Politecnico di Torino

DIGITAL SYSTEMS ELECTRONICS

04OIHNX - A.Y. 2024/2025

Prof. G. Masera

Laboratory assignment no. 1 - Multiplexers, Light Emitting Diodes, Switches and Testbench

DUE DATE: 18/03/2025 DELIVERY DATE: 13/03/2025

GROUP 08 - Contributions:

- Daniele Becchero (308299) – 33.3%
- Bohotici Ionut Viorel (300061) – 33.3%
- Simone Viola (310779) – 33.3%

The members of the group listed above declare under their own responsibility that no part of this document has been copied from other documents and that the associated code is original and have been developed expressly for the assigned project.

Introduction

In this laboratory, it was conducted an experience that allowed us to familiarize with simulation software and hardware while also generating our first VHDL code.

The first exercise involved controlling the DE1 board's LEDs using the toggle switches on the board. In particular, the board features 10 switches (SW9-0) and 10 red LEDs (LEDR9-0) that display the states of the switches.

Then the second exercise was related to a 2-to-1 multiplexer. The 2-to-1 multiplexer is a digital circuit that allows you to choose between two inputs (*X*, *Y*) by routing the selected signal to the output based on the value of a control signal. In practice, if the select signal *s* is 0, the output reproduces the value of input *X*; if *s* is 1, the output takes the value of input *Y*.

The third exercise was related to a 5-to-1 multiplexer. The 5-to-1 multiplexer is a combinational circuit that allows you to select one among five inputs, which we can denote, as *U*, *V*, *W*, *X*, and *Y*, and pass it to the output *M*. In the case of a multiplexer with five inputs, at least 3 bits of selection (*S2*, *S1*, *S0*) are necessary since there are $2^3 = 8$ possible combinations, even though not all are used (in this component, the values from 0 to 4 (total 5) are defined to choose each input, while the combinations from 5 to 7 may be considered unused or handled as default cases).

Submitted material

In the folder delivered, there are three different folders (one for each exercise). In each folder, there are two other folders named “VHDL” and “Synthesis”. The first includes the VHDL files for the entity (or entities) and the device's testbench. The other one includes the assignment file for the DE1-SoC FPGA board and the *.sof* file coming from the compilation of the project (just the entities, not the testbench). This last file is the one which is ready to be uploaded into the board to be programmed, without the need for creating a new project, select the right device, import the pin assignments and the other long tasks.

Section 1: Controlling the LEDs

For a first approach with VHDL and the DE1-SoC board, we are asked to control the LEDs of the board with the corresponding switch. The objective is to write the VHDL code and test its functionality by using an appropriate testbench and a simulation software. Once the functionality has been validated, we are asked to use Quartus Prime to compile and synthesize the code and fit the design for our device.

Design entry

First, we have written the VHDL code for our circuit. Our code, *Lab1_1.vhd*, is the same that has been suggested in the assignment. The entity *part1* has 10 input ports and 10 output ports that store their logical value in two vectors. The behavioural description of the circuit is simply the assignment of the input values that comes from the switches to the output values, as it was suggested:

```
LEDR <= SW;
```

Functional simulation

In order to test the functionality of our code, we have written the following VHDL code that implement the testbench, named *Lab1_1_tb.vhd*. The testbench is an entity without any I/O ports. Its architecture includes the signals INPUTS and OUTPUTS that have the same dimensions as the input-outputs signals of the *part1* entity. Its behavioural description includes just one process in which the INPUT values change every 20 ns. We have chosen to try four different combinations of input values, for a total simulation time of 80 ns. The PROCESS statement of the testbench is shown below.

```
process
begin
     INPUTS <= "0011001100";
     wait for 20 ns;
     INPUTS <= "1100110011";
     wait for 20 ns;
     INPUTS <= "1010101010";
     wait for 20 ns;
     INPUTS <= "1111100000";
     wait;
end process;
```

The last thing to do is to connect the signals INPUTS and OUTPUTS (created specifically for simulation purposes) to the ports of the device under test (DUT), which in this case is just *part1*. This assignment is made by using the PORT MAP statement:

```
DUT : part1 port map(SW => INPUTS, LEDR => OUTPUTS);
```

Using Modelsim, we have created a new project (*Lab1_1_MODELSIM*) that includes the mentioned VHDL files and then we set up a simulation. The waveforms are shown below.

