Synthesizable VHDL Design for FPGAs

Eduardo Augusto Bezerra
Djones Vinicius Lettnin

Universidade Federal de Santa Catarina
Florianópolis, Brazil

August 2013

Contents

Chapter 4. Multiplexer and Demultiplexer

In this chapter components for building decision maker circuits are introduced. The reader will be guided through the design of multiplexer components in VHDL using only logical gates (i.e., structural level), and also in a higher level of abstraction using *when/else* statements (i.e., behavioral level). As a case study the hierarchical design described in Chapter 3 will be re-used and modified in order to include a multiplexer. At the end of the chapter, the students should be able:

- to understand the concept of multiplexer and demultiplexer;
- to implement a multiplexer in VHDL using only Boolean functions;
- to implement a multiplexer in VHDL using *when* / *else* statements;
- to design, simulate and prototype a proposed case study using an FPGA board.

4.1 Theoretical Background

A multiplexer (Mux) component has several inputs, a control signal, and one output. The output receives the value present in one of its inputs, according to the control signal. The block diagram of a general mux in Figure 4.1(a) has *n* input signals (*I*) and *m* control signals (*S*). A 2^n inputs mux, needs *n* control signals.

The mux component in Figure 4.1(b) has four 1 bit inputs A, B, C and D, set to the values 0, 1, 1, and 0, respectively. The selection is set to allow the third input to be connected to the output and, consequently, the F signal presents the value 1. Every time C has its input value changed, F will also be changed. To connect another input to the F output, it is necessary to change the selection signal.

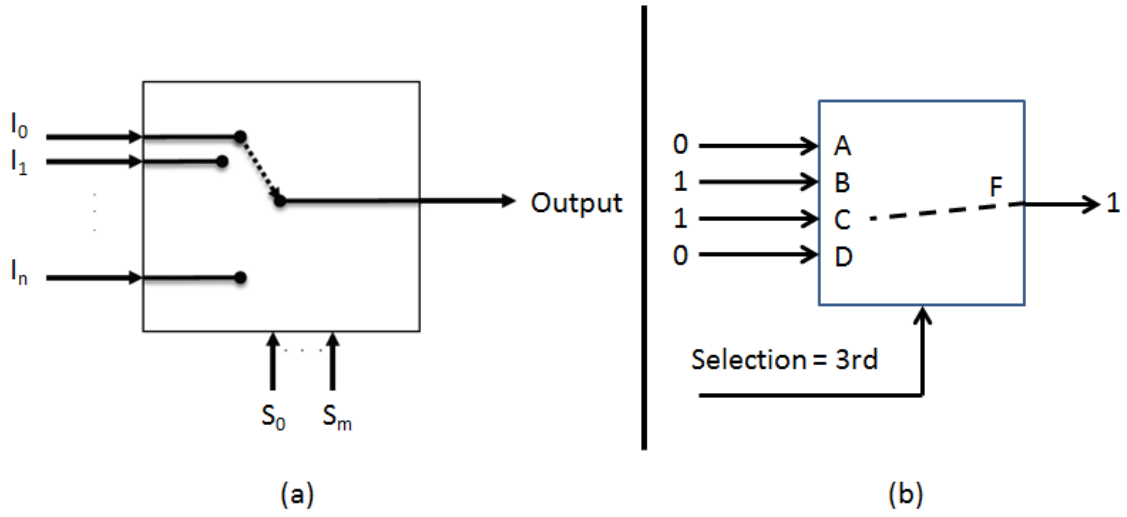


Figure 4.1. Mux component block diagram. (a) n:1 mux. (b) 4:1 mux.

A mux circuit, also known as “data selector”, only allows one input to be available at its output at a time, according to the control signals (“Selection” in Figure 4.1). As examples of applications, this component can be used for data selection, for data routing, in parallel to serial conversions, and also to implement truth tables.

