Lab#1

– Digital Electronics –

Combinational Circuits

20.10.2021

Group members:

Wojciech Paluszkiewicz,

Muhammad Arshad,

Martin Callegarin Demangeat,

Nicolas Masson

Task.1

1 – Controlling the LEDs

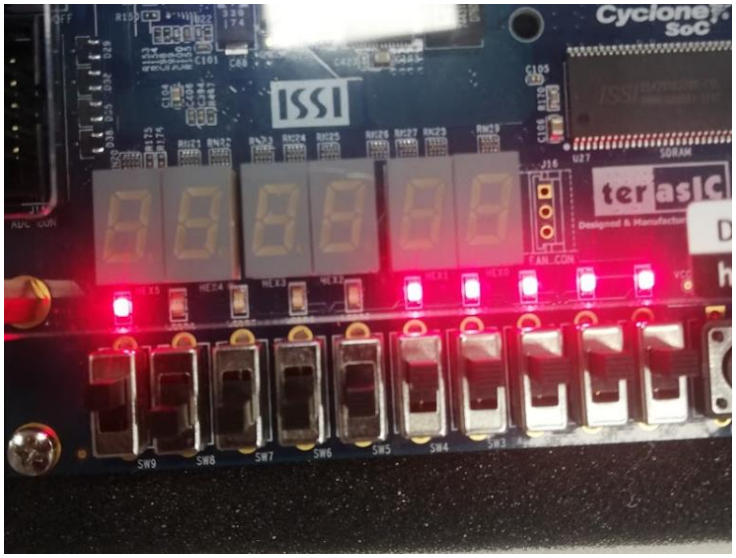
Code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
 -- Simple module that connects
-- the SW switches to the LEDR lights

ENTITY part1 IS PORT (
    SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
    LEDR : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));--using the name of DE1_SoC file
    --the pins are directly assigned from the code to the card
    --we'll be using this technic all along in the futur.
 -- red LEDs
END part1;

 ARCHITECTURE Behavior OF part1 IS
BEGIN
 LEDR <= SW;-- A very simple line of code to test the leds
--assigning to LEDR(9-0) values from switches SW
END Behavior;
```

Description: When switch is a logical “0” (no power) the light above switch will not light.

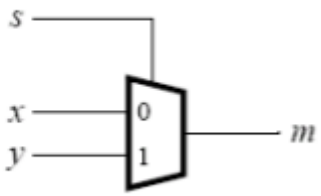
Otherwise, when switch is set with logical “1” (power) the light above this switch will light.

*All switches (from 9 to 0) have diodes in same order .



Task.2

2 - 2-to-1 Multiplexer



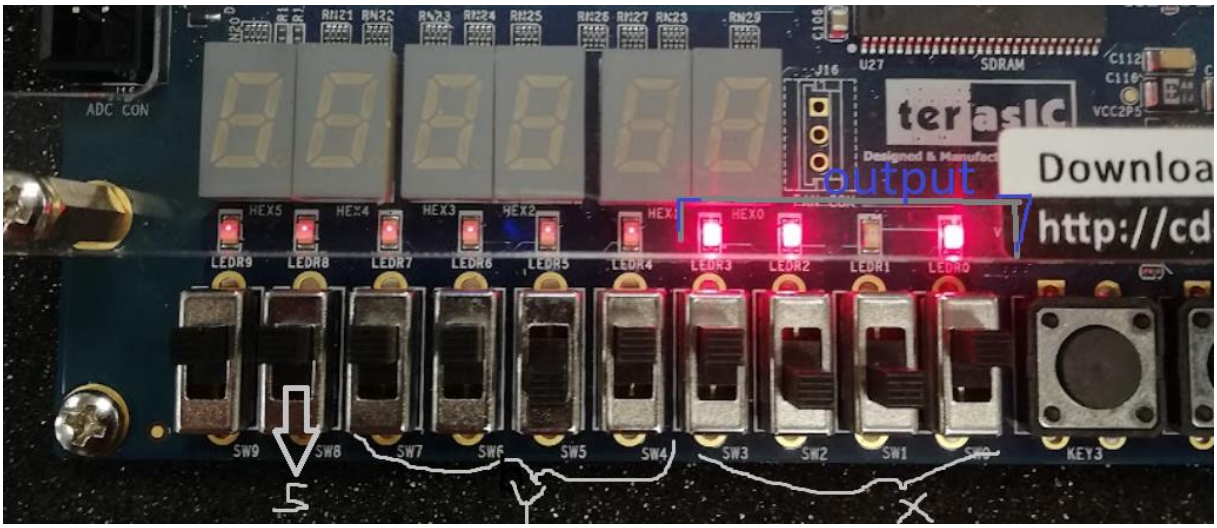
Code:

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY mux IS
5   PORT (
6       sw : in STD_LOGIC_VECTOR(8 downto 0);--Switches
7       Ledr :    OUT STD_LOGIC_VECTOR(3 downto 0));--LED
8   END mux;
9
10
11  ARCHITECTURE Behavior OF mux IS
12  Begin
13   with SW(8) select LEDR<=     --SW(8) is a controlling signal for MUX
14   SW(3 downto 0) when '0',     --OUTPUT has 4 bits, so it is possible to move..
15  --..4b input to output
16   SW(7 downto 4) when '1';
17  END Behavior;
18
```

