L01P03TB;

architecture behaviour of L01P03TB is

    component L01P03
        port(SW   : in  std_logic_vector(9 downto 0);
             LEDR : out std_logic_vector(1 downto 0));
    end component;

    signal Sel : std_logic_vector(1 downto 0);
    signal U   : std_logic_vector(1 downto 0);
    signal V   : std_logic_vector(1 downto 0);
    signal W   : std_logic_vector(1 downto 0);
    signal X   : std_logic_vector(1 downto 0);
    signal LED : std_logic_vector(1 downto 0);

begin
    vectors: process begin
        U <= "00"; V <= "00"; W <= "00"; X <= "00";
        Sel <= "00";
        wait for 20 ns;
        U <= "11"; V <= "10"; W <= "01"; X <= "00";
        Sel <= "01";
        wait for 20 ns;
        Sel <= "10";
        wait for 20 ns;
        Sel <= "11";
        wait for 20 ns;
        U <= "11"; V <= "00"; W <= "11"; X <= "11";
        Sel <= "01";
        wait for 20 ns;
        U <= "00"; V <= "11"; W <= "00"; X <= "00";
        wait for 20 ns;
        Sel <= "10";
        wait for 20 ns;
        U <= "11"; V <= "11"; W <= "11"; X <= "11";
        Sel <= "11";
        wait;
    end process;

    U1: L01P03 port map
        (SW(9 downto 8) => Sel,
         SW(7 downto 6) => U,
         SW(5 downto 4) => V,
         SW(3 downto 2) => W,
         SW(1 downto 0) => X,
         LEDR => LED);

end behaviour;
```

Code 3.1: TestBench for the multiplexer

3.1. Simulation results



(a) First four values



(b) Next three values

Figure 3.2: Simulation results

Here you can see that the output is as expected. When Sel is set to 00 the output, LED is equal to input U. When Sel is set to 01 the output is equal to input V. When Sel is set to 10 the output is equal to input W. When Sel is set to 11 the output is equal to input X. With the same reasoning as both previous tasks, the last line of test values are not shown, and screenshots will not be redone for this task either.

4. Task number 4

This task was to make a simple decoder for a 7 segment display. It would have a 2 bit input and a 7 bit output with the possibility to display ”d”, ”E”, ”1” or off depending on the input. First I started with finding the boolean expression needed and then simply made a structural architecture based on the expressions for each output. Then it was tested on a FPGA board to show that it worked as expected.

4.0. Boolean expression

To find the boolean expressions I could have calculated it manual using boolean algebra but instead I oped for a truth table to expression solver to save time as there was no point in doing it manual. For segments 1-4 and 6 I saw there was a simple relation, just depending on one of the inputs and those were just used diractly, for the few segments with a bit more advanced relation I used https://tma.main.jp/logic/index_en.html to find the simplified boolean expression from the truth table.

4.1. Code

VHDL

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

entity L01P04 is
    port(SW   : in  std_logic_vector(1 downto 0);
         HEX0 : out std_logic_vector(6 downto 0));
end L01P04;

architecture structural of L01P04 is
    signal c : std_logic_vector(1 downto 0);

begin
    c <= SW;

    HEX0(0) <= (not(c(0)) or c(1));
    HEX0(1) <= c(0);
    HEX0(2) <= c(0);
    HEX0(3) <= c(1);
    HEX0(4) <= c(1);
    HEX0(5) <= (not(c(0)) or c(1));
    HEX0(6) <= c(1);

end structural;
```

