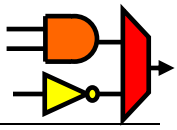


Elettronica dei Sistemi Digitali
Digital Systems Electronics
Lab#2

Switches, Decoders, Numbers and Displays

The purpose of this laboratory session is to learn how to connect previously existing units and other ad-hoc logic circuits. We will use the switches SW_{9-0} on the DE1-SoC board and control a 7-segment display with a specific decoder.

As already said for LAB1 projects, it is required that every single project is always validated by using Modelsim functional simulation and a dedicated testbench, able to generate the necessary input patterns. This validation has to be completed before compiling the code with Quartus Prime and programming the board.

Moreover, starting from this Lab, the timing simulation must be run on the synthesized circuit. This kind of simulation provides the real propagation delays of a circuit and replaces the final test on the board. Follow the guidelines in the Section 5 to know about timing simulation.

Contents:

Abbreviations and acronyms:

IC – Integrated Circuit
LED – Light Emitting Diode
MUX – Multiplexer
VHDL – Very high speed integrated circuits Hardware Description Language
[VHDL cookbook: http://www.onlinefreeebooks.net/engineering-ebooks/electrical-engineering/the-vhdl-cookbook-pdf.html]

1 - Controlling a 7-segment display

Figure 1 shows a *7-segment* decoder module with the three input bits $c_2\,c_1\,c_0$. This decoder drives seven output signals that control a 7-segment display. Table 1 lists the characters that should be displayed for each value of $c_2\,c_1\,c_0$. In order to keep the design simple, only four characters are included in the table (plus the '`blank`' character, which is selected by the four codes $100 - 111$).

The seven-segment in the display are identified by means of the indexes 0 to 6 shown in the figure. Each segment is lit when driven to the logic value '0'. You need to write a VHDL entity that implements the logic functions needed to

activate each of the seven-segment displays, according to the mapping of the table. Use only simple VHDL assignments and write simple Boolean expressions in your code to specify each logic function.

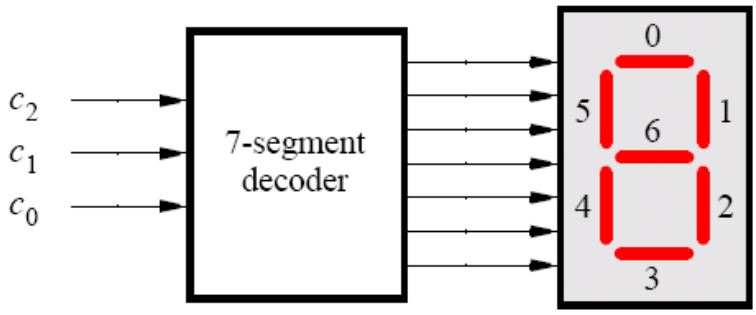


Figure 1 - A 7-segment decoder.

$c_2c_1c_0$	Character
000	H
001	E
010	L
011	O
100	
101	
110	
111	

Table 1 - Character codes.

In particular, you need to do the following steps:

1. **Create a new Quartus Prime project** for your circuit.
2. **Create a VHDL entity** for the 7-segment decoder. Connect the $c_2\,c_1\,c_0$ inputs to switches $SW_{2\text{-}0}$, and connect the outputs of the decoder to the *HEX0* display on the DE1 board. The segments in this display are called $HEX0_0$, $HEX0_1$, . . ., $HEX0_6$, corresponding to Figure 1. Remember to declare the 7-bit port

   ```
   HEX0 : OUT STD_LOGIC_VECTOR(0 TO 6);
   ```

   in your VHDL code in such a way that the names of the outputs match the corresponding names in the ***DE1_SoC.qsf*** file.
3. Create a proper testbench and import the whole project in Modelsim, to validate your code, before the download to the board.
4. After adding the required DE1 board **pin assignments**, compile the project.
5. **Download** the compiled circuit in the FPGA chip.
6. **Test** the functionality of the circuit by toggling the $SW_{2\text{-}0}$ switches and by looking at the 7-segment display.

In your report, describe your circuit (partitioning, Boolean equation, …), comment on how you validated the VHDL model (testbench) and give references to the delivered VHDL source files.

