Digital Electronics

Laboratory 1: combinational circuit

Group 15:

Aga Andi 281743

Zhang Zhifan 287876

Vacchetto Edoardo 223218

Tong Lin 287649

27/10/2020

Index

This laboratory session consists of the implementation of combinatorial circuits using the Altera DE1 with the final purpose of controlling some 7-segment displays.

Part 1) Controlling the LEDs of the board

In this first simple example, a code to control the LEDs of the DE1 board is to be implemented. The 10 toggle switches $SW_{9\text{-}0}$ of the board are used as the circuit inputs, while the 10 red lights $LEDR_{9\text{-}0}$ have been used to display the output values. (See appendix B-1 for VHDL: Code_1-1)

A simple testbench was used to test the code. (See appendix B-1 for VHDL: Code_1-2)

The obtained waveforms are shown in Fig. 1, appendix A.

It can be clearly noticed that the output SW follows the input LEDR, having used three different inputs, each 10 ns after the other.

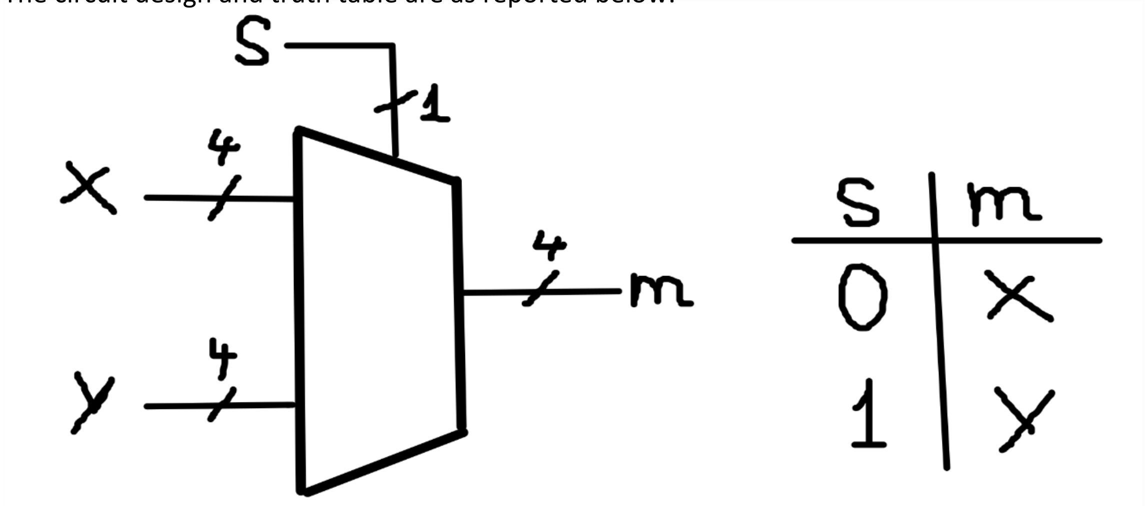
Moreover, the pin assignment has been imported from the DE1_SoC.qsf file, since it was initially decided to use as input and output the exact names used for the pins in that file, namely SW and LEDR. Figure 2 in appendix A shows the resulting pin assignment.

Finally, after compiling the code, a snapshot of the correct flow can be observed in Fig. 3, of appendix A.

Part 2) 2-to-1 Multiplexer

In this part, a 2-to-1 multiplexer must be implemented. It has two four-bit inputs, x and y, and produces a four-bit output m.

The circuit design and truth table are as reported below:



As for the implementation, the switches SW have been used to store all the three inputs, using $SW_{3\text{-}0}$ for x, $SW_{7\text{-}4}$ for y and SW_8 for s. On the other hand, the red lights $LEDR_{3\text{-}0}$ are used to store and display the output. (Refer to appendix B: VHDL Code_2-1).

To test the code, a simple test bench was implemented. (Refer to appendix B: VHDL Code_2-2).

Three signals have been introduced for the inputs in order to select the desired elements of the SW vector, whereas one signal has been introduced for the output. Setting the values for the inputs and changing the parameter "s" each 10 ns, it can be noticed how the output changes based on "s". Specifically, when "s" is set to a logic value 1, the output is equal tox, whereas when "s" is set to a logic value 0, the output is equal to y. This result can be consulted in Figure 4 of appendix A.

As mentioned above, the pin assignment was again done automatically using the file provided, using as input the vectors SW and LEDR. Different names for the input and output could have been used, but then the pins would have to be assigned manually looking at the board manual. This method would be more time consuming. The final pin assignment can be found in Figure 5 of appendix A.

Finally, the flow of the compilation can be seen in Figure 6 of appendix A.

Part 3) Controlling a 7-segment display

In this part we have a 7-segment decoder module which has three input bits (c2, c1, c0), which drive seven outputs that display an alphanumeric character on the 7-segment display. Table 1 shows the alphanumeric character output corresponding to the input combination:

c2,c1,c0	Character
000	H
001	E
010	L
011	O
100	C
101	F
110	P
111	blank

Table 1

The output indexes are shown in Figure A and it should be noted that each segment is lit when its corresponding index is driven to a logic value ‘0’. So, for example, if the letter O to be displayed, then the segments 0 1 2 3 4 5 must be lit, meaning that their corresponding index should be set to a logic value ‘0’: 0000001

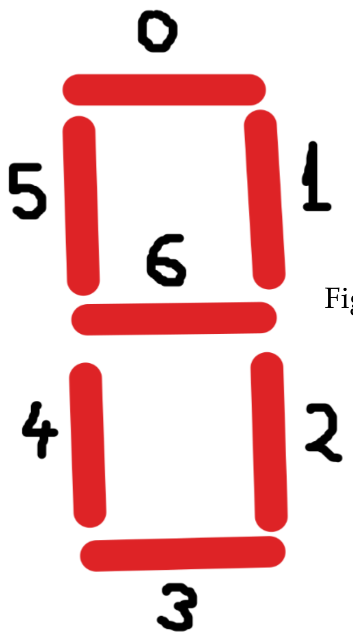


Figure A

First, the circuit was implemented using the “if else” statement provided by VHDL. In this case, the description of the circuit is very simple as it is sufficient to represent the output behavior for each possible combination of inputs. In fact, it is the Quartus synthesizer that associates logical elements to the logic function represented in VHDL, in order to implement the circuit on the FPGA. (See appendix B-3 for VHDL: Code_3-1)

Subsequently the circuit was implemented using manual design techniques such as truth table and the Karnaugh map. (See Appendix B-3 for VHDL: Code_3-2). With this technique it is possible to control the logical gates that will be actually used to implement the circuit.

The switches SW2-0 were used to store the input, whereas the output was stored in the 7-segment-display HEX0.

A testbench was used to check the actual operation of the circuits. The same code was used for both entities, in order to understand if the behavior was the same. The circuit was simulated by providing all the possible input values form the above given table. The output is verified for 10 ns for each input. (See appendix B-3 for VHDL: Code_3-3)

The obtained waveforms are shown in Fig. 7.

Again, based on the DE1_SoC.qsf file, the pin assignment is shown in Fig. 8.

The flow of the compilation can be seen in Fig.9.

Part 4) Multiplexing the 7-segment display output

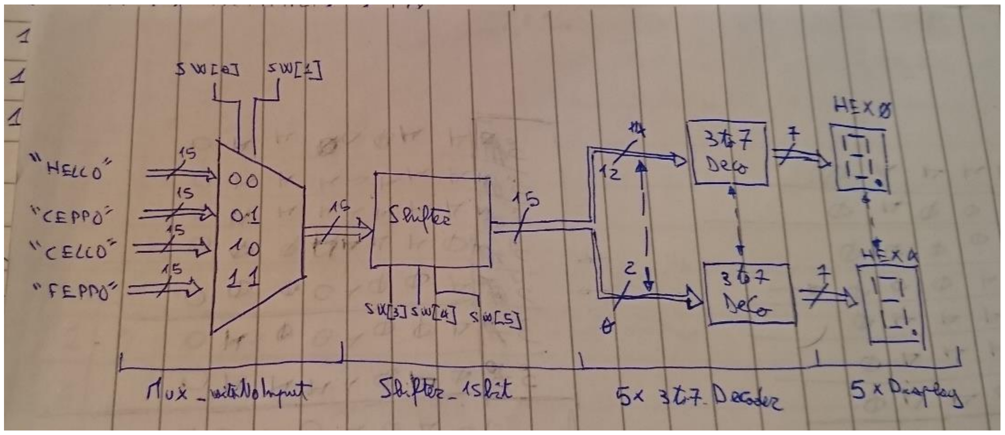
Description

Selecting 5 character group ("HELLO", "CEPPO", "CELLO", "FEPPO") by two inputs SW0-1, displaying the character group on five 7-segment displays respectively. And then rotating the character groups by shifter, the result should be displayed on 7-segments displays after decoding.