Code 4.0: TLE for a simple 7 segment decoder

4.2. RTL

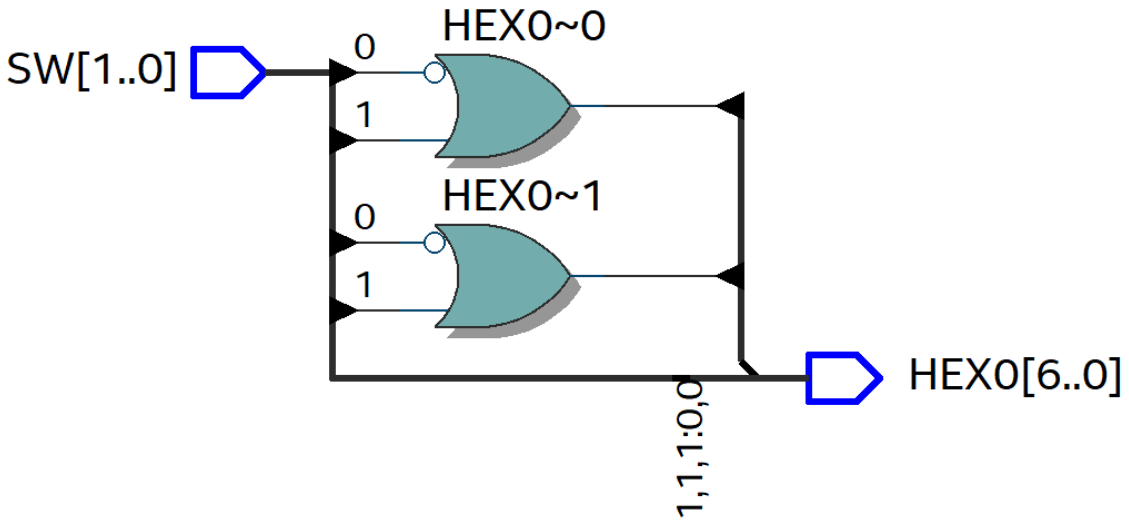
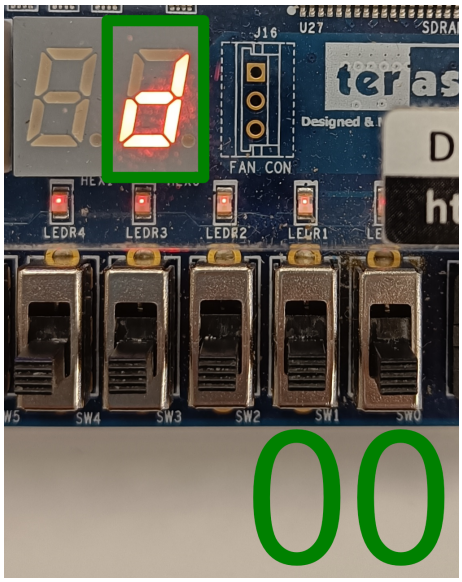


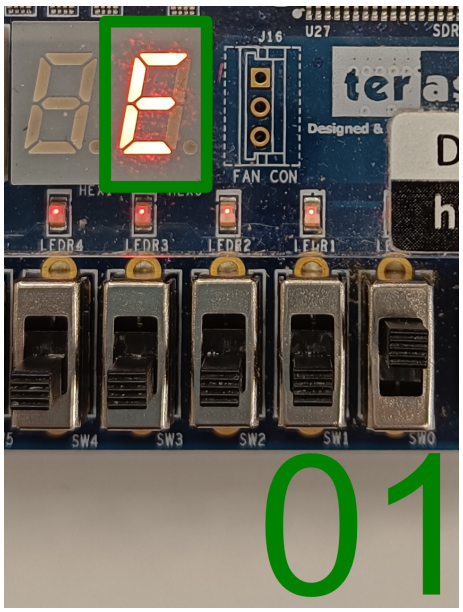
Figure 4.1: RTL synthesizing of the circuit

It is not clear from the RTL view what input is connected to witch segments. Based on the numbering near the bottom, you can see that two instances of SW[0] is connected directly and three instances of SW[1]. Two segments are connected to each an or gate that is connected to SW[1] on one input and inverted SW[0]. This fits logically with what the code is doing, although there was no need to use two gates as the outputs are equal and could be connected together directly.