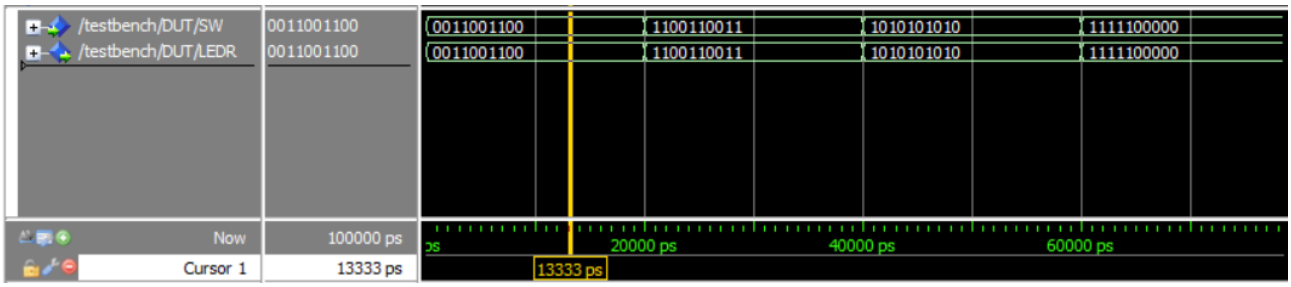


Figure 1 -Ex.1 waveform screen capture

For every combination of SW, the same pattern is copied to LEDR. Hence, the VHDL code of *part1* is correct and it is ready to be uploaded to the board.

Synthesis

Once the correct functionality has been tested, we load our VHDL file in a new Quartus Prime project and run the compilation with "Cyclone V 5CSEMA5F31C6N" as destination board. For the synthesis, we used only the VHDL file of *part1* entity, because the testbench is needed only for the validation of *part1*, and moreover the testbench is not synthesizable. This action compiles the code and generates the files for the programming of the DE1-SoC board. We have also included the assignment file in the Quartus project, which automatically maps the pins of the FPGA that are connected to the LEDs and the switches. The compilation takes a few minutes and once finished we have tried to program our device. At this point we have had some difficulties because we did not understand well how to select the correct device in the programmer window. We asked the instructor for help, then we finally managed to correctly program the device. At this point we have tried several combinations of the switches, and we have checked that, for every combination, the corresponding LED was switched on.

Section 2: 2-to-1 Multiplexer

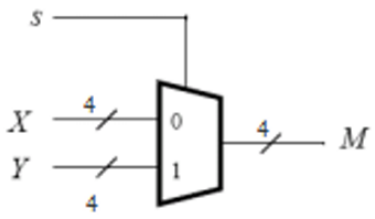


Figure 2 - 4-bit wide 2-to-1 multiplexer

This section is about the realization of a four-bit wide 2-to-1 multiplexer.

Design entry

The VHDL code for this component (named *2to1mux.vhd*) is very similar to the examples shown in lectures. First, a new entity is created, named *mux21*. This entity has nine input ports (one bit for selection and two 4-bits input for the two data inputs of the multiplexer) that will be connected to the switches and four output ports that will be connected to the LEDs. The data type is a standard logic vector of appropriate size (nine elements for the input and four elements for the output). For the architecture definition, we are asked to use multiple statements like the one below:

```
m <= (NOT (s) AND x) OR (s AND y);
```

To accomplish this, we need to write a dataflow-style description of the architecture. At the top of the architecture, four new signals have been defined to enhance comprehension of the following code. Hence, the first thing to do is to connect these signals to the input/output signals coming from the ports of the entity, as you can see below.

```
X    <= SW(3 downto 0);
Y    <= SW(7 downto 4);
s    <= SW(8);
LEDR <= M;
```

Once the new signals have been declared and initialized, they can be used in the architecture description of the entity previously declared. The following lines of code assign the value of the selected input to the output. Then, the value of the output of the mux can be viewed on the first 4 on-board LEDs.

```
M(0) <= (not (s) and X(0)) or (s and Y(0));
M(1) <= (not (s) and X(1)) or (s and Y(1));
M(2) <= (not (s) and X(2)) or (s and Y(2));
M(3) <= (not (s) and X(3)) or (s and Y(3));
```

Functional simulation

For this circuit the testbench is very similar to the previous one. First, we have declared an empty entity. Then, in the architecture section, we have included our component named *mux21* and four new signals for simulation purposes. The behaviour of the testbench is set declaring a process in which the inputs and/or the input selector change every 20 ns. Again, we have chosen four different combinations for a total simulation time of 80 ns. The process statement is shown below.

```
process
begin
     A   <= "0101";      -- Set A to the value 0101
     B   <= "1111";      -- Set B to the value 1111

     SEL <= '0';         -- Set SEL to the value 0
     wait for 20 ns;
     SEL <= '1';         -- Change SEL
     wait for 20 ns;
     A <= "1010";        -- Change A (no change in M)
     wait for 20 ns;
     B <= "0000";        -- Change B
     wait;
end process;
```

The waveforms results are shown below. The functionality of *mux21* is correct.