There are 5 inputs of this part. SW1-0 are used to select the word, and SW 4-2 are used to control how the characters will be shifted.

SW1-0	Word	SW4-2	Character Position
		000	HELLO
		001	ELLOH
00	HELLO	010	LLOHE
		011	LOHEL
		100	OHELL
01	CEPPO		
10	CELLO	…	
11	FEPPO		

To accomplish these functions, we need to use a 15-bit wide 4-to-1 multiplexer to select the words to be displayed on the 7-segment display, use the shifter to shift the positions, and use the 7-segment decoder to accomplish the display.



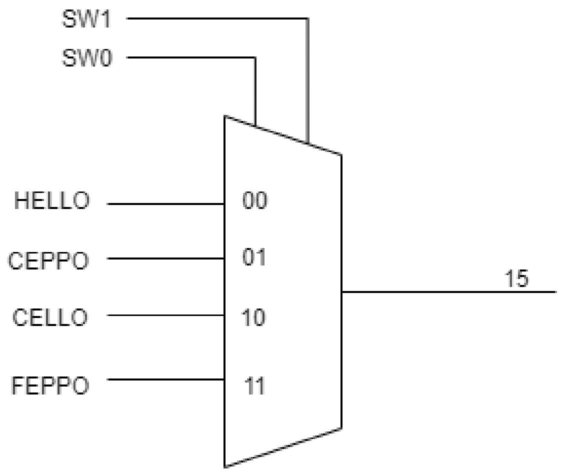
The structure of this project is divided into three main part, respectively Mux_withNoinput (VHDL code title name), Shifter_15Bits, deco_3to7.

The 3 to 7 decoder was designed in part 3. The other two components have to be designed ad hoc.

Mux_withNoInput:

The first step to accomplish the whole function, is to select a word by using a multiplexer with two inputs SW1 and SW0. "Mux_withNoInput" was created by using the multiplexer created in part 2. Initially the 4 to 1 multiplexer was created, connecting 3 multiplexer 2 to 1, see VHDL code on appendix A-4). The VHDL code is in appendix-B1. Then the final circuit was obtained by connecting fifteen multiplexer, 4 to 1, in parallel and fixing the four inputs to the desired value.

The figure and the chart below show the basic function of this section.



SW1, SW0	Words
00	HELLO
01	CEPPO
10	CELLO
11	FEPPO

Explanation: Sel_2 select MUX_1 (x1x2) or MUX_2 (x3x4), if MUX_1 is selected, then select x1 or x2 in MUX_3.

In Mux_withNoInput section, main function is selecting responding 5 Character Group by SW0-1. Then output 15 bits which encoded from the original words "HELLO", "CEPPO", "CELLO", "FEPPO".

The structure of this code is as below:

SW1, SW0	i=	U14-12	11-9	8-6	5-3	2-0
00	X1	000 **H**	001 **E**	010 **L**	010 **L**	011 **O**
01	X2	100 **C**	001 **E**	110 **P**	110 **P**	011 **O**

10	X3	100 **C**	001 **E**	010 **L**	010 **L**	011 **O**
11	X4	101 **F**	001 **E**	110 **P**	110 **P**	011 **O**

Shifter_15Bits:

The last block remaining is the 15 bit shifter. To realize this device the VHDL "case" statement has been exploited. To obtain a shift of one letter (3 bit), the "&" operator has been used, allowing concatenation of the last 12 bits of the word with the first three and so on. (See on appendix B-5 for VHDL code)

From the image taken from the RTL viewer (appendix A-5) you can see that the synthesizer implemented the circuit using only multiplexer.

Five 7 segments display controlling

The final circuit has been implemented by appropriately connecting the three main blocks in a single vhd file. Each component has been connected to the others through specially defined signals and the "port map" statement, as can be seen from the code in appendix B-7.

Particular attention has been paid to the testbench, useful to understand the ideal functioning of the circuit. For this reason, since the possible inputs are not many, it was chosen to simulate them all.

As can be seen from the waveforms obtained in appendix B-7, the designed circuit evolves over time in the desired way, depending on the SW[4] to SW[0] inputs.

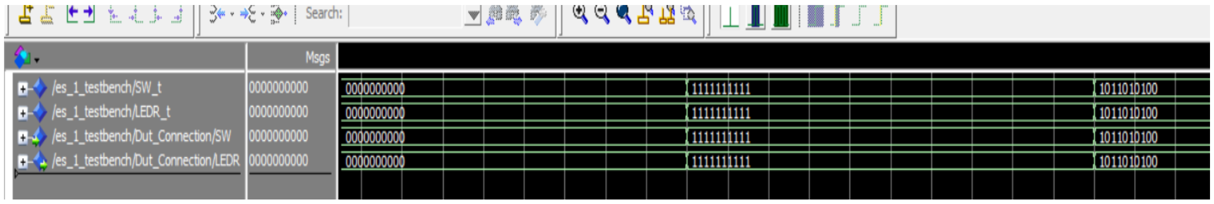
Normally, the code is uploaded on the Altera-DE1 development board to evaluate the actual operation of the circuit in the real world. For this purpose it is necessary to assign a real pin to each input and output present in the circuit (see appendix A-6, for an image of the assigned pin).

APPENDIX:

A. Image and Waveform:

1. Simple Example

Fig. 1:



```
/es_1_testbench/SW_t                 0000000000   0000000000   1111111111   1011010100
/es_1_testbench/LEDR_t               0000000000   0000000000   1111111111   1011010100
/es_1_testbench/Dut_Connection/SW    0000000000   0000000000   1111111111   1011010100
/es_1_testbench/Dut_Connection/LEDR  0000000000   0000000000   1111111111   1011010100
```

Fig. 2:

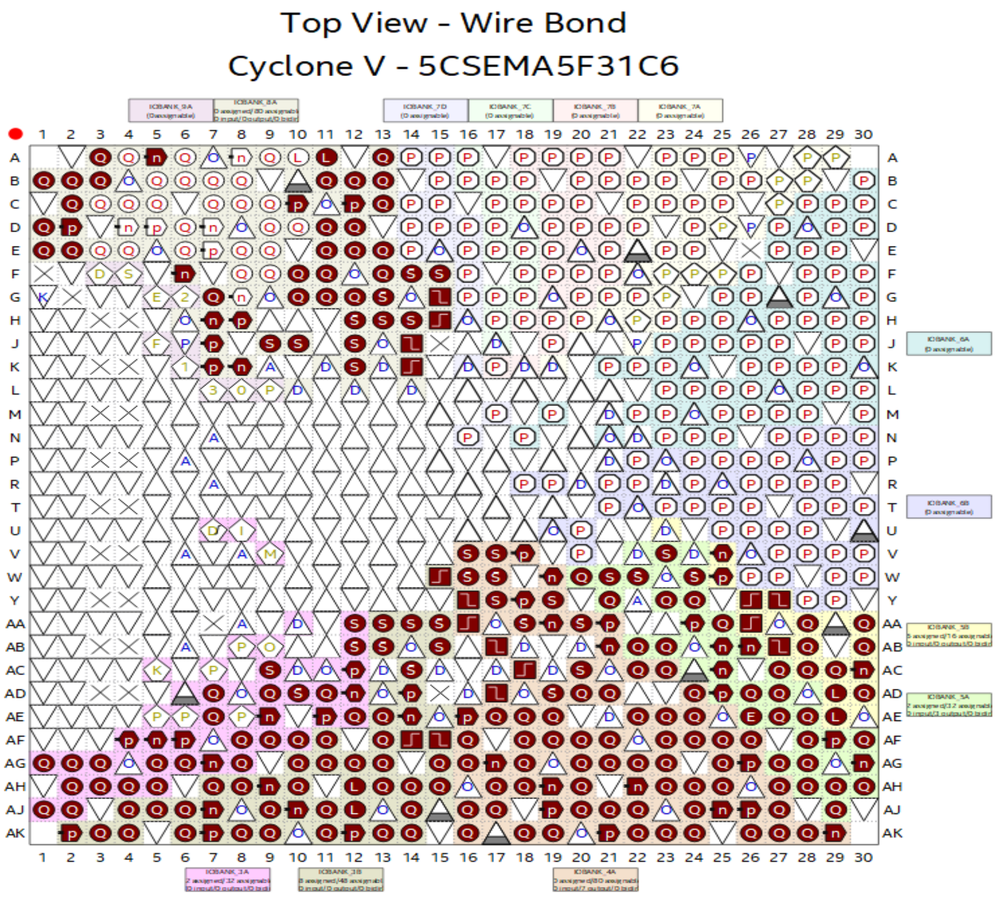
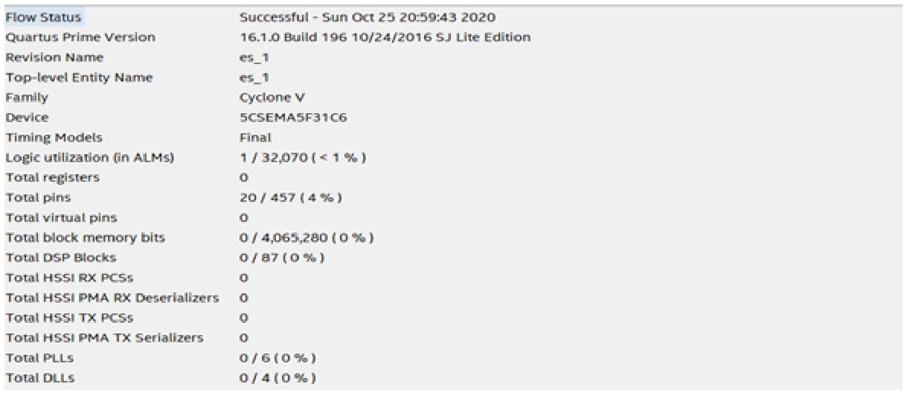


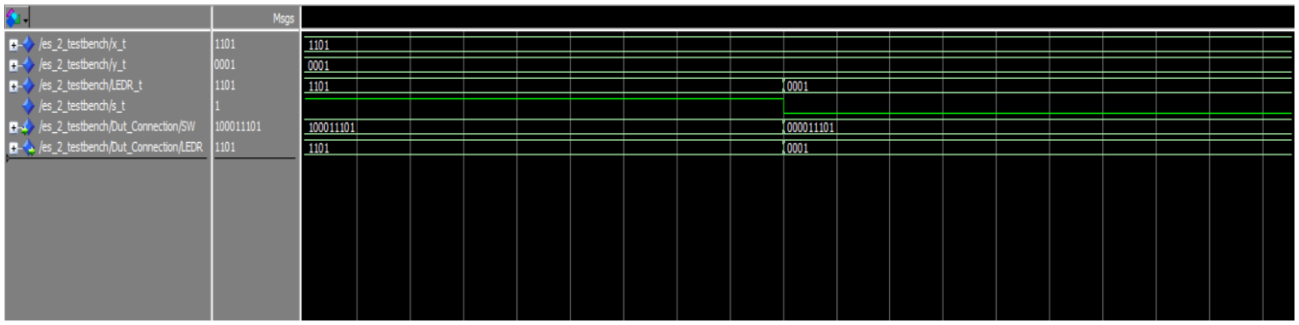
Fig. 3:



Flow Status	Successful - Sun Oct 25 20:59:43 2020
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	es_1
Top-level Entity Name	es_1
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	1 / 32,070 (< 1 %)
Total registers	0
Total pins	20 / 457 (4 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

2. 2-to-1 Multiplexer

Fig. 4:



```
                                    Msgs
/es_2_testbench/x_t                 1101        1101
/es_2_testbench/y_t                 0001        0001
/es_2_testbench/LEDR_t              1101        1101                0001
/es_2_testbench/s_t                 1
/es_2_testbench/Dut_Connection/SW   100011101   100011101           000011101
/es_2_testbench/Dut_Connection/LEDR 1101        1101                0001
```

Fig. 5:

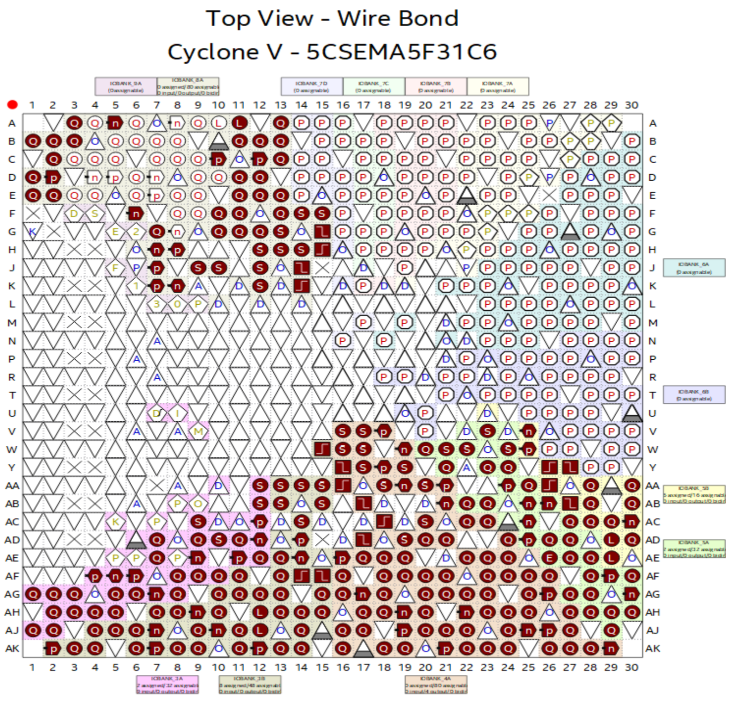
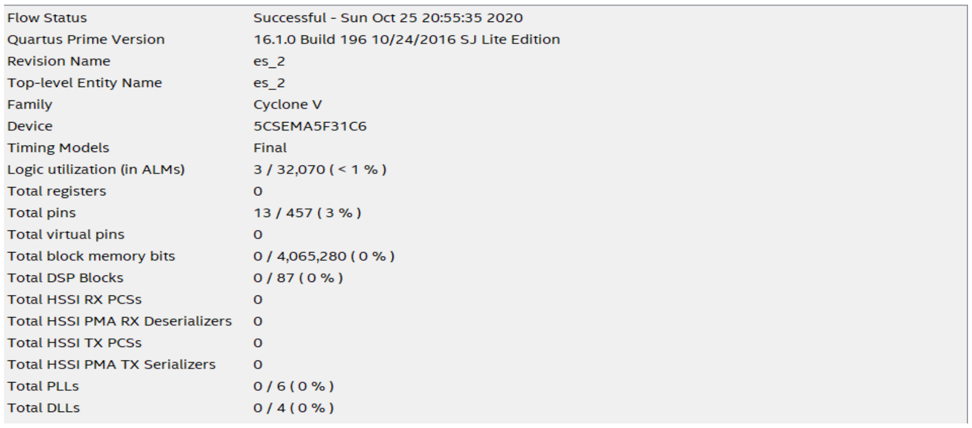


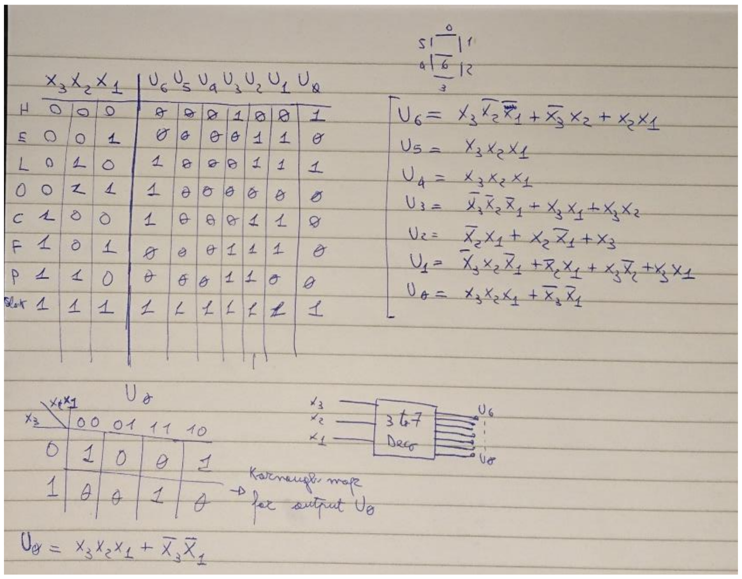
Fig. 6:



Flow Status	Successful - Sun Oct 25 20:55:35 2020
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	es_2
Top-level Entity Name	es_2
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	3 / 32,070 (< 1 %)
Total registers	0
Total pins	13 / 457 (3 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

3. 3 to 7 decoder:

Fig. 7:



	x_3	x_2	x_1	U_6	U_5	U_4	U_3	U_2	U_1	U_0
H	0	0	0	0	0	0	1	0	0	1
E	0	0	1	0	0	0	0	1	1	0
L	0	1	0	1	0	0	0	1	1	1
O	0	1	1	1	0	0	0	0	0	0
C	1	0	0	1	0	0	0	1	1	0
F	1	0	1	0	0	0	1	1	1	0
P	1	1	0	0	0	0	1	1	0	0
Blak	1	1	1	1	1	1	1	1	1	1

$$U_6 = x_3\bar{x}_2\bar{x}_1 + \bar{x}_3 x_2 + x_2 x_1$$
$$U_5 = x_3 x_2 x_1$$
$$U_4 = x_3 x_2 x_1$$
$$U_3 = \bar{x}_3\bar{x}_2\bar{x}_1 + x_3 x_1 + x_3 x_2$$
$$U_2 = \bar{x}_2 x_1 + x_2\bar{x}_1 + x_3$$
$$U_1 = \bar{x}_3 x_2\bar{x}_1 + \bar{x}_2 x_1 + x_3\bar{x}_2 + x_3 x_1$$
$$U_0 = x_3 x_2 x_1 + \bar{x}_3\bar{x}_1$$

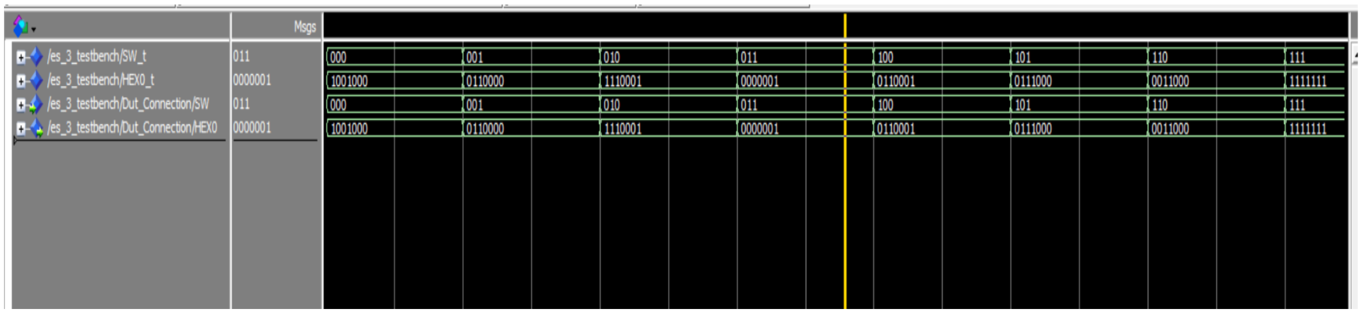
U_0

$x_3 \backslash x_2x_1$	00	01	11	10
0	1	0	0	1
1	0	0	1	0

→ Karnaugh map for output U_0

$$U_0 = x_3 x_2 x_1 + \bar{x}_3\bar{x}_1$$

Fig. 8:



```
                                   Msgs
/es_3_testbench/SW_t               011      000      001      010      011      100      101      110      111
/es_3_testbench/HEX0_t             0000001  1001000  0110000  1110001  0000001  0110001  0111000  0011000  1111111
/es_3_testbench/Dut_Connection/SW  011      000      001      010      011      100      101      110      111
/es_3_testbench/Dut_Connection/HEX0 0000001 1001000  0110000  1110001  0000001  0110001  0111000  0011000  1111111
```

Fig. 9:

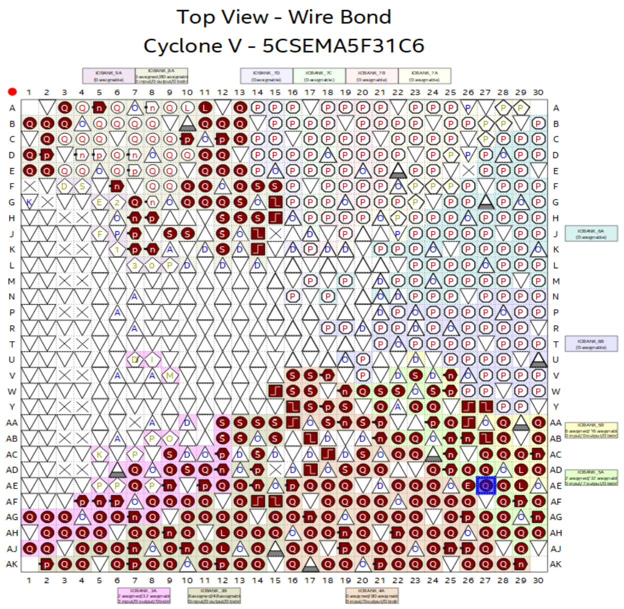
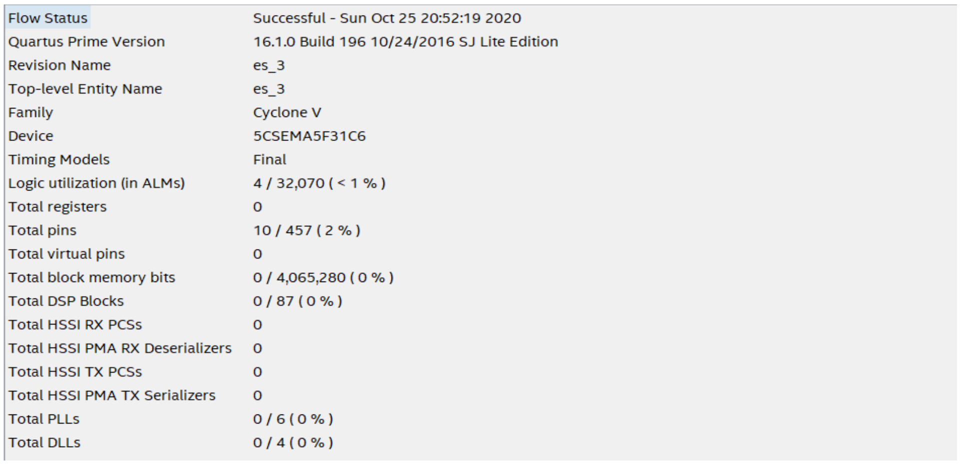


Fig. 10:



Flow Status	Successful - Sun Oct 25 20:52:19 2020
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	es_3
Top-level Entity Name	es_3
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	4 / 32,070 (< 1 %)
Total registers	0
Total pins	10 / 457 (2 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