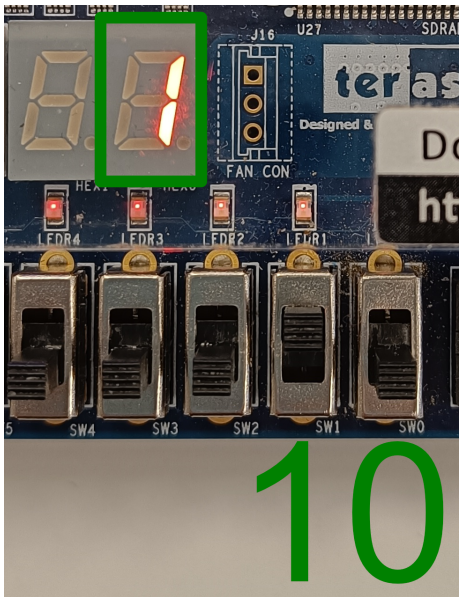
4.3. Results



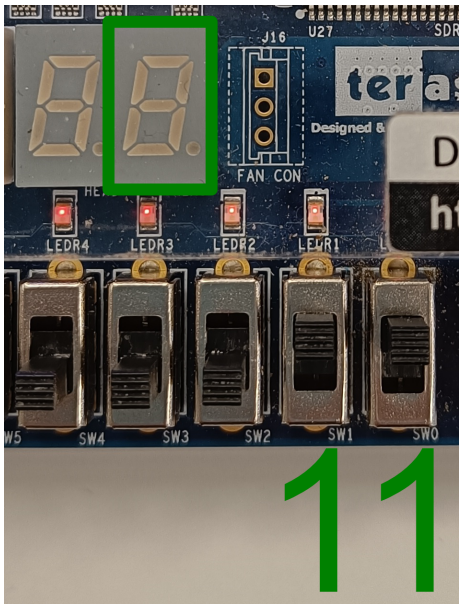
(a) ”d” at 00



(b) ”E” at 01



(c) ”1” at 10



(d) off at 11

Figure 4.2: Test results

Here you can see that the output is as expected. When the input is set to 00 the output is the letter ”d”. When the input is set to 01 the output is the letter ”E”. When the input is set to 10 the output is the number ”1”. When the input is set to 11 the output is off.

5. Task number 5

This is the final task for this lab and it was to combine the 2 bit 4-to-1 multiplexer with the 2 bit 7 segment display encoder. And rotated for 4 displays, such that the first display gets the first input, the second display gets the second input, third gets third and fourth gets fourth. Then when changing the selection input everything would rotate to the display next to it, and the last would wrap around to the first. I started with making the code for a 2 bit 4-to-1 multiplexer, very similar as in part 3. Then I made the code to a 2 bit decoder to a 7 segment display, very similar to part 4. Finally I made a TLE(Top Level Entity) code that imported the two previous codes. The TLE is constructed with four instances of the multiplexer to have one for each display, with the inputs rotated, such that if selection is 00 the output would be the first input on the left most display, the second input on the next display and so on. Four instances of the encoder was also added such that all 4 displays got their own encoder. As an extra feature, described in the task, all the switch inputs should light up each corresponding led to easier show on off position easier in pictures. After this the code was uploaded to a FPGA board and tested.

5.0. Code

VHDL

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

entity MUX_2bit_4to1 is
    port(S, U, V, W, X : in  std_logic_vector(1 downto 0);
         M             : out std_logic_vector(1 downto 0));
end MUX_2bit_4to1;

architecture structural of MUX_2bit_4to1 is

    signal A : std_logic_vector(1 downto 0);
    signal B : std_logic_vector(1 downto 0);

begin
    A(0) <= (not S(0) and U(0)) or (S(0) and V(0));
    A(1) <= (not S(0) and U(1)) or (S(0) and V(1));
    B(0) <= (not S(0) and W(0)) or (S(0) and X(0));
    B(1) <= (not S(0) and W(1)) or (S(0) and X(1));

    M(0) <= (not S(1) and A(0)) or (S(1) and B(0));
    M(1) <= (not S(1) and A(1)) or (S(1) and B(1));

end structural;
```

Code 5.0: 2 bit 4-to-1 multiplexer

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

entity DECODER_7seg is
    port(C : in  std_logic_vector(1 downto 0);
         D : out std_logic_vector(6 downto 0));
end DECODER_7seg;

architecture structural of DECODER_7seg is

begin
    D(0) <= (not(C(0)) or C(1));
    D(1) <= C(0);
    D(2) <= C(0);
    D(3) <= C(1);
    D(4) <= C(1);
    D(5) <= (not(C(0)) or C(1));
    D(6) <= C(1);

end structural;
```