Here is an exemple of a successful compilation of our code.

```
332102 Design is not fully constrained for setup requirements
332102 Design is not fully constrained for hold requirements
       Quartus Prime TimeQuest Timing Analyzer was successful. 0 errors, 578 warnings
293000 Quartus Prime Full Compilation was successful. 0 errors, 1404 warnings
```

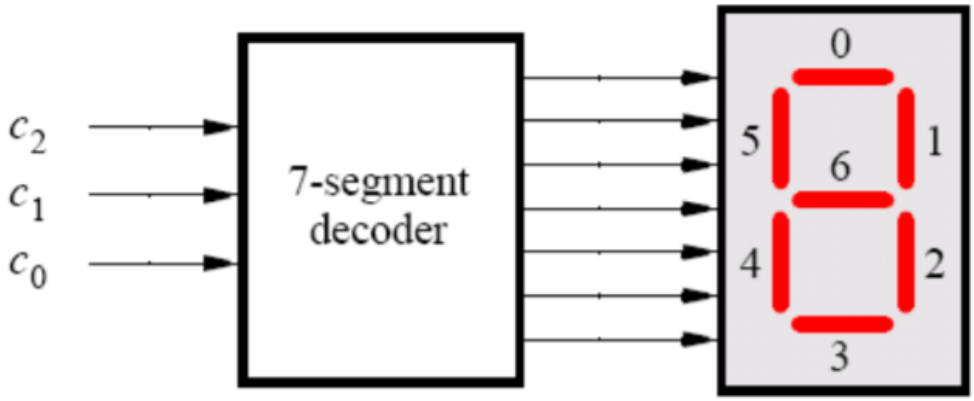
Description: In case when s equals 0, signal “X” goes to output, otherwise (s=1) like in the image below - output signa is equal to y.


output
Downloa
http://cd
ADC CON
SDRAM
HEX5
HEX4
HEX3
HEX2
HEX1
HEX0
LEDR9
LEDR8
LEDR7
LEDR6
LEDR5
LEDR4
LEDR3
LEDR2
LEDR1
LEDR0
SW9
SW8
SW7
SW6
SW5
SW4
SW3
SW2
SW1
SW0
KEY3


SW9
SW8
SW7
SW6
SW5
SW4
SW3

3 - Controlling a 7-segment display

Task.3



For this exercise, we want to display different letters depending on the position of three switches: c0, c1 and c2 according to the table 1.

As before, the things that commands what we want to display are the switches.