2 – Multiplexing the 7-segment display output

Consider the circuit shown in Figure 2. It uses a three-bit wide 4-to-1 multiplexer to enable the selection of five characters that are displayed on a 7-segment display. By using the 7-segment decoder from section 1, this circuit can display any of the characters H, E, L, O, and 'blank'. The character codes are set according to Table 1. For this project, four words are selected, they are already placed into the design. The output of the multiplexer corresponds to the input of a combinational shifter, the output of the shifter corresponds to the input of the 7-segment display. By mean of SW_{1-0}, we are able to select the right word and the switches SW_{4-2} are used to decide how many positions the word has to shift. The words we are going to show are "HELLO", "CELLO", "CEPPO" and "FEPPO".

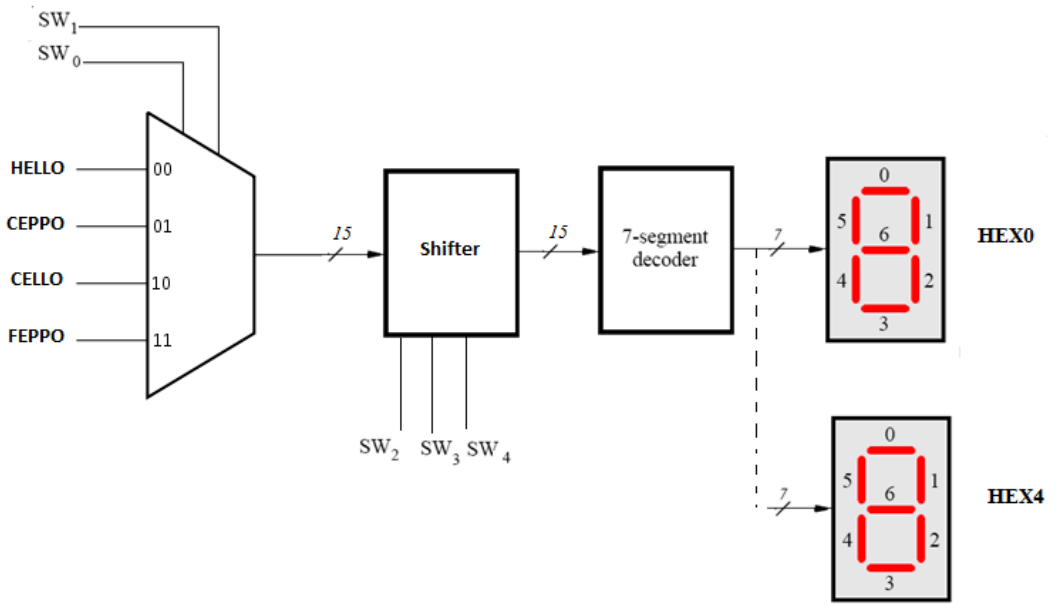
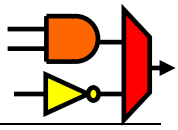


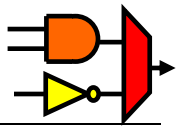
Figure 2 – Driving the 7-Segment display and output letter selection.

An outline of the VHDL code of the circuit is provided below.

```
-- File 7segments_out.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.all;


ENTITY Part2 IS
PORT ( SW : IN STD_LOGIC_VECTOR(4 DOWNTO 0);
HEX0,HEX1,HEX2,HEX3,HEX4 : OUT STD_LOGIC_VECTOR(6 downto 0));
END Part2;
ARCHITECTURE Behavior OF Part2 IS

component mux IS
PORT ( sel: IN STD_LOGIC_VECTOR(1 downto 0);
```



```
output : OUT STD_LOGIC_VECTOR(14 downto 0));
end component;

component 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 component ;

COMPONENT decoder7
PORT ( C : IN STD_LOGIC_VECTOR(2 downto 0);
Display : OUT STD_LOGIC_VECTOR(6 downto 0));
END COMPONENT;
SIGNAL a1,a2 : std_logic_vector (14 downto 0);
BEGIN
MUX0: mux PORT MAP (SW(1 DOWNTO 0),a1);
SHIFT0: shifter PORT MAP (a1, SW(4 downto 2),a2);
H0: decoder7 PORT MAP (a2(2 downto 0), HEX0);
H1: decoder7 PORT MAP (a2(5 downto 3), HEX1);
H2: decoder7 PORT MAP (a2(8 downto 6), HEX2);
H3: decoder7 PORT MAP (a2(11 downto 9), HEX3);
H4: decoder7 PORT MAP (a2(14 downto 12), HEX4);
END Behavior;

-- File mux_3bit_5to1.vhd

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux IS
PORT ( sel: IN STD_LOGIC_VECTOR(1 downto 0);
output : OUT STD_LOGIC_VECTOR(14 downto 0));
END mux;
ARCHITECTURE Behavior OF mux IS
. . . code not shown
END Behavior;

-- File shifter.vhd
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
. . . code not shown
END Behavior;



-- File char 7seg.vhd

LIBRARY ieee;
```



```
USE ieee.std_logic_1164.all;

ENTITY char_7seg IS
      PORT ( C          : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
             Display    : OUT STD_LOGIC_VECTOR(0 TO 6));
END char 7seg;

ARCHITECTURE Behavior OF char_7seg IS
. . . code not shown
END Behavior;
```