A demultiplexer (Demux) circuit, also known as “data distributor”, has the opposite function of the multiplexer, that is, it allows an input to be available at only one output at a time. This decision is also defined by control signals. The block diagram of a general demux shown in Figure 4.2(a) has *n* output signals (*O*) and *m* control signals (*S*). A 2n outputs demux needs *n* control signals. In Figure 4.2(b) the value 0 is provided to input A, and the control signal “Selection” is set to the second output. As a result, the output F2 is connected to the input A, providing also the 0 value. If the selection is changed, for instance, to the first output, so F1 will also provide the 0 value. When the A input receives a new value, the corresponding output will also be updated.

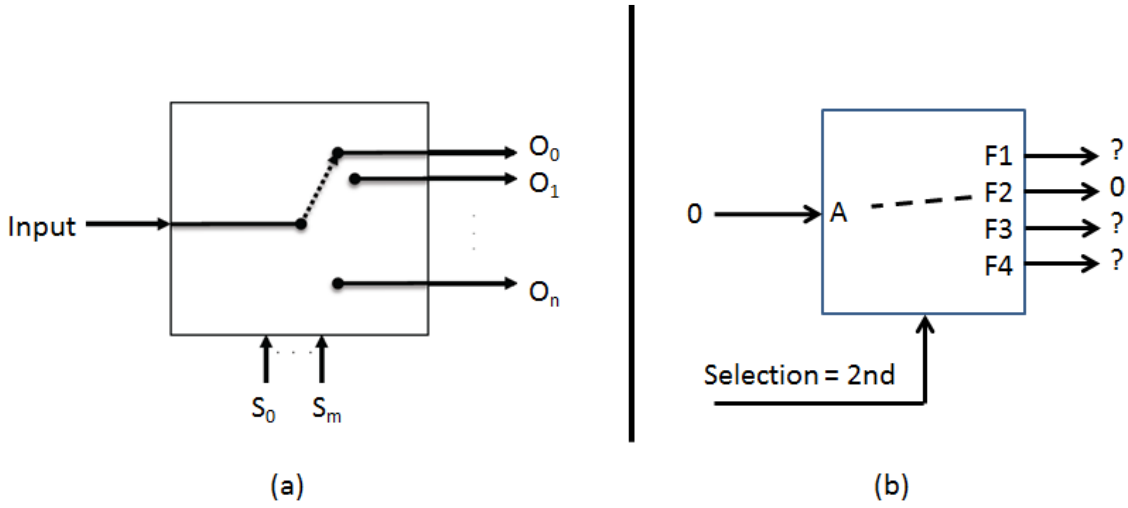


Figure 4.2. Demux component block diagram. (a) 1:n demux. (b) 1:4 demux.

The 2:1 multiplexer (or 2:1 selector) has the function to select one of its two inputs, making the selected entry appear on its output. In Figure 4.3, if selector *s* is equal to 0, then *m* output receives the *x* input. If *s* = 1, then the *m* output receives the *y* input.

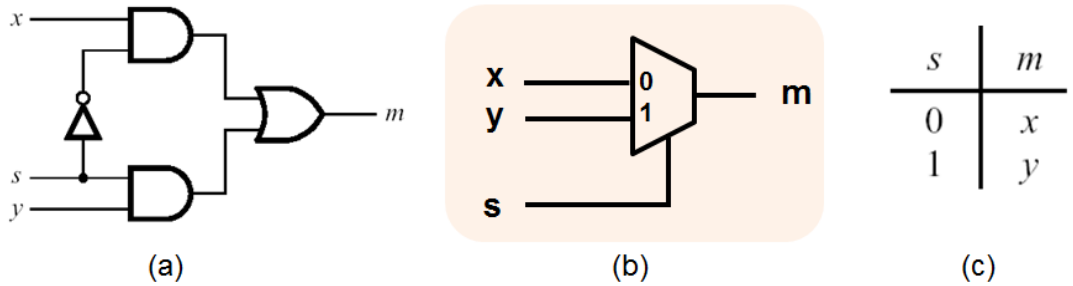


Figure 4.3. 2:1 multiplexer. (a) schematic diagram. (b) block diagram. (c) truth table.