The output is called HEX0, and it corresponds to the first 7-segments display beginning to the right. As its name suggests, it is a vector of 7 standard logic values: the 7 segments can be either turned on or turned off and, as for LEDs, a segment is turned on when it receives 0 and turned off when it receives 1.

We must characterize each letter of the table by their corresponding segments, for H, only the segments 0 and 3 are turned off so we have to put 1 on their place. As we put 6 downto 0 and not 0 to 6, the code for H is 0001001 and not 1001000. We repeated the same method for the following letters. Logically, for the blank, every segment is turned off, so we put 1 everywhere.

For example, the description of this code for the picture beneath would be: when you set the with combination “001”, HEX0 must display an E.

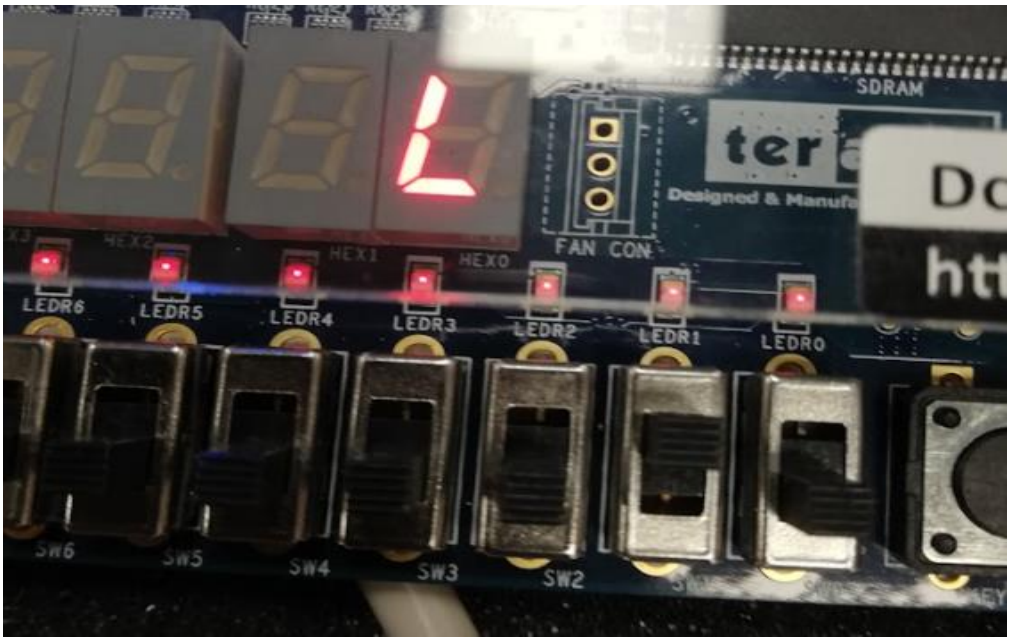
Code:

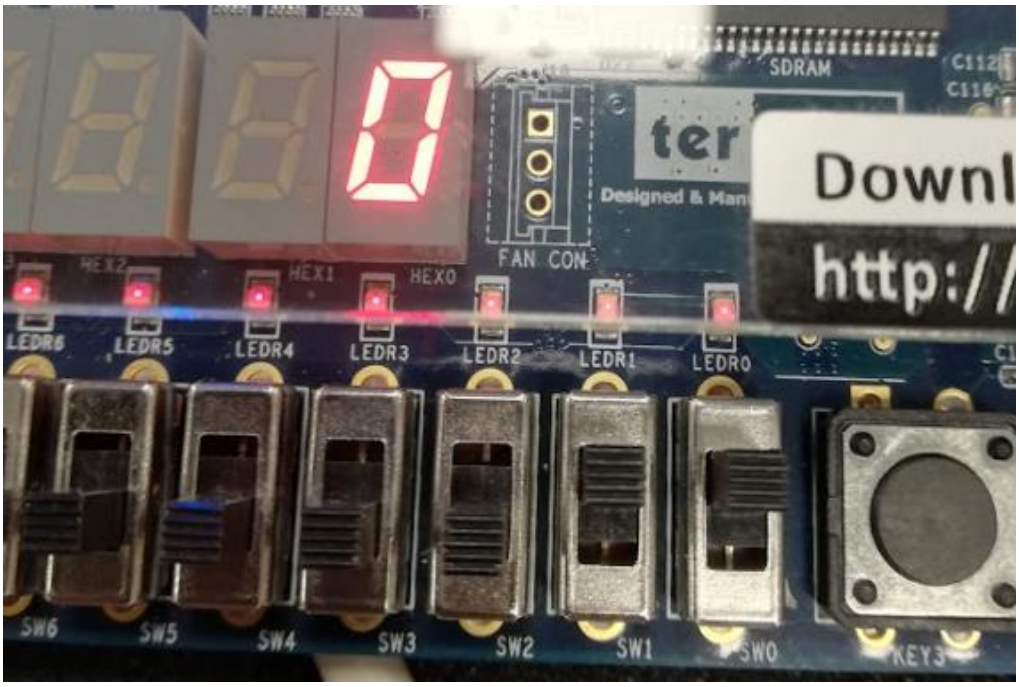
```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY decoder7 IS
5    PORT ( C : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
6    Display : OUT STD_LOGIC_VECTOR(6 downto 0));
7   END decoder7;
8
9   ARCHITECTURE Behavior OF decoder7 IS
10  -- FROM 3 BITS TO 7BITS ON HEX DISPLAY
11     begin
12     PROCESS(C)--we chose to implement a process since C input will tell..
13     --..which character should be displayed on one 7-seg display
14     BEGIN
15  --0 IS active value,light
16     case C is
17     when "000"=>DISPLAY<="0001001";--H
18     when "001"=>DISPLAY<="0000110";--E
19     when "010"=>DISPLAY<="1000111";--L
20     when "011"=>DISPLAY<="1000000";--O
21     when "100"=>DISPLAY<="1000110";--C
22     when "101"=>DISPLAY<="0001110";--F
23     when "110"=>DISPLAY<="0001100";--P
24     when others=>DISPLAY<="1111111";--blank
25     end case;
26  END PROCESS;
27
28  END Behavior;
```

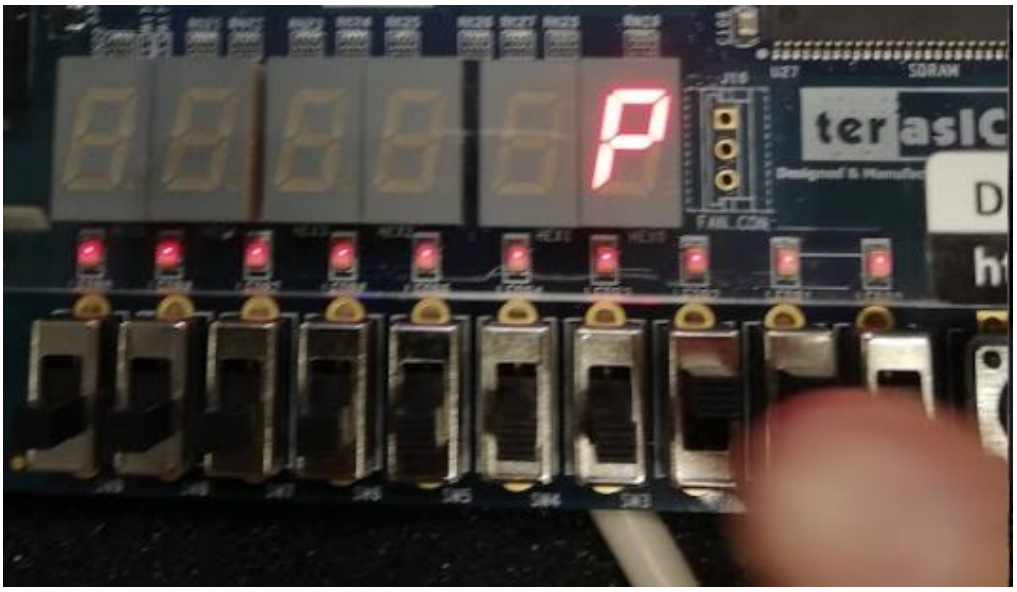
We'll then re-use this VHDL code as a component for the next exercise which requires the same decoder for the display of words.