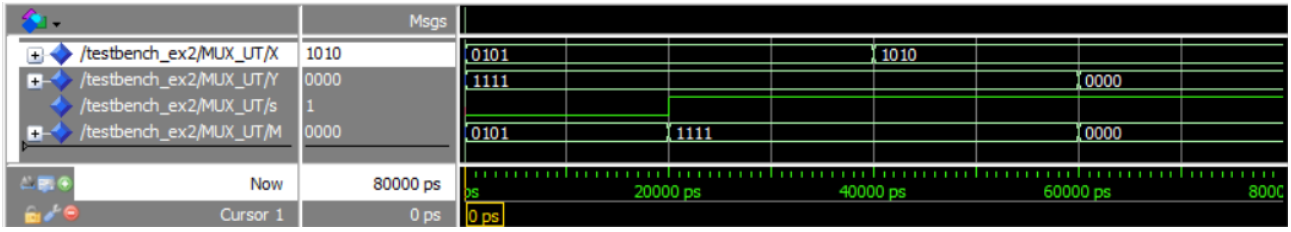


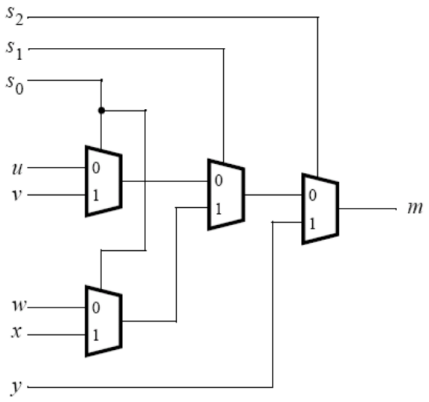
Figure 3 - Ex 2 waveform screen capture

Synthesis

After we have completed the simulation, the code is ready to be loaded into the FPGA. In order to test our circuit, the following component have been used: the switch 8 works as the *s* input, while the switches 0 to 3 are used as the *X* input and the switches 4 to 7 are used as the *Y* input. The output *M* of the multiplexer is connected to the 0 to 3 LEDs.

With the same process discussed in the previous section, we have created a new project in Quartus II, we have opened the VHDL file of *mux21* and then we have compiled it for our specific device. Again, after several minutes the compilation was done, and we were ready to program the FPGA. This time all the process was quicker because we have avoided all the problems we had in the first section. We tried many combinations of inputs and input selection, and we checked that our circuit works fine, following the specifications.

Section 3: 5-to-1 Multiplexer



s_2	s_1	s_0	m
0	0	0	u
0	0	1	v
0	1	0	w
0	1	1	x
1	0	0	y
1	0	1	y
1	1	0	y
1	1	1	y

Figure 4 - Logical structure and truth table of 3-bits wide 5-to-1 mux

The third and final section focuses on the implementation of three-bit wide 5-to-1 multiplexer. The Figure 4 displays the circuit of a 5-to-1 MUX, which is intended to be implemented in VHDL. It consists of four 2-to-1 MUXs, each with 3-bit inputs. The two MUXs on the left use the LSB of the selection signal, the middle MUX uses the next bit, and the rightmost MUX uses the MSB of the selection sequence. The corresponding truth table is shown above.