This multiplexer can be implemented in VHDL using only Boolean functions (i.e., structural VHDL) as well as using *when/else* statements (i.e., behavioral VHDL). In Figure 4.4, in the structural VHDL version only logic gates are used to implement the functionality described in the truth table shown in Figure 4.3(c). In the algorithmic version, the behavior of the truth table is modeled using the *when/else* VHDL statements.

Structural VHDL (logic gates):

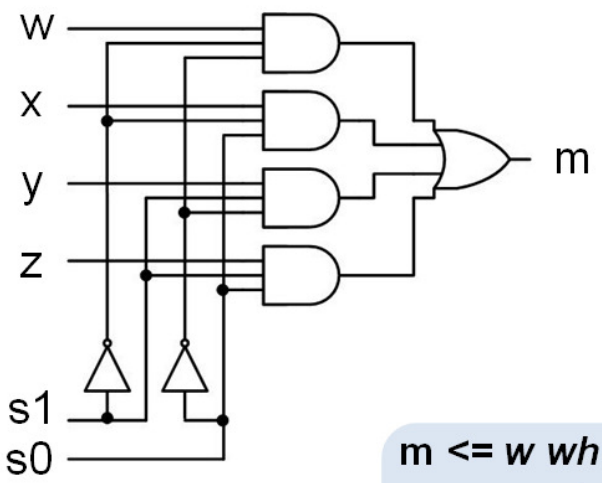
```
m <= (NOT (s) AND x) OR (s AND y);
```

Behavioral VHDL (algorithm):

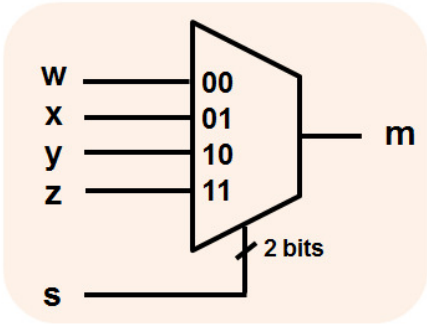
```
m <= x when s = '0' else
     y;
```

Figure 4.4. 2:1 multiplexer in VHDL.

The 4:1 multiplexer selects one of its four inputs, making the selected entry appear on its output. In a four inputs mux, its selector must be 2 bits long. In Figure 4.5, if selector *s* is equal to "00", then the *m* output receives the *w* input. If *s* = "01", then the *m* output receives the *x* input, and so on. In Figure 4.5(a), the 2 bits of the *s* selector are represented explicitly by S0 and S1. In Figure 4.5(b) and Figure 4.5(c), the *s* symbol is used to represent the 2 bits needed to select one of the four inputs.



(a)



(b)

```
m <= w when s = "00" else
     x when s = "01" else
     y when s = "10" else
     z;
```

(c)

Figure 4.5. 4:1 multiplexer. (a) schematic diagram. (b) block diagram. (c) truth table.

4.2 Laboratory Assignment

The laboratory objectives are:

- to implement a multiplexer in VHDL;
- to learn and to practice the use of *when* / *else* VHDL statements;
- to add a multiplexer to the circuit of the case study design.

Laboratory Session

In this laboratory session the reader will have the opportunity to design a multiplexer in VHDL in two levels of abstraction: structural; and behavioral. At the structural level, the multiplexer is written in VHDL using only logic gates (see Chapter 1). At the behavioral level, the *when* / *else* statements in VHDL are used in a more algorithmic approach to the design of digital systems. Figure 4.6 shows the two versions to be implemented. In version I, the two bits of the selector input are explicated presented, as the figure shows the internal gate connections of the multiplexer. In version II, the multiplexer is described using the *when* / *else* VHDL statements, in a higher level of abstraction, with no information regarding the circuit's internal gate connections.