4. 4 to 1 Multiplexer

Fig. 11: RTL view of 4 to 1 multiplexer

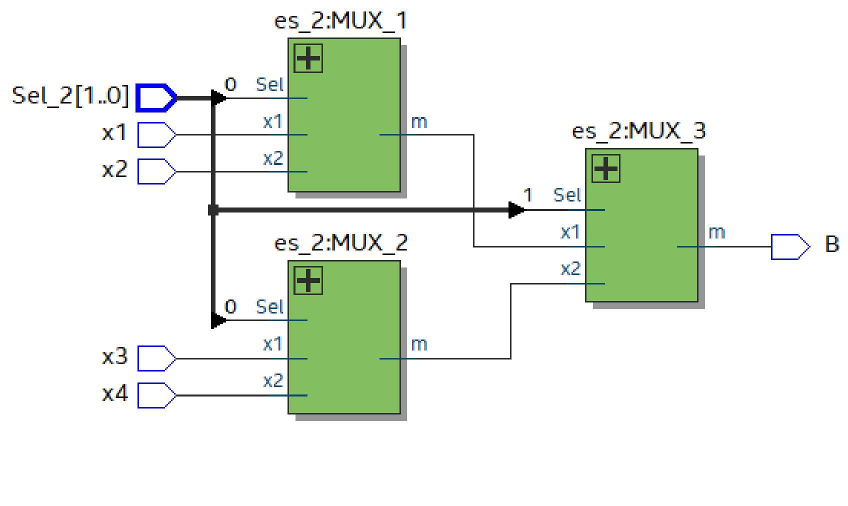


Fig. 12: Pin assignement

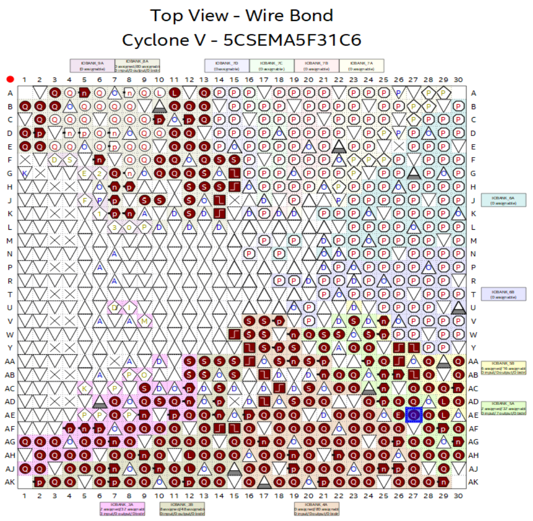
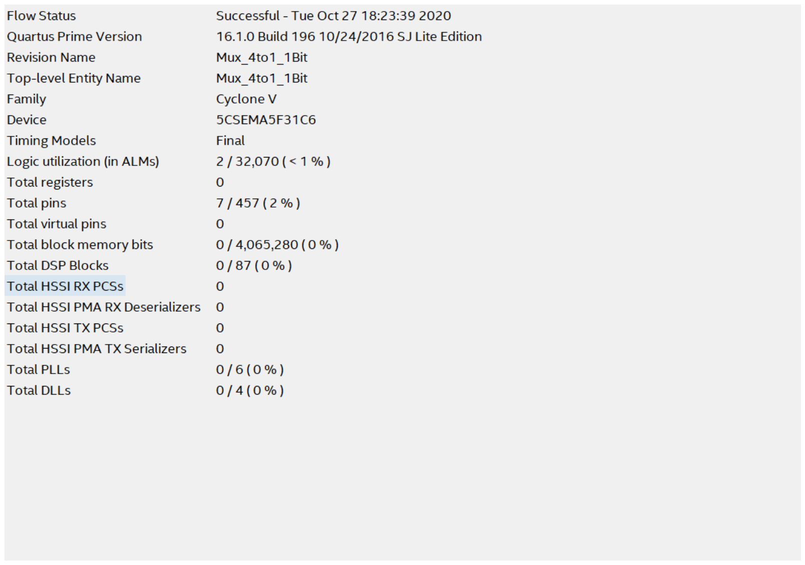


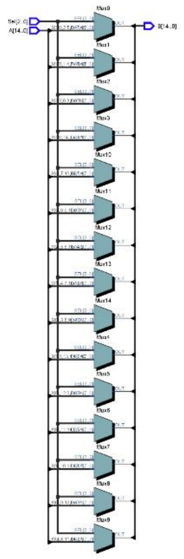
Fig. 13: Compilation transfered



```
Flow Status                        Successful - Tue Oct 27 18:23:39 2020
Quartus Prime Version              16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name                      Mux_4to1_1Bit
Top-level Entity Name              Mux_4to1_1Bit
Family                             Cyclone V
Device                             5CSEMA5F31C6
Timing Models                      Final
Logic utilization (in ALMs)        2 / 32,070 ( < 1 % )
Total registers                    0
Total pins                         7 / 457 ( 2 % )
Total virtual pins                 0
Total block memory bits            0 / 4,065,280 ( 0 % )
Total DSP Blocks                   0 / 87 ( 0 % )
Total HSSI RX PCSs                 0
Total HSSI PMA RX Deserializers    0
Total HSSI TX PCSs                 0
Total HSSI PMA TX Serializers      0
Total PLLs                         0 / 6 ( 0 % )
Total DLLs                         0 / 4 ( 0 % )
```