Design entry

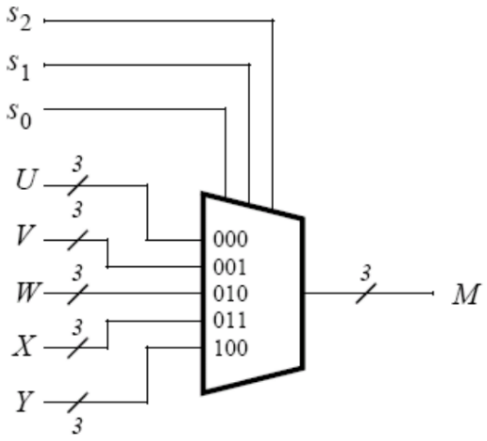


Figure 5 - Compact representation of 3-bits wide 5-to-1 mux

By analysing the circuit in Figure 4, we notice that, for simplicity, we can model the circuit with the four 2-to-1 MUXs using a single symbol representing a 5-to-1 MUX, shown in the Figure 5. Observing the component, we immediately notice the presence of six inputs, each 3 bits wide, and a single output M, also a 3 bits wide. Since the signals U, V, and W are required to be constant with a fixed values of “101”, “010”, and “111” respectively, we do not need to implement switches to simulate them on the DE1-SoC board. Instead, we need to use three switches, one for each bit of the selection, three switches for the bits of input signal X, and another three for Y. For this reason, a 9-bit wide SW had been initialized. Since the output M is 3 bits wide, we can visualize its state using 3 LEDs on the board.

First, we have identified the inputs (U, V, W, X, Y, Sel) and the single output (M), which are all three-bit wide. As stated in the assignment, in the definition of inputs and outputs, a 9-bit wide

SW has been initialized. The bits having indices from 8 to 6 are reserved for the 3-bit selection (S2, S1, S0), the bits having indices from 5 to 3 are reserved for setting the value of Y, and the remaining bits having indices from 2 to 0 are reserved for setting the value of X externally. We can now proceed with writing these initial details in the entity definition of this component, named *mux5to1*.

```
entity mux5to1 is
     port (
          SW : in STD_LOGIC_VECTOR(8 downto 0);
          LEDR : out STD_LOGIC_VECTOR(2 downto 0)
     );
end mux5to1;
```

First, we have identified the inputs (U, V, W, X, Y, Sel) and the single output (M), which are all 3-bit wide. The bits having index from 8 to 6 are reserved for the 3-bit selection (S2, S1, S0), the bits having index from 5 to 3 are reserved for setting the value of Y, and the remaining bits from index 2 to 0 are reserved for setting the value of X externally. Now that the inputs and output signals, along with their respective sizes, have been defined, we proceed to describe the functionality of the circuit, referring to the truth table derived from the initial representation of this circuit. As you can see below, the encoding of the inputs U, V, W which have fixed values, does not read external values from the switches.

```
architecture Behavioral of mux5to1 is
     signal M : STD_LOGIC_VECTOR(2 downto 0);
     signal Sel : STD_LOGIC_VECTOR(2 downto 0);
     signal X : STD_LOGIC_VECTOR(2 downto 0);
     signal Y : STD_LOGIC_VECTOR(2 downto 0);
begin
     LEDR <= M;
     process (Sel, SW, X, Y)
     begin
          Sel <= SW(8 downto 6);
          X <= SW(2 downto 0);
          Y <= SW(5 downto 3);

          case Sel is
               when "000" => M <= "101"; -- U
               when "001" => M <= "010"; -- V
               when "010" => M <= "111"; -- W
               when "011" => M <= X;
               when "100" => M <= Y;
               when others => M <= Y;
          end case;
     end process;
end Behavioral;
```

As observed, different equivalent syntaxes can be used to describe all possible combinations of the selection bits. In this case, the *"case-when"* syntax was chosen because it is more concise and allows for an easy representation of all combinations from "100" onward with a single command *"when others"*.

Functional simulation

The code used as the testbench for this circuit is very similar to the one just described and to that of the 2-to-1 MUXs in Section 2. It begins by declaring an empty entity, as seen during lecture. Then, the architecture of the component to be tested is described, defining the inputs and outputs as already detailed in the previous paragraph.

Next, the signals to be simulated in this test are specified. The key part of the testbench lies in the final section, namely the "process", where the initial values of the inputs are set. Immediately after, different combinations of the selection signal bits, as represented in the truth table, are assigned one at a time at 10 ns intervals.

```
uut : mux5to1 port map(
     SW   => SW,
     LEDR => LEDR
);

-- Stimulus process
stim_proc : process
begin
     -- Test first input "101"
     SW <= "000000000"; -- Sel = "000", M = U = "101"
     wait for 10 ns;
     -- Test second input "010"
     SW <= "001000000"; -- Sel = "001", M = V = "010"
     wait for 10 ns;
     -- Test third input "111"
     SW <= "010000000"; -- Sel = "010", M = W = "111"
     wait for 10 ns;
     -- Test fourth input (X = SW(2:0))
     SW <= "011010101"; -- Sel = "011", M = X = "101"
     wait for 10 ns;
     -- Test fifth input (Y = SW(5:3))
     SW <= "100010101"; -- Sel = "100", M = Y = "010"
     wait for 10 ns;

     -- Test unexpected selections
     SW <= "101010101"; -- Sel = "101", M = Y = "010"
     wait for 10 ns;
     SW <= "110010101"; -- Sel = "110", M = Y = "010"
     wait for 10 ns;
```

```
        SW <= "111010101"; -- Sel = "111", M = Y = "010"
        wait for 10 ns;

        wait;
    end process;
```