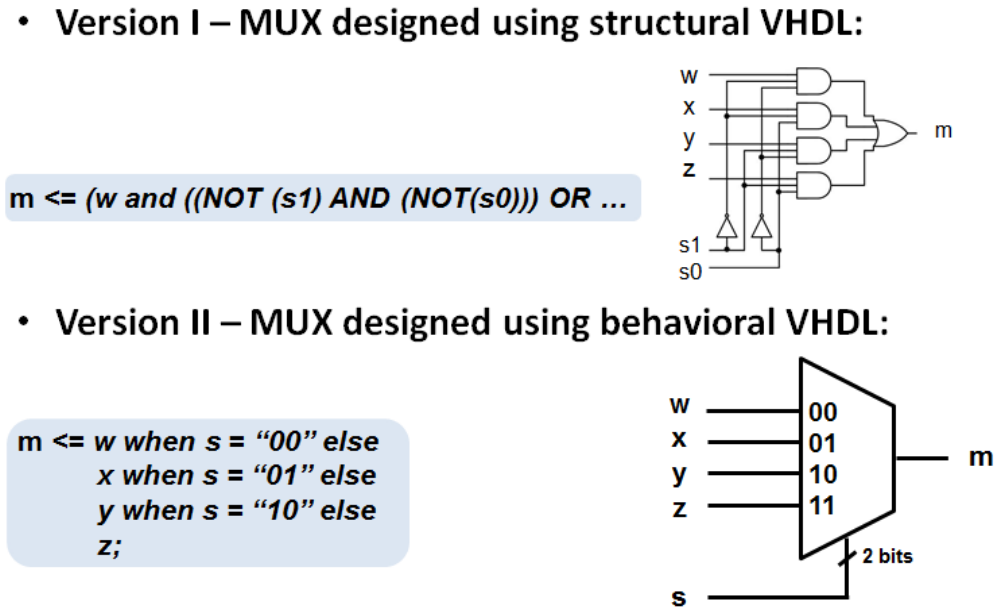


Figure 4.6. Multiplexer described in two abstraction levels.

The case study discussed in previous chapters is reused, but with the C4 component shown in Figure 3.6 replaced by a 4:1 multiplexer component. In the circuit shown in Figure 4.7 and in Figure 4.8, the multiplexer is used to direct one of its inputs F1, F2 or F3 (w, x, y, z), to the F output (m). Thus, LEDR(0) will light on when the 2 bits selection input is set to a position whose component C1, C2 or C3 provides an '1' logic on its output.