Code 5.1: 2 bit 7 segment decoder

VHDL

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

entity L01P05 is
    port(SW   : in  std_logic_vector(9 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);
         HEX2 : out std_logic_vector(6 downto 0);
         HEX3 : out std_logic_vector(6 downto 0));
end L01P05;

architecture structural of L01P05 is

    component MUX_2bit_4to1
        port(S : in  std_logic_vector(1 downto 0);
             U : in  std_logic_vector(1 downto 0);
             V : in  std_logic_vector(1 downto 0);
             W : in  std_logic_vector(1 downto 0);
             X : in  std_logic_vector(1 downto 0);
             M : out std_logic_vector(1 downto 0));
    end component;

    component DECODER_7seg
        port(C : in  std_logic_vector(1 downto 0);
             D : out std_logic_vector(6 downto 0));
    end component;

    signal A   : std_logic_vector(7 downto 0);
    signal Sel : std_logic_vector(1 downto 0);
    signal IN0 : std_logic_vector(1 downto 0);
    signal IN1 : std_logic_vector(1 downto 0);
    signal IN2 : std_logic_vector(1 downto 0);
    signal IN3 : std_logic_vector(1 downto 0);

begin
    LEDR <= SW;

    IN0 <= SW(1 downto 0);
    IN1 <= SW(3 downto 2);
    IN2 <= SW(5 downto 4);
    IN3 <= SW(7 downto 6);
    Sel <= SW(9 downto 8);

    U0: MUX_2bit_4to1 port map
        (S => Sel,
         U => IN0,
         V => IN1,
         W => IN2,
         X => IN3,
         M => A(1 downto 0));

    U1: MUX_2bit_4to1 port map
        (S => Sel,
         U => IN1,
         V => IN2,
         W => IN3,
         X => IN0,
         M => A(3 downto 2));

```