H
FAN_CON
LEDR9
LEDR8
LEDR7
LEDR6
LEDR5
LEDR4
LEDR3
LEDR2
LEDR1
SW9
SW8
SW7
SW6
SW5
SW4
SW3
SW2
SW1

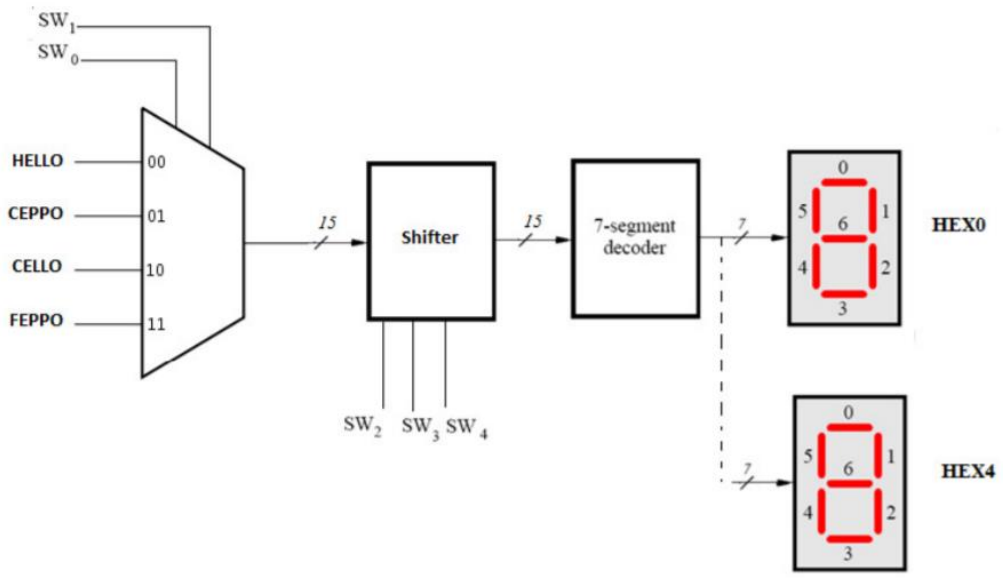

E


L
SDRAM
FAN CON
LEDR6
LEDR5
LEDR4
LEDR3
LEDR2
LEDR1
LEDR0
SW6
SW5
SW4
SW3
SW2


0
SDRAM
FAN CON
HEX2
HEX1
HEX0
LEDR6
LEDR5
LEDR4
LEDR3
LEDR2
LEDR1
LEDR0
SW6
SW5
SW4
SW3
SW2
SW1
SW0
KEY3


P
SDRAM

4 – Multiplexing the 7-segment display output



Explanation of our code:

We will be using multiple components as part of our main code named “ex4”. This main code will receive it’s input instructions from the 5 switches.

The 2 first switches will pilot which word will be sent by the mux. These words being fixed (hello,ceppo and cie) their 5*3-bits combination can be stored onto the mux component code.

Therefore, we have the main code and the mux component for now. Let us show you the component code one by one, then the main code linking them all together.