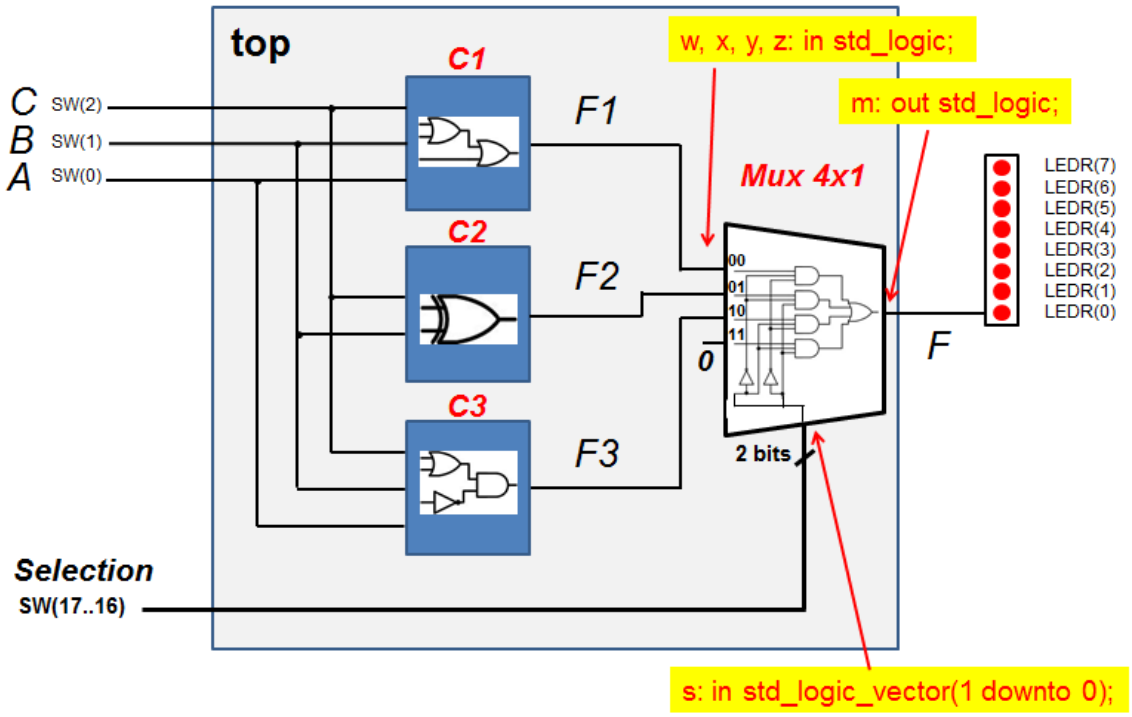


Figure 4.7. Using a multiplexer instead of C4, version I – structural VHDL.

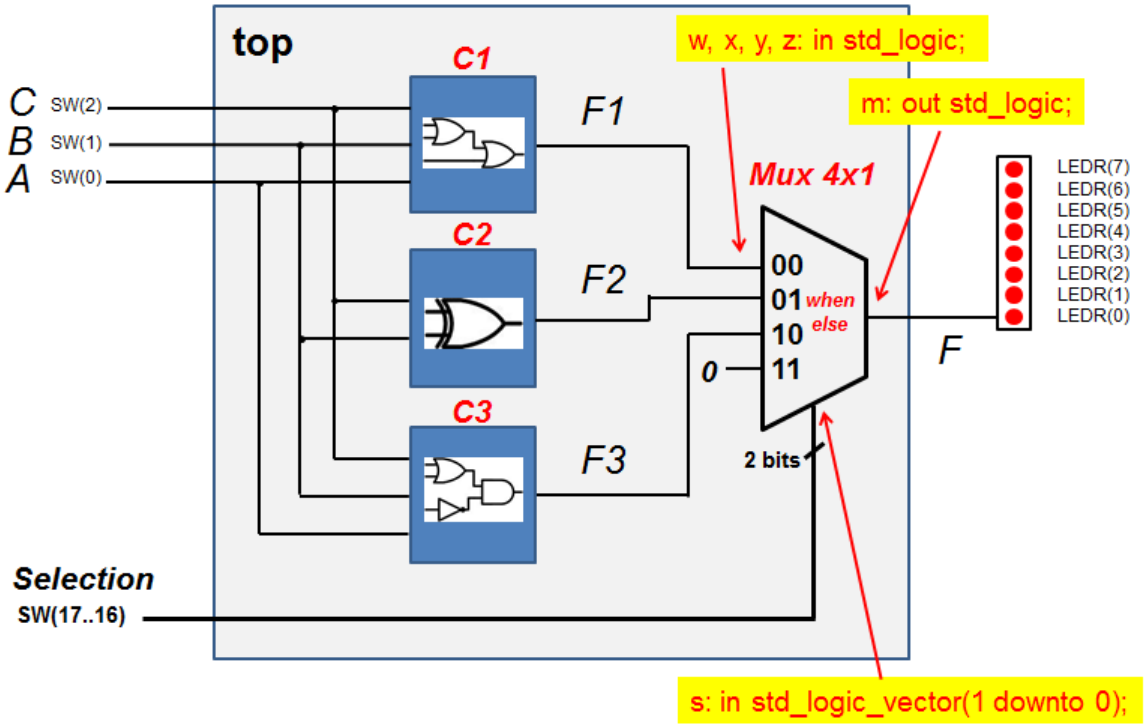


Figure 4.8. Using a multiplexer instead of C4, version II – behavioral VHDL

For instance, when the selection is set to "00" (SW_{17} and SW_{16} are switched off), the F output receives F1. In this case, if one of the three inputs C1, C2 or C3 is switched on, so C1 will evaluate to '1', and LEDR(0) will light on. In Figure 4.9, the selection bits are represented by switches SW_{17} and SW_{16}, input C is SW_2, input B is SW_1, input A is SW_0, and output F is LEDR(0). Table 4.1 shows all possible combinations for the multiplexer selection bits, and the corresponding output in LEDR(0).