You have to use separate *vhd* files for each different module (a file for each Entity-Architecture couple). Try to re-use as much as possible the circuits from previous Labs as sub-circuits of this more complex logic. As an example, the 7-segment decoder designed in the first project can be extended to also cover the additional letters of this project and re-used as component.

You need to extend the given code in such a way that it uses five 7-segment displays rather than just one. You will need to use five instances of each sub-circuit. The purpose of your circuit is to display the selected word, and to rotate it in a circular fashion across the displays, based on the values set with the three switches SW_4 SW_3 SW_2. For example, if the displayed word is HELLO, then your circuit should drive the output patterns as illustrated in Table 2.

SW4 SW3 SW2	Character pattern
000	HELLO
001	ELLOH
010	LLOHE
011	LOHEL
100	OHELL

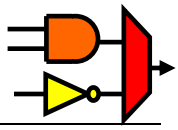
Table 2 - Rotating the word HELLO on five 7-segment displays.

You need to do the following steps.

1. **Create a new Quartus Prime** project for your circuit.
2. **Include your VHDL top-level entity** in the Quartus Prime project as well as the other VHDL files. Connect the switches SW_{1-0} to select the inputs of the shifter. Also, connect SW_{4-2} to the shifter to select how many positions the word has to rotate. Connect the output of the shifter to the 7-segment displays *HEX4*, *HEX3*, *HEX2*, *HEX1*, and *HEX0*.
3. Validate your model by means of a testbench and the Modelsim simulator, before compiling the code with Quartus prime.
4. Include the required **pin assignments** for the DE1 board for all switches, and 7-segment displays. Compile the project.
5. **Download** the compiled circuit in the FPGA chip.
6. **Test** the functionality of the circuit by setting the switches, and observe the rotation of the characters.

As for the previous project, describe in your report the designed circuit and give references to the delivered VHDL source files.

3 – Binary to Decimal converter



You need to design a circuit that converts a four-bit binary number $V = v_3\, v_2\, v_1\, v_0$ into its equivalent two-digit decimal number $D = d_1 d_0$. Table 3 shows the required output values. A partial design of this circuit is given in Figure 3. It includes a comparator that checks when the value of V is larger than 9, and uses its output to control the 7-segment displays. You need to complete the design of this circuit by creating a VHDL entity, which includes the comparator, some multiplexers and circuit *A* (do not include circuit *B* or the 7-segment decoder at this point). Your VHDL entity should have the four-bit input *v*, the four-bit output *m* and the output *z*, like in the following VHDL template:

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

ENTITY converter IS
      PORT ( v     : IN      UNSIGNED(3 DOWNTO 0);
             m     : OUT     STD_LOGIC_VECTOR(3 DOWNTO 0);
             z     : OUT     STD_LOGIC);
END converter;
```

Binary value	Decimal digits	
0000	0	0
0001	0	1
0010	0	2
...	...	...
1001	0	9
1010	1	0
1011	1	1
1100	1	2
1101	1	3
1110	1	4
1111	1	5

Table 3 - Binary-to-Decimal conversion.

Do the following steps:

1. **Create a Quartus Prime project** for your VHDL entity. You can reuse any entity-architecture already developed for previous projects.
2. **Compile the circuit** and use functional simulation to verify the correct operation of your comparator, the multiplexers and circuit *A*.
3. **Comment your VHDL code** and **include circuit *B*** in Figure 3 **as well as the 7-segment decoder**. Here the idea is to have a separate entity for the BCD converter and to reuse the previous VHDL exercises for the upper level architecture. Change the inputs and outputs of your code in such a way that switches ***SW*$_{3-0}$** on the DE1 board are used to represent the binary number *V*. Moreover, use the displays ***HEX1*** and ***HEX0*** to show the values of the decimal digits d_1 and d_0. Make sure you include the required pin assignments for the DE1 board in the project.
4. **Validate** the whole model with Modelsim and a proper testbench.
5. **Recompile the project** with Quartus prime, and then download the circuit into the FPGA chip.
6. **Test your circuit** by setting all possible values of *V* and by looking at the 7-segment displays.

In your report, clearly show the structure of the circuit and the mapping between the allocated blocks and the corresponding VHDL components.

4 – Binary-to-BCD Converter

In section 3 we discussed how it is possible to represent a binary number in a decimal convention by using 7-segment displays. Sometimes this conversion is mandatory because humans are used to refer to decimal representations instead of binary ones. As you can guess, in these circuits each decimal digit is represented by using four bits. This scheme is known as the *Binary Coded Decimal* (BCD) representation. As an example, the decimal value 59 is encoded in BCD form as 0101 1001.

Design a combinational circuit that converts a 6-bit binary number into a 2-digit decimal number represented in the BCD system. Use switches ***SW5–0*** to input the binary number and the 7-segment displays ***HEX1*** and ***HEX0*** to display the decimal number. First, validate your circuit with Modelsm, then implement it on the DE1-SoC board with Quartus Prime and demonstrate its functionality.

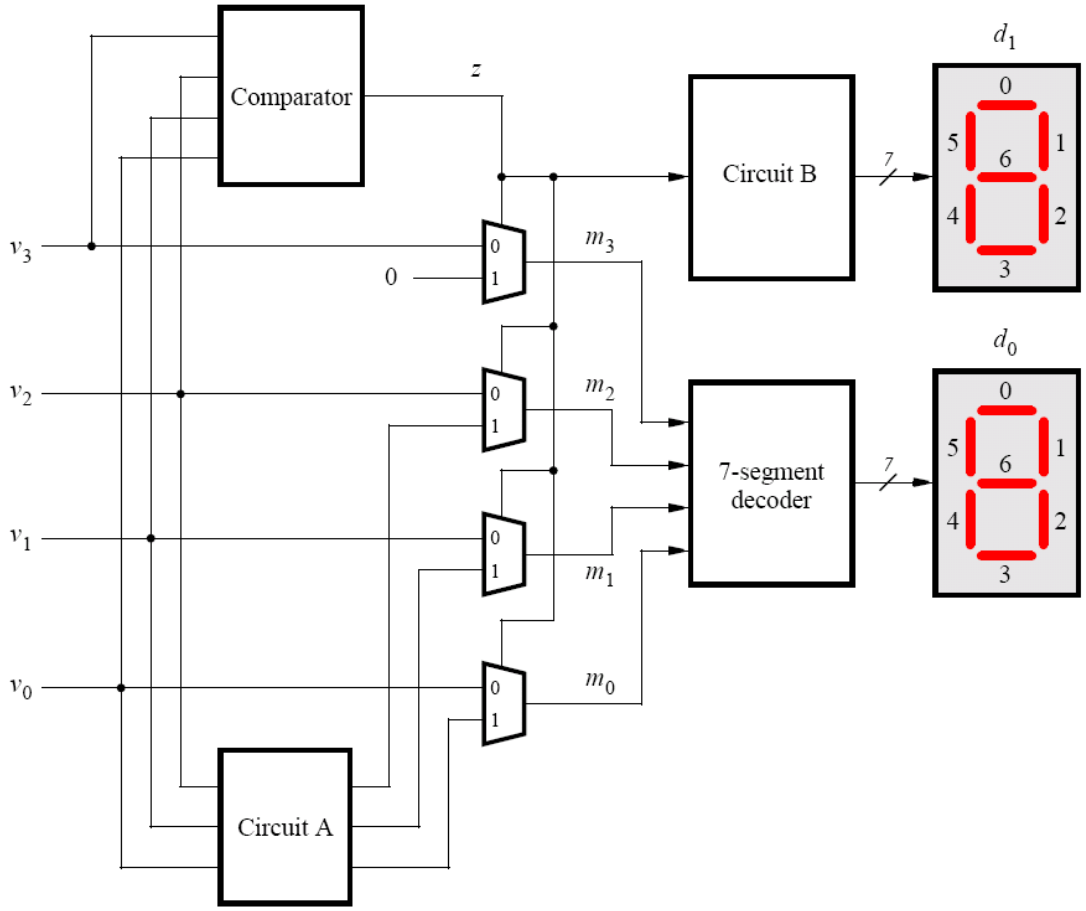
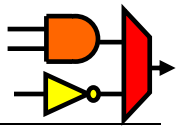


Figure 3 - Partial block scheme of the system that implements the binary-to-decimal conversion circuit.

5 – Appendix – A: Timing simulation

The behavioural simulation simply allows verifying that a circuit generates the correct output sequences for a given input sequence. Therefore, the waveforms obtained in the behavioural simulation are not associated with the real delays of the



circuit. On the contrary, the timing simulation takes into account the true delays of the circuit and therefore it provides its actual performance in terms of speed, propagation delay of clock frequency. Of course, before running a timing simulation, the circuit must be synthesized, thus, the timing simulation is also called post-synthesis simulation.