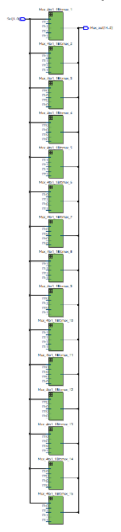
5. 15 Bits Shifter

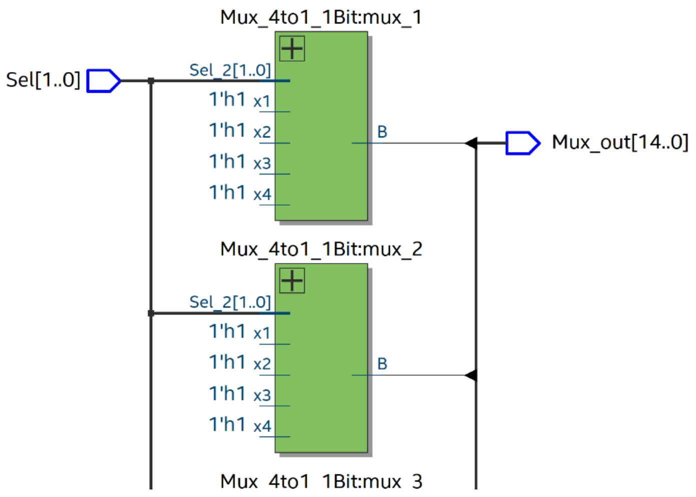
Fig. 14 : RTL view shifter



6. Mux_withNoInput

Fig. 16: RTL view mux no input





7. Five 7-segment displays controlling

Fig.17

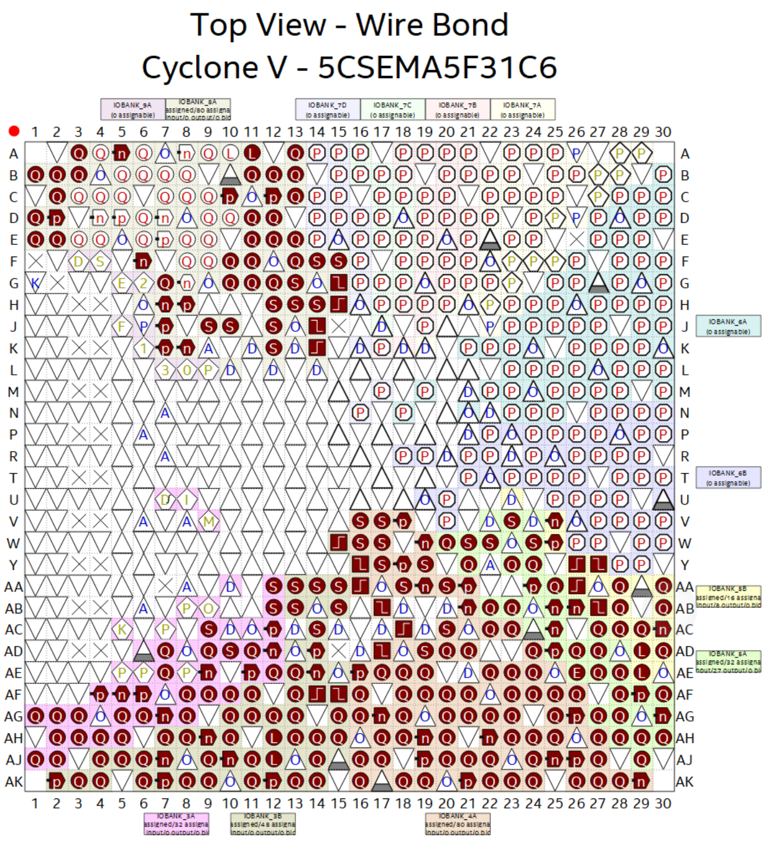


Fig. 18: simulated wave form

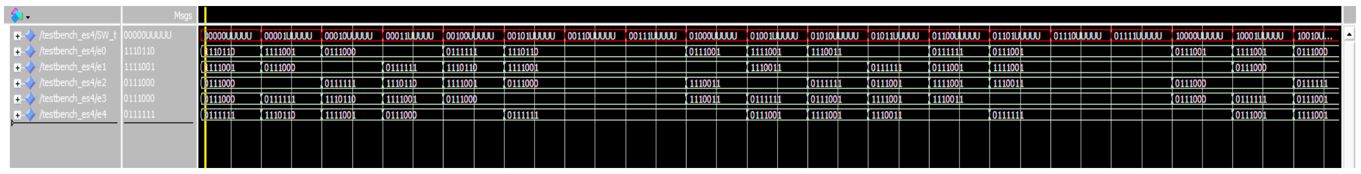
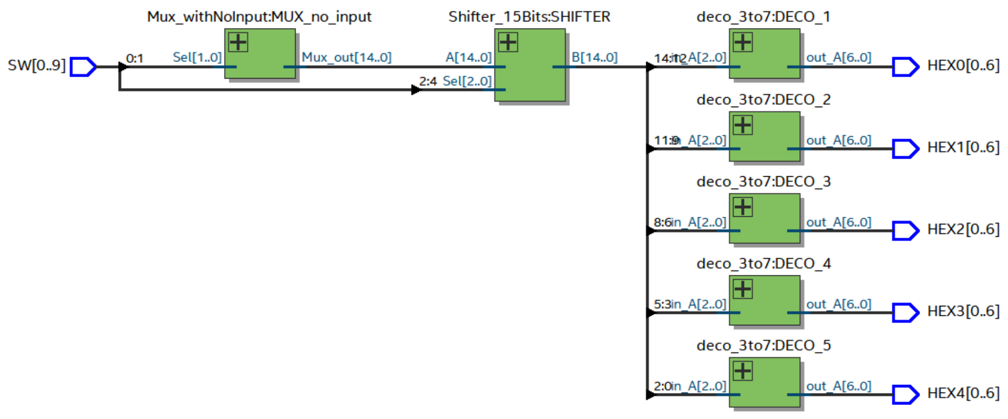


Fig. 19: synthesised circuit view in RTL Viewer



B. VHDL:

1. Simple Example:

Code_1-1:

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

-- Module to connect SW switches to the LEDR lights

entity es_1 is
    port(SW: IN STD_LOGIC_VECTOR(9 downto 0); -- Inputs
         LEDR: OUT STD_LOGIC_VECTOR(9 downto 0)); -- Outputs
end es_1;

architecture beh of es_1 is
begin
    LEDR<=SW;   -- Assign the values of LEDR to SW
end beh;
```

Code_1-2:

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

entity es_1_testbench is
end es_1_testbench;

architecture simulation_of_es_1 of es_1_testbench is

    -- declaration of the component
    component es_1
        port(SW : in std_logic_vector(9 downto 0);
             LEDR : out std_logic_vector(9 downto 0));
    end component;

    -- add two signals for connecting the input and output
    signal SW_t, LEDR_t : std_logic_vector(9 downto 0);

begin

-- Set the connections
Dut_Connection: es_1 port map  (SW => SW_t, LEDR => LEDR_t);

process
begin
    -- We assign different values in the input each 10 ns.
    SW_t <= "0000000000";
    wait for 10 ns;
    SW_t <= "1111111111";
    wait for 10 ns;

    SW_t <= "0010101101";
    wait for 10 ns;
end process ;

end simulation_of_es_1;
```

2. 2-to-1 Multiplexer:

Code_2-1:

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

entity es_2 is
  port (
    SW : IN std_logic_vector(8 downto 0);            -- We decide to store s,
y and x in vector SW
    LEDR : OUT std_logic_vector(3 downto 0));   -- Output stored in vector LEDR
end es_2;

architecture beh of es_2 is
begin
    -- We use SW(8) for Sel, SW(3 downto 0) for x and SW(7 downto 4) for y
    process(SW)
    begin
        -- Here we implement the as shown in the truth table
        if(SW(8)='1') then
            LEDR<=SW(3 downto 0);
        else
         LEDR<=SW(7 downto 4);
        end if;
    end process;
end beh;
```

Code_2-2:

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

entity es_2_testbench is
end es_2_testbench;

architecture simulation_of_es_2 of es_2_testbench is

    -- declaration of the component
    component es_2
```

```
        port (
            SW : IN std_logic_vector(8 downto 0);
            LEDR : OUT std_logic_vector(3 downto 0));
      end component;

    -- We introduce three signals for the inputs and one for the output
    signal x_t, y_t LEDR_t : std_logic_vector(3 downto 0);
    signal s_t : std_logic;
begin

-- Set the connections
Dut_Connection: es_2 PORT MAP  (SW(3 downto 0) => x_t, SW(7 downto 4) => y_t, SW(8)=>s_
t, LEDR => LEDR_t);

process
Begin
    -- We set x and y the same for both cases, changing the input s and
verifying that the multiplexer gives the correct output
    x_t <= "1101";
    y_t <= "0001";
    s_t <= '1';
    wait for 10 ns;

    x_t <= "1101";
    y_t <= "0001";
    s_t <= '0';
    wait for 10 ns;
end process;
end simulation_of_es_2 ;
```

3. 3 to 7 decoder:

Code_3-1:

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

entity es_3 is
  port (
    SW : IN std_logic_vector(2 downto 0);   -- Input stored in SW
    HEX0 : OUT STD_LOGIC_VECTOR(0 to 6));   -- Output stored in HEX0 in an MSB first
fashion
end es_3;

architecture beh of es_3 is
    begin

    process(SW)
    begin
        -- Here we implement the combinations shown in the table
        if(SW="000") then
            HEX0 <= "1001000";  -- H
        elsif (SW="001") then
            HEX0 <= "0110000";  -- E
        elsif (SW="010") then
            HEX0 <= "1110001";  -- L
        elsif (SW="011") then
            HEX0 <= "0000001";  -- O
        elsif (SW="100") then
            HEX0 <= "0110001";  -- C
        elsif (SW="101") then
            HEX0 <= "0111000";  -- F
        elsif (SW="110") then
            HEX0 <= "0011000";  -- P
        elsif (SW="111") then
            HEX0 <= "1111111";  -- blank
        end if;
    end process;

end beh; -- arch
```

Code_3-2:

```
      library ieee;

use ieee.std_logic_1164.all;

entity deco_3to7 is
    port(in_A : in std_logic_vector(2 downto 0);
         out_A : out std_logic_vector(6 downto 0));
end deco_3to7;

architecture behaviour of deco_3to7 is

begin

out_A(0) <= (in_A(2) and in_A(1) and in_A(0)) or (not in_A(2) and not in_A(0));
out_A(1) <= (not in_A(2) and in_A(1) and not in_A(0)) or (not in_A(1) and in_A(0)) or (
in_A(2) and not in_A(1)) or (in_A(2) and in_A(0));
out_A(2) <= (not in_A(1) and in_A(0)) or (in_A(1) and not in_A(0)) or in_A(2);
out_A(3) <= (not in_A(2) and not in_A(1) and not in_A(0)) or (in_A(2) and in_A(0)) or (
in_A(2) and in_A(1));
out_A(4) <= in_A(2) and in_A(1) and in_A(0);
out_A(5) <= in_A(2) and in_A(1) and in_A(0);
out_A(6) <= (in_A(2) and not in_A(1) and not in_A(0)) or (not in_A(2) and in_A(1)) or (
in_A(1) and in_A(0));

end behaviour;
```

Code_3-3:

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

entity es_3_testbench is
end es_3_testbench;

architecture simulation_of_es_3 of es_3_testbench is

    -- declaration of the component
    component es_3
        port (
            SW : IN std_logic_vector(2 downto 0);
            HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6));
        end component;

    -- signals to connect to the input and output
    signal SW_t : std_logic_vector(2 downto 0);
    signal HEX0_t : std_logic_vector(0 to 6);
begin

-- Port mapping
Dut_Connection: es_3 port map  (SW => SW_t, HEX0=>HEX0_t);

process
begin
    -- Set various inputs and check the output
    SW_t<="000";
    wait for 10 ns;
    SW_t<="001";
    wait for 10 ns;
    SW_t<="010";
    wait for 10 ns;
    SW_t<="011";
    wait for 10 ns;
    SW_t<="100";
    wait for 10 ns;
    SW_t<="101";
    wait for 10 ns;
    SW_t<="110";
    wait for 10 ns;
    SW_t<="111";
    wait for 10 ns;
end process;

end simulation_of_es_3;
```