For the simulation on the test bench, the values of the SW vector bits from 8 to 6 varies according to the order of the combinations in the truth table, while the bits from 5 to 3 are kept at zero to test the fixed inputs U, V and W. Subsequently, when the selectable external inputs Y and X need to be output, test values are set with bits 5-3 as “010” for Y and bits 2-0 as “101” for X.

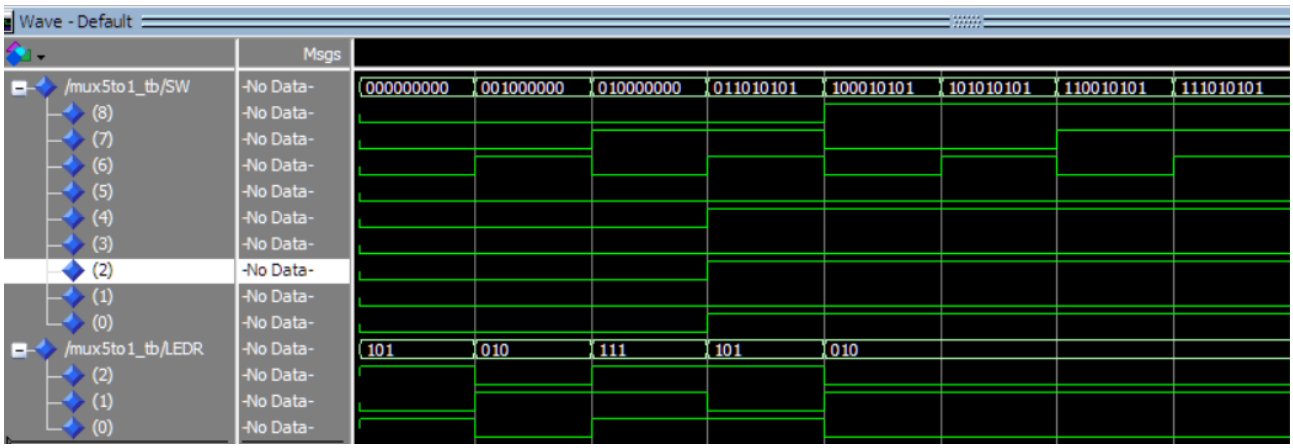


Figure 6 - Ex 3 waveform screen capture

By running the simulation and observing the waveforms of the inputs and the output, it is possible to verify that the circuit operates correctly and that the VHDL code accounts for all possible input combinations.

Synthesis

Following the same procedure described in the previous sections, we opened the VHDL file for the mux5to1 in Quartus and initiated its compilation. After a few minutes, the compilation was successfully completed, allowing us to proceed with programming the FPGA. This time, the process was faster as we avoided the issues encountered in the first and second sections, and we were able to confirm that our project works as intended.

Conclusions

Our implementation of these three exercises in VHDL code yielded results consistent with the expected ones, both from the testbench simulation on ModelSim and the practical verification on the DE1-SoC FPGA board. The skills acquired from this lab allowed us to become familiar with the Quartus and ModelSim software, as well as the process of loading the VHDL file onto the FPGA in the lab, identifying errors and debug them to resolve issues.

The primary challenges faced during the first exercise were not connected to the simulation itself but rather to transferring the file to the board. These issues stemmed from our lack of familiarity with the software, which led to difficulties in correctly importing the .qsf file required to map the LEDs to the switches.

Sources and notes

For all the VHDL files provided, an online tool has been used to format the code to ensure coherence between all the different files. The tool is free to use, and it is available at this link: VHDL Beautifier.

The VHDL files were developed using course materials produced by Prof. G. Masera (available on the course webpage) and the following eBooks recommended by the professor:

1. Mealy, Bryan, and Fabrizio Tappero. *Free Range VHDL*. 2016.
2. Ashenden, Peter J. *The VHDL Cookbook*. 1990.