This Section describes the design flow used to run a timing simulation under Modelsim and Quartus Prime
Unfortunately, the current versions of Quartus Prime do not support the timing simulation for the Cyclone V device family. Consequently, as a preliminary step, we have to synthesize again our circuit by changing the FPGA device to a Cyclone IV device. You could have to install Cyclone IV device if you only installed the Cyclone V family, initially.

To install a new device family, run the Device Installer from the Intel FPGA application menu. This requires a network connection and logging in to your Intel account. After downloading the qdz file from the download center, the Device Installer will make it available for Quartus designs.

Open the project already synthesized for Cyclone V and modify the device from the Assignment menu. Select any device from the Cyclone IV family.

Then, run the synthesis again (Analysis & Synthesis plus Fitter steps). If you have a constraint file, keep it in the new synthesis. Of course, for this synthesis, the pin assignment file we use with the Cyclone V device must be excluded, because it contains pins different from those available in the Cyclone IV FPGA.

Now, on the Assignments menu, click Settings and, in the Category list, select Simulation under EDA Tool Settings. In the Tool name list, specify simulation tool as ModelSim-Altera. As for the Format, select VHDL. Do not turn on Run gate level simulation automatically. Click Apply. Then, under the Simulation window, select More EDA Netlist Writer Settings and check that the Generate Functional Simulation Netlist option is set to Off.

Compile the design again. If not already done together with the synthesis, in the processing menu, point to Start and click on Start EDA netlist writer. Now you have two files in the folder: simulation/modelsim/ in your working directory:
(1) .vho (VHDL Output) file and
(2) .sdo (Standard Delay Format Output) file.
The first one is a detailed VHDL description of the synthesized circuit, while the second file contains the propagation delays across logic resources and interconnects exploited by the software tool to implement the circuit.

In order to run the timing simulation, both files have to be imported in Modelsim. First, open your Modelsim project and remove from the project all VHDL files, except the testbench. Then, add the .vho file generated by Quartus.

Compile your files and verify that no errors are returned. Notice that any file or folder name containing blanks or other unusual characters may generate errors.

Then, in the Simulate menu click start simulation. In the Start simulation window, you have to perform two actions:
(1) Click on the SDF tab and then click add. Find the proper .sdo file created by Quartus in the simulation/modelsim folder, select it and then, in the Apply to Region field, enter the name of the circuit instance that you allocate in your testbench (e.g. UUT).
(2) In the Design tab of the Start simulation window, under work, select the Entity of the testbench.

You can now proceed as usual, by loading the waveforms of the main signals and running the simulation for a given time interval. The resulting waveforms will incorporate the real circuit delays.

The given guidelines are also available from the Intel FPGA web site: https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/mapIdTopics/jka1465596867659.htm.