```
61      U2: MUX_2bit_4to1 port map
62          (S => Sel,
63           U => IN2,
64           V => IN3,
65           W => IN0,
66           X => IN1,
67           M => A(5 downto 4));
68
69      U3: MUX_2bit_4to1 port map
70          (S => Sel,
71           U => IN3,
72           V => IN0,
73           W => IN1,
74           X => IN2,
75           M => A(7 downto 6));
76
77      H0: DECODER_7seg port map
78          (C => A(1 downto 0),
79           D => HEX0);
80
81      H1: DECODER_7seg port map
82          (C => A(3 downto 2),
83           D => HEX1);
84
85      H2: DECODER_7seg port map
86          (C => A(5 downto 4),
87           D => HEX2);
88
89      H3: DECODER_7seg port map
90          (C => A(7 downto 6),
91           D => HEX3);
92
93  end structural;
```

Code 5.2: TLE for rotating display

5.1. RTL

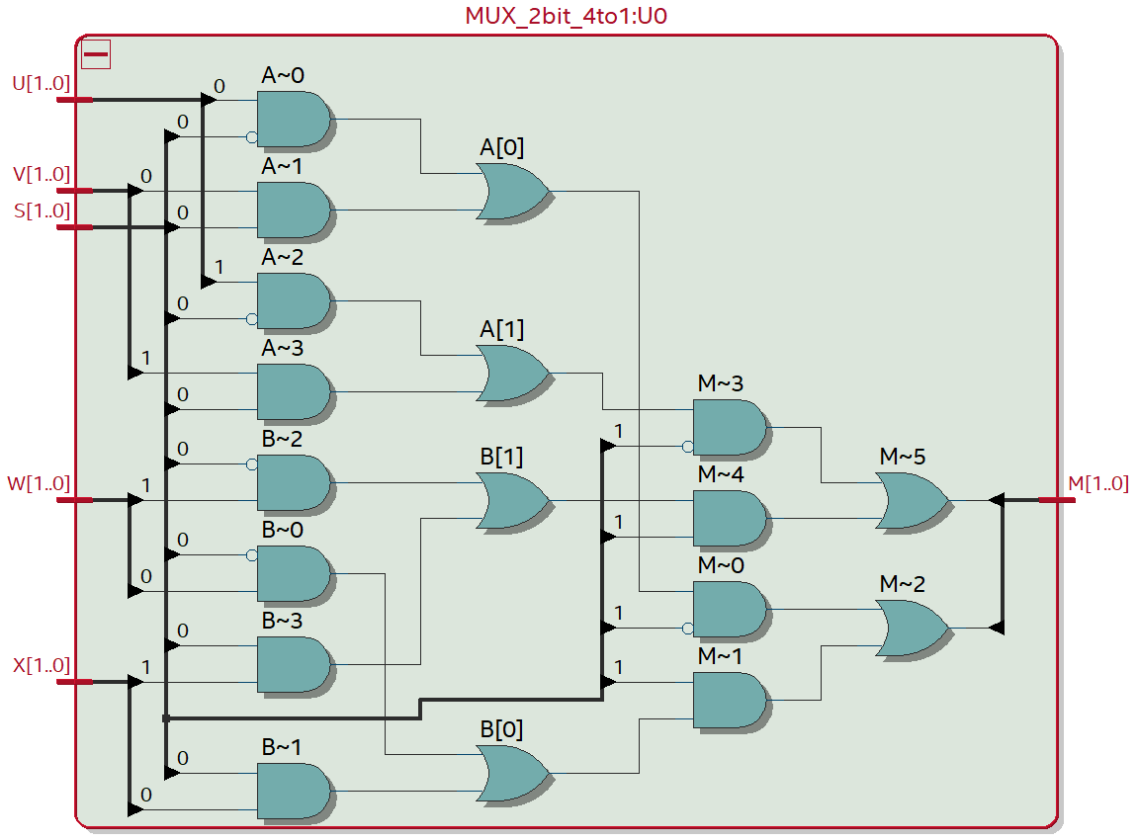


Figure 5.3: RTL synthesizing of the multiplexer circuit