4. 4 to 1 Multiplexer

Code_4-1:

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

entity Mux_4to1_1Bit is
    port(Sel_2 : in std_logic_vector(1 downto 0);
            x1, x2, x3, x4 : in std_logic;
            B : out std_logic);
end Mux_4to1_1Bit;

architecture behaviour of Mux_4to1_1Bit is

component es_2
    port(Sel, x1, x2 : in std_logic;
          m : out std_logic);
end component;

signal Sel_a, Sel_b, mux3_a, mux3_b : std_logic;

begin

MUX_1: es_2 port map (Sel => Sel_b, x1 => x1, x2 => x2, m => mux3_a);
MUX_2: es_2 port map (Sel => Sel_b, x1 => x3, x2 => x4, m => mux3_b);
MUX_3: es_2 port map (Sel => Sel_a, x1 => mux3_a, x2 => mux3_b, m => B);

Sel_a <= Sel_2(1);
Sel_b <= Sel_2(0);

end behaviour;
```

5. Shifter 3 bits

VHDL code:

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

entity Shifter_15Bits is
port(A : in std_logic_vector(14 downto 0);
      B : out std_logic_vector(14 downto 0);
      Sel : in std_logic_vector (2 downto 0));
end Shifter_15Bits;

architecture behaviour of Shifter_15Bits is

begin
```

```

    process(A,Sel)
        Begin
            case Sel is
                when"001" => B <= A(11 downto 0) & A(14 downto 12);
                when "010" => B <= A(8 downto 0) & A(14 downto 9);
                when "011" => B <= A(5 downto 0) & A(14 downto 6);
                when "100" => B <= A(2 downto 0) & A(14 downto 3);
                when others => B <= A;
            end case;
    end process;

end behaviour;
```

VHDL testbench:

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

entity testbench_shifter is
end testbench_shifter;

architecture simulation of testbench_shifter is

    constant Sel_000 : std_logic_vector(2 downto 0) := "000";
    constant Sel_001 : std_logic_vector(2 downto 0) := "001";
    constant Sel_010 : std_logic_vector(2 downto 0) := "010";
    constant Sel_011 : std_logic_vector(2 downto 0) := "011";
    constant Sel_100 : std_logic_vector(2 downto 0) := "100";
    constant Sel_101 : std_logic_vector(2 downto 0) := "101";
    constant Sel_110 : std_logic_vector(2 downto 0) := "110";
    constant Sel_111 : std_logic_vector(2 downto 0) := "111";

    constant Input_1 : std_logic_vector(14 downto 0) := "011101011101001";
    constant Input_2 : std_logic_vector(14 downto 0) := "000000011111111";
    constant Input_3 : std_logic_vector(14 downto 0) := "000111000111000";
    constant Input_4 : std_logic_vector(14 downto 0) := "110011001100110";
    constant Input_5 : std_logic_vector(14 downto 0) := "001100101010111";

    signal Sel_t : std_logic_vector(2 downto 0);
    signal IN_t, OUT_t : std_logic_vector(14 downto 0);
    signal correct : std_logic;

    component Shifter_15Bits   -- component under test
        port(A : in std_logic_vector(14 downto 0);
             B : out std_logic_vector(14 downto 0);
             Sel : in std_logic_vector (2 downto 0));
    end component;

```

```vhdl
begin

    Process -- input signal generation
    Begin

        IN_t <= Input_1;
        Sel_t <= Sel_000;
    if (OUT_t = Input_1) then
        correct <= '1';
    else
        correct <= '1';
    end if;
        wait for 100 ns;

        Sel_t <= Sel_001;
    if (OUT_t = (Input_1(11 downto 0) & Input_1(14 downto 12))) then
        correct <= '1';
    else
        correct <= '1';
    end if;
        wait for 100 ns;

        Sel_t <= Sel_010;
    if (OUT_t = (Input_1(8 downto 0) & Input_1(14 downto 9))) then
        correct <= '1';
    else
        correct <= '1';
    end if;
        wait for 100 ns;

        Sel_t <= Sel_011;
    if (OUT_t = (Input_1(5 downto 0) & Input_1(14 downto 6))) then
        correct <= '1';
    else
        correct <= '1';
    end if;
        wait for 100 ns;

        Sel_t <= Sel_101;

    if (OUT_t = (Input_1(2 downto 0) & Input_1(14 downto 3))) then
        correct <= '1';
    else
        correct <= '1';
    end if;
        wait for 100 ns;

        Sel_t <= Sel_110;
    if (OUT_t = Input_1) then
        correct <= '1';
    else
        correct <= '1';
    end if;
        wait for 100 ns;

        Sel_t <= Sel_111;
    if (OUT_t = Input_1) then
        correct <= '1';
    else
        correct <= '1';
    end if;
        wait for 500 ns;

        IN_t <= Input_2;
        Sel_t <= Sel_000;
        wait for 100 ns;
        Sel_t <= Sel_001;
        wait for 100 ns;
        Sel_t <= Sel_010;
        wait for 100 ns;
        Sel_t <= Sel_011;
        wait for 100 ns;
        Sel_t <= Sel_101;
        wait for 100 ns;
        Sel_t <= Sel_110;
        wait for 100 ns;
        Sel_t <= Sel_111;
        wait for 500 ns;

        IN_t <= Input_3;
        Sel_t <= Sel_000;
        wait for 100 ns;
        Sel_t <= Sel_001;
        wait for 100 ns;
        Sel_t <= Sel_010;
        wait for 100 ns;
        Sel_t <= Sel_011;
        wait for 100 ns;
        Sel_t <= Sel_101;
        wait for 100 ns;
        Sel_t <= Sel_110;
        wait for 100 ns;
        Sel_t <= Sel_111;
        wait for 500 ns;

    end process;

    DUT: Shifter_15Bits port map (A => IN_t, B => OUT_t, Sel => Sel_t);

end simulation;
```

6. Mux_withNoInput

VHDL source:

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

entity Mux_withNoInput is
    PORT (Mux_out : out std_logic_vector(14 downto 0);
            Sel : in std_logic_vector(1 downto 0));
end Mux_withNoInput;

architecture behaviour of Mux_withNoInput is

        component Mux_4to1_1Bit
      port(Sel_2 : in std_logic_vector(1 downto 0);
           x1, x2, x3, x4 : in std_logic;
           B : out std_logic);
        end component;

          signal  x1,x2,x3,x4, B: std_logic_vector(14 downto 0) ;
begin

mux_1: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(0), x2 => x2(0),x3 => x3(0),x4 =>
 x4(0), B => B(0));
mux_2: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(1), x2 => x2(1),x3 => x3(1),x4 =>
 x4(1), B => B(1));
mux_3: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(2), x2 => x2(2),x3 => x3(2),x4 =>
 x4(2), B => B(2));
mux_4: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(3), x2 => x2(3),x3 => x3(3),x4 =>
 x4(3), B => B(3));
mux_5: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(4), x2 => x2(4),x3 => x3(4),x4 =>
 x4(4), B => B(4));
mux_6: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(5), x2 => x2(5),x3 => x3(5),x4 =>
 x4(5), B => B(5));
mux_7: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(6), x2 => x2(6),x3 => x3(6),x4 =>
 x4(6), B => B(6));
mux_8: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(7), x2 => x2(7),x3 => x3(7),x4 =>
 x4(7), B => B(7));
mux_9: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(8), x2 => x2(8),x3 => x3(8),x4 =>
 x4(8), B => B(8));
mux_10: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(9), x2 => x2(9),x3 => x3(9),x4 =
> x4(9), B => B(9));
mux_11: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(10), x2 => x2(10),x3 => x3(10),x
4 => x4(10), B => B(10));
```

```vhdl
mux_12: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(11), x2 => x2(11),x3 => x3(11),x
4 => x4(11), B => B(11));
mux_13: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(12), x2 => x2(12),x3 => x3(12),x
4 => x4(12), B => B(12));
mux_14: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(13), x2 => x2(13),x3 => x3(13),x
4 => x4(13), B => B(13));
mux_15: Mux_4to1_1Bit port map (Sel_2 => Sel, x1 => x1(14), x2 => x2(14),x3 => x3(14),x
4 => x4(14), B => B(14));

x1 <= "000001010010011";  -- HELLO
x2 <= "100001110110011";  -- CEPPO
x3 <= "100001010010011";  --CELLO
x4 <= "101001110110011";  --FEPPO
Mux_out <= B;

end behaviour;
```