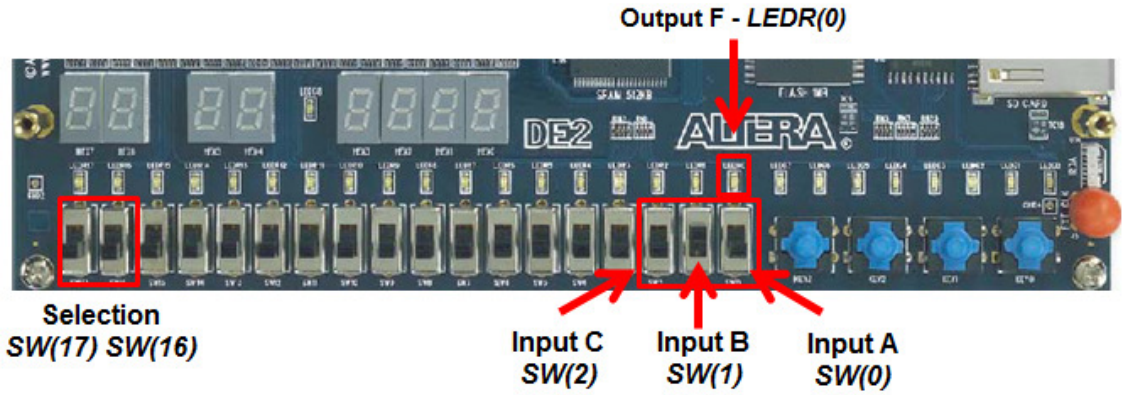


Figure 4.9. User interface – MUX input selection.

Table 4.1. Mux input selection and corresponding outputs.

Input $SW_{17..16}$	Output $LEDR_0$
0 0	F1
0 1	F2
1 0	F3
1 1	0 (LED off)

In this laboratory assignment, two versions for the multiplexer based design are to be implemented. Version I, i.e., multiplexer implemented in structural VHDL is shown in Figure 4.7. In version II, shown in Figure 4.8, the multiplexer is to be implemented in behavioral VHDL. Therefore, in this laboratory assignment, two different VHDL designs should be created. The tasks to be completed in this laboratory session, using Altera's Quartus II EDA tool are as follows:

- Create a project with all the components shown in Figure 4.8 (last chapter components can be reused);
- Create a MUX component using the VHDL design entry editor – two MUX components should be developed, one of them using a gate level VHDL description, and another one using a behavioral (algorithmic) VHDL description;
- Modify the top-level component provided in Figure 3.5, in order to replace C4 by the new MUX component;
- Perform the synthesis;
- Perform the simulation and fix errors, if any;
- Prototype and test the circuit in the FPGA board.

Version I – Multiplexer in structural VHDL

Step 1 – Creating a new project

This step is essentially the same as "Step 1" described in previous chapters. Just remember to use "top" as the project's name (see Figure 1.13), as this is the name of the top-level entity. Remember also to create the folder for this new project in a place such as c:\LabSessions\mux4x1, where "mux4x1" is the name of the new folder.

Step 2 – Adding VHDL files to the project and creating the mux component

Using File Explorer (formerly known as Windows Explorer), or any other file manager program, locate the folder where the five VHDL files developed in last chapter are located (c1.vhd, c2.vhd, c3.vhd, c4.vhd, and top.vhd), and copy all files to the new folder c:\LabSessions\mux4x1. This procedure is shown in Figure 4.10.