This is the RTL view of the first of the four instances of the multiplexer. If part 3 were synthesized it would look pretty close as it is the same logic used in both circus. Here you can see that the inputs are the U, V, W, X and the selector input S, all of which are 2 bit. and the output is the 2 bit M.

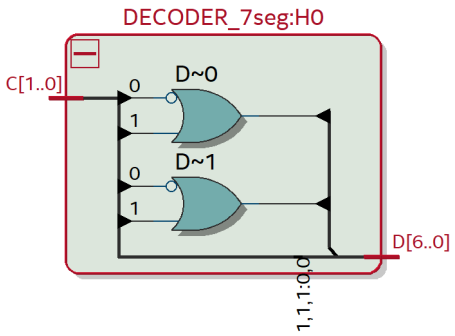


Figure 5.4: RTL synthesizing of the decoder circuit

This is the exact same function as the RTL in task 4 shows, as the codes are nearly identical.

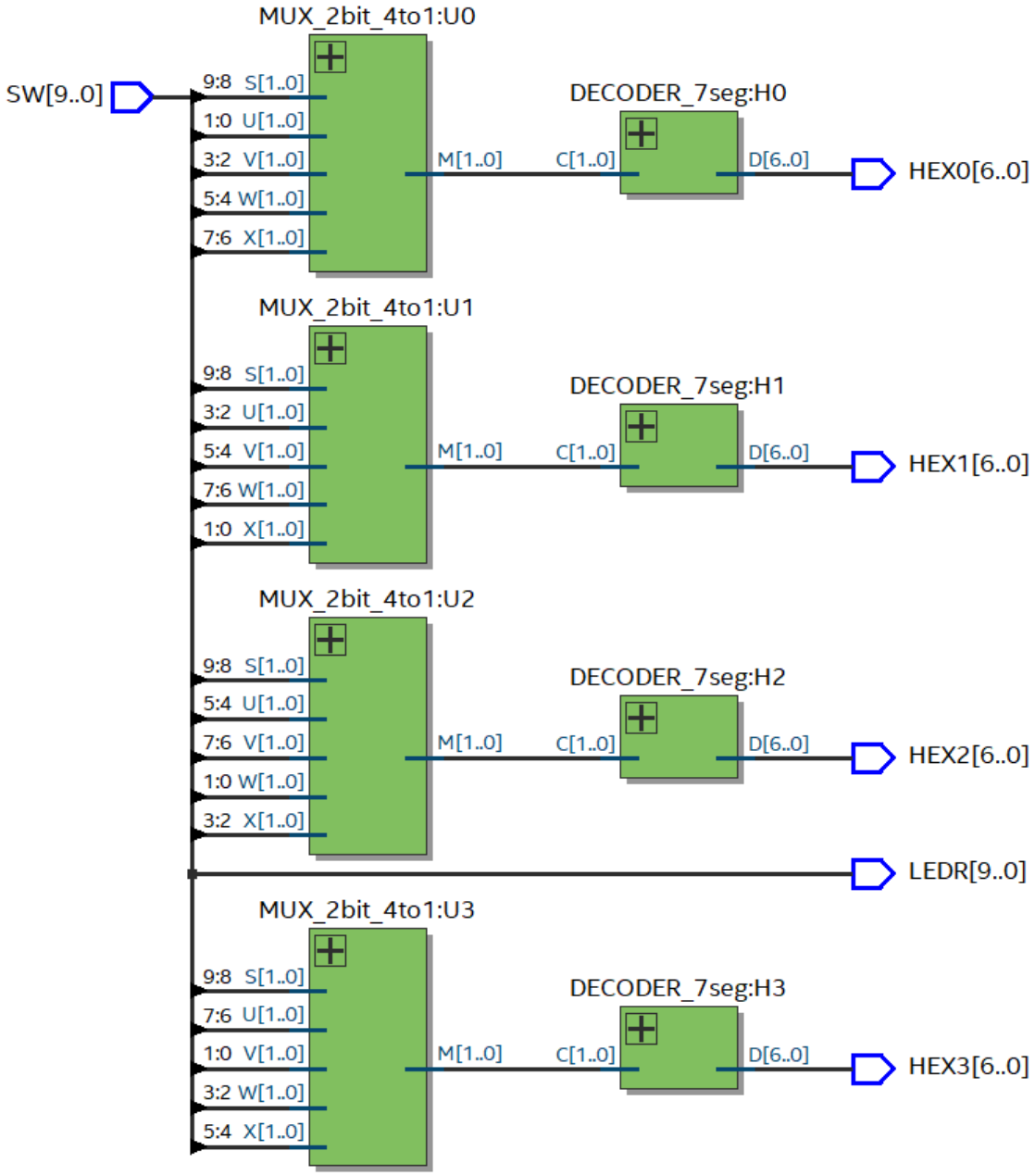
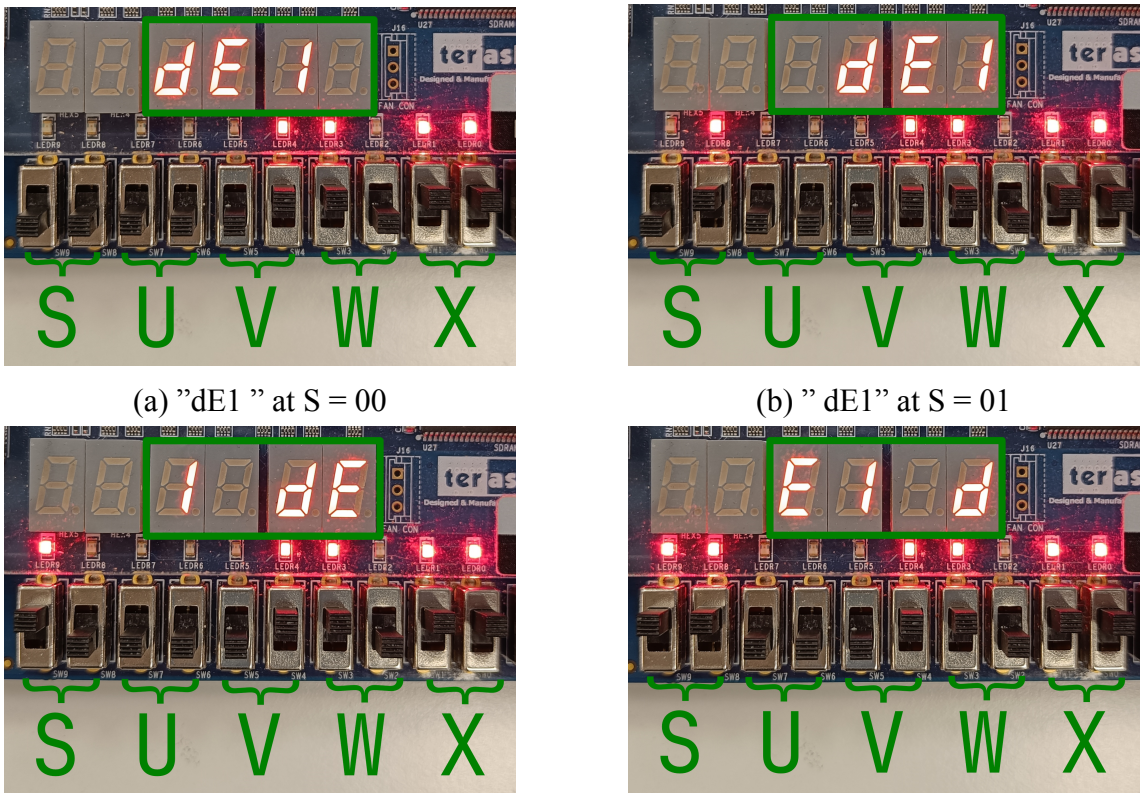


Figure 5.5: RTL synthesizing of the TLE circuit

Here the setup of how each multiplexer is connected to each decoder that is connected to its own display is quite clear. You can also see on the input numbering that the numbers leading into each U, V, W and X on each multiplexer is rotating through the instances. Here you also see that the switches directly control the LEDs as well.

5.2. Results



(a) ”dE1 ” at S = 00