VHDL testbench:

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

entity testbench is
end testbench;

architecture simulation of testbench is

    component Mux_withNoInput
        PORT (Mux_out : out std_logic_vector(14 downto 0);
              Sel : in std_logic_vector(1 downto 0));
    end component;

    signal Sel_t : std_logic_vector(1 downto 0);
    signal Mux_Out : std_logic_vector(14 downto 0);
    signal correct : std_logic;

    constant HELLO : std_logic_vector(14 downto 0) := "000001010010011";
    constant CEPPO : std_logic_vector(14 downto 0) := "100001110110011";
    constant CELLO : std_logic_vector(14 downto 0) := "100001010010011";
    constant FEPPO : std_logic_vector(14 downto 0) := "101001110110011";

Begin

    DUT: Mux_withNoInput port map (Mux_out => Mux_Out, Sel => Sel_t);

    Process
    Begin

        Sel_t <= "00";
        if (Mux_out = HELLO) then
        correct <= '1';
        else
            correct <= '0';
        end if;
        wait for 100 ns;

        Sel_t <= "01";
        if (Mux_out = CEPPO) then
            correct <= '1';
        else
            correct <= '0';
        end if;
        wait for 100 ns;

        Sel_t <= "10";
        if (Mux_out = CELLO) then
            correct <= '1';
        else
            correct <= '0';
        end if;
        wait for 100 ns;

        Sel_t <= "11";
        if (Mux_out = FEPPO) then
            correct <= '1';
        else
            correct <= '0';
        end if;
        wait for 100 ns;

    end process;

end simulation;
```

7. Five 7-segments display controlling:

VHDL source:

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

entity es_4 is
    PORT (SW : in std_logic_vector(0 to 9);
            HEX0,HEX1,HEX2,HEX3,HEX4 : out std_logic_vector(0 to 6));
end es_4;
```

```vhdl
architecture behaviour of es_4 is

    component Mux_withNoInput
    port (Mux_out : out std_logic_vector(14 downto 0);
            Sel : in std_logic_vector(1 downto 0));
    end component;

    component Shifter_15Bits
    port(A : in std_logic_vector(14 downto 0);
         B : out std_logic_vector(14 downto 0);
         Sel : in std_logic_vector (2 downto 0));
    end component;

    component deco_3to7
    port(in_A : in std_logic_vector(2 downto 0);
         out_A : out std_logic_vector(6 downto 0));
    end component;

    signal muxToShifter, ShifterToDeco : std_logic_vector(14 downto 0);

begin

    -- connectin all components together
MUX_no_input: Mux_withNoInput port map (Mux_out => muxToShifter, Sel => SW(0 to 1));
  -- mux selector
SHIFTER: Shifter_15Bits port map (A => muxToShifter, B => ShifterToDeco, Sel => SW(2 to
 4));

DECO_1: deco_3to7 port map (in_A => ShifterToDeco(14 downto 12), out_A => HEX0);
DECO_2: deco_3to7 port map (in_A => ShifterToDeco(11 downto 9), out_A => HEX1);
DECO_3: deco_3to7 port map (in_A => ShifterToDeco(8 downto 6), out_A => HEX2);
DECO_4: deco_3to7 port map (in_A => ShifterToDeco(5 downto 3), out_A => HEX3);
DECO_5: deco_3to7 port map (in_A => ShifterToDeco(2 downto 0), out_A => HEX4);


end behaviour;
```

VHDL testbench:

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

entity testbench_es4 is   -- void entity for a testbench
end testbench_es4;

architecture test of testbench_es4 is

    component es_4    --declaring component to test
        PORT (SW : in std_logic_vector(0 to 9);
        HEX0,HEX1,HEX2,HEX3,HEX4 : out std_logic_vector(0 to 6));
    end component;

    -- constant for the MUX sel signal
    constant cHELLO : std_logic_vector(1 downto 0) := "00";   -
-  := symbol is used to assign a value to a constant
    constant cCEPPO : std_logic_vector(1 downto 0) := "01";
    constant cCELLO : std_logic_vector(1 downto 0) := "10";
    constant cFEPPO : std_logic_vector(1 downto 0) := "11";

    signal SW_t : std_logic_vector(0 to 9);
    signal e0, e1, e2, e3, e4 : std_logic_vector(0 to 6);

    Begin

        DUT: es_4 port map (SW => SW_t, HEX0 => e0, HEX1 => e1, HEX2 => e2, HEX3 => e3,
 HEX4 => e4); -- creating connection with signal

        Process  -- Input Generation
        Begin

        SW_t(5) <= '0';
        SW_t(6) <= '0';
        SW_t(7) <= '0';
        SW_t(8) <= '0';
        SW_t(9) <= '0';
                                        -- HELLO
        SW_t(0 to 1) <= cHELLO;  -
- selection of the word to write, thanks to mux_withFix_input
        SW_t(2 to 4) <= "000";  -- shifting the word whit Shifter_15Bits
        wait for 10 ns;
        SW_t(2 to 4) <= "001";
        wait for 10 ns;
        SW_t(2 to 4) <= "010";
        wait for 10 ns;
        SW_t(2 to 4) <= "011";
        wait for 10 ns;
        SW_t(2 to 4) <= "100";
        wait for 10 ns;
        SW_t(2 to 4) <= "101";
        wait for 10 ns;
        SW_t(2 to 4) <= "110";
        wait for 10 ns;
        SW_t(2 to 4) <= "111";
        wait for 10 ns;
                                      -- CEPPO
        SW_t(0 to 1) <= cCEPPO;
        SW_t(2 to 4) <= "000";  -- shifting the word whit Shifter_15Bits
        wait for 10 ns;
        SW_t(2 to 4) <= "001";
```

```
        wait for 10 ns;
        SW_t(2 to 4) <= "010";
        wait for 10 ns;
        SW_t(2 to 4) <= "011";
        wait for 10 ns;
        SW_t(2 to 4) <= "100";
        wait for 10 ns;
        SW_t(2 to 4) <= "101";
        wait for 10 ns;
        SW_t(2 to 4) <= "110";
        wait for 10 ns;
        SW_t(2 to 4) <= "111";
        wait for 10 ns;
                                            -- CELLO
        SW_t(0 to 1) <= cCELLO;
        SW_t(2 to 4) <= "000";  -- shifting the word whit Shifter_15Bits
        wait for 10 ns;
        SW_t(2 to 4) <= "001";
        wait for 10 ns;
        SW_t(2 to 4) <= "010";
        wait for 10 ns;
        SW_t(2 to 4) <= "011";
        wait for 10 ns;
        SW_t(2 to 4) <= "100";
        wait for 10 ns;
        SW_t(2 to 4) <= "101";
        wait for 10 ns;
        SW_t(2 to 4) <= "110";
        wait for 10 ns;
        SW_t(2 to 4) <= "111";
        wait for 10 ns;
                                                                -- FEPPO
        SW_t(0 to 1) <= cFEPPO;
        SW_t(2 to 4) <= "000";  -- shifting the word whit Shifter_15Bits
        wait for 10 ns;
        SW_t(2 to 4) <= "001";
        wait for 10 ns;
        SW_t(2 to 4) <= "010";
        wait for 10 ns;
        SW_t(2 to 4) <= "011";
        wait for 10 ns;
        SW_t(2 to 4) <= "100";
        wait for 10 ns;
        SW_t(2 to 4) <= "101";
        wait for 10 ns;
        SW_t(2 to 4) <= "110";
        wait for 10 ns;
        SW_t(2 to 4) <= "111";
        wait for 10 ns;
        end process;
```

```
end test;
```