Mux code:

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   ENTITY mux IS
5   PORT (
6      sel: IN STD_LOGIC_VECTOR(1 downto 0);
7      output : OUT STD_LOGIC_VECTOR(14 downto 0));
8   END mux;
9
10  architecture beh of mux is
11
12  begin
13
14   with sel select output <=
15   "000001010010011" when "00",--hello
16   "100001110110011" when "01",--ceppo and cie
17   "100001010010011" when "10",
18   "101001110110011" when others;
19
20   end beh;
```

Then we introduce the shifter, taking the instruction from the 3rd switch to the 5th. His role is to shift the 15-bits combination in order to change the final word signification.

This is an additional component to the main code.

Shifter code:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY shifter IS
PORT (   input : IN STD_LOGIC_VECTOR(14 downto 0);
         sel: IN STD_LOGIC_VECTOR(2 downto 0);
         output : OUT STD_LOGIC_VECTOR(14 downto 0));
END shifter;


ARCHITECTURE Behavior OF shifter IS

begin
   process(sel)
      begin

      case sel is

         when "000"=>--no shift
         output<=input;

         when "001"=>--one shift
         output(14 downto 3) <= input(11 downto 0);
         output(2 downto 0) <= input(14 downto 12);

         when "010"=>--two shifting
         output(14 downto 6) <= input(8 downto 0);
         output(5 downto 0) <= input(14 downto 9);

         when "011"=>--three shifting
         output(14 downto 9) <= input(5 downto 0);
         output(8 downto 0) <= input(14 downto 6);

         when others=>--four shifting but ensuring reliability with "others"
         output(14 downto 12) <= input(2 downto 0);
         output(11 downto 0) <= input(14 downto 3);

      end case;
   end process;
END Behavior;
```

For the same reason than previously in exercise 3, we chose to pilot the shifter output with a process having the switches (rank 2 to 4) in it's sensitive list since the display must react according to the operator choice of inputs.

Finally, we'll add a last component that we've already presented in exercise 3 that will help us translate the 15-bits combination into 5 part of 3-bits combination to display the word. This is the decoder7. His code is the same, there is no use to show it again. It takes 3-bit std_logic_vector in input and 7-bits std_logic_vector as output.

That makes 4 components to be piloted by the main code.

Main code:

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity ex4 is
5   port(
6       SW : in std_logic_vector (4 downto 0);
7       HEX0,HEX1,HEX2,HEX3,HEX4 : out std_logic_vector (6 downto 0));
8   end ex4;
9
10  architecture beh of ex4 is
11
12  component mux IS
13  PORT (   sel: IN STD_LOGIC_VECTOR(1 downto 0);
14           output : OUT STD_LOGIC_VECTOR(14 downto 0));
15  end component;
16
17  component shifter IS
18  PORT (   input : IN STD_LOGIC_VECTOR(14 downto 0);
19           sel: IN STD_LOGIC_VECTOR(2 downto 0);
20           output : OUT STD_LOGIC_VECTOR(14 downto 0));
21  end component ;
22
23  COMPONENT decoder7
24  PORT (   C : IN STD_LOGIC_VECTOR(2 downto 0);
25           Display : OUT STD_LOGIC_VECTOR(6 downto 0));
26  END COMPONENT;
27
28
29  SIGNAL a1,a2 : std_logic_vector(14 downto 0);
30
31  begin
32
33  MUX0 : mux PORT MAP (SW(1 DOWNTO 0),a1);
34  SHIFT0 : shifter PORT MAP (a1, SW(4 downto 2),a2);
35  H0: decoder7 PORT MAP (a2(2 downto 0), HEX0);
36  H1: decoder7 PORT MAP (a2(5 downto 3), HEX1);
37  H2: decoder7 PORT MAP (a2(8 downto 6), HEX2);
38  H3: decoder7 PORT MAP (a2(11 downto 9), HEX3);
39  H4: decoder7 PORT MAP (a2(14 downto 12), HEX4);
40
41  end beh;
```

What help this code piloting is the portmap. We basically tell the main code what the name of all the signals that are going in and coming out of the different components. Thus, to link the shifter to the main, we tell him that the 15-bits logic vector he'll be receiving is the signal a1 and the switches are from the logic vector SW. then, shifter will be able to deliver a 15-bits signal a2 that is gonna be read by the next component and so on.