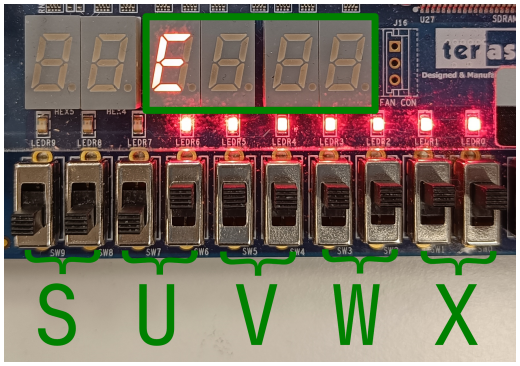
(b) ” dE1” at S = 01

(c) ”1 dE” at s = 10

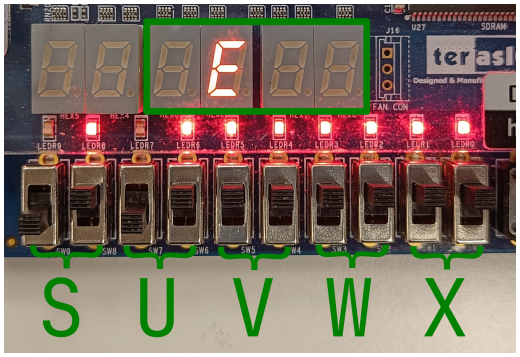
(d) ”E1 d” at s = 11

Figure 5.6: Test results for ”dE1 ” input

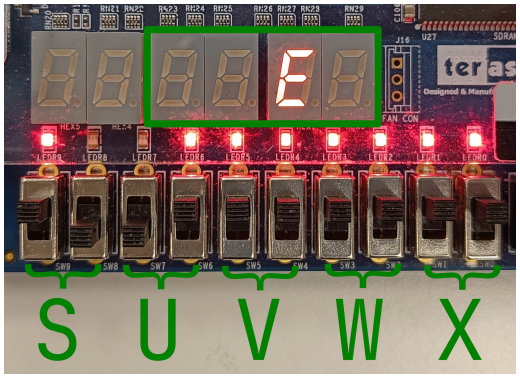
Here you can see that the only input changing is the selection and changing it makes the whole display rotate. The input for U is 00, which is for d, the input for V is 01 which is for E, the input for W is 10 which is for 1, and the input for X is 11 which is for off.



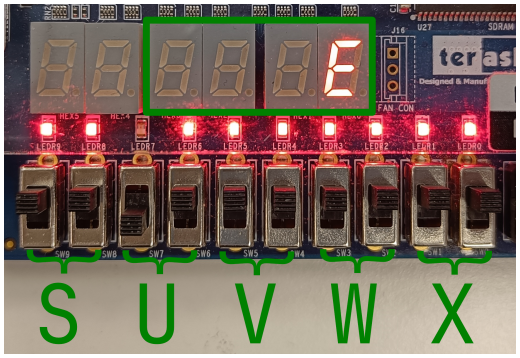
(a) E first at S = 00



(b) E second at S = 01



(c) E third at s = 10



(d) E last at s = 11

Figure 5.7: Test results for ”E ” input

As you can see here as with the previous example it is very easy to see how the rotation. At U = 01 witch makes E and V, W and X = 11 wich makes the rest off, then only changing S between the photos clearly shows how the inputs are rotating along the multiplexer inputs.