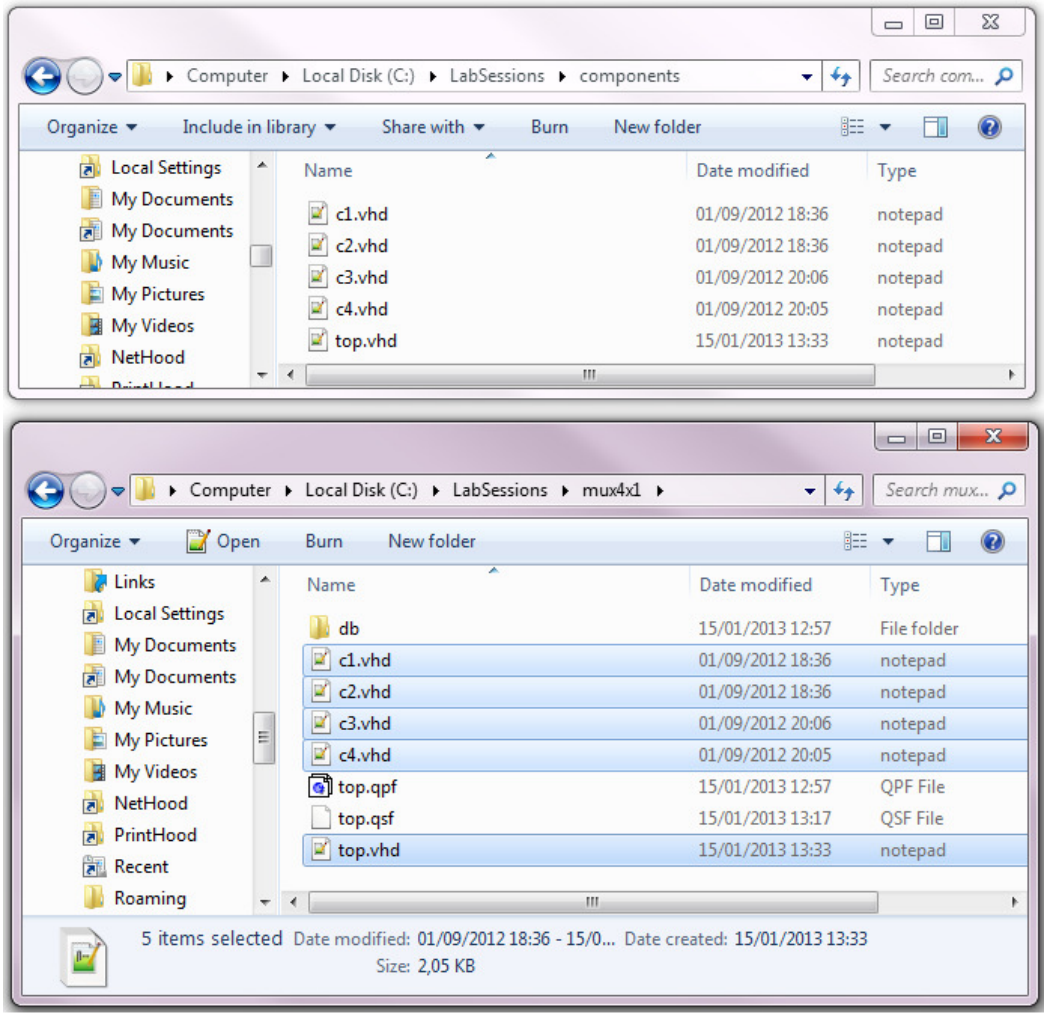


Figure 4.10. Copying files using File Explorer.

In Quartus II, choose *Project –> Add/Remove Files in Project*, as shown in Figure 3.8. A window will open allowing c1.vhd, c2.vhd, c3.vhd, c4.vhd and top.vhd, to be imported into the project. In this window, after adding the files press the *Apply* button.

In the Project Navigator window, select the Files tab, and double-click on c4.vhd in order to open it in the text editor. Make the following changes in c4.vhd, in order to turn it into a multiplexer component:

- Change the entity's name from "C4" to "Mux" (three changes are needed);
- Change the architecture's name to "mux_stru";
- Add the new entity's signals w, x, y, z, s, and m, as shown in Figure 4.7;
- Replace the expression F <= (A and B) or C), by the multiplexer described in Figure 4.6, version I;

Chose *File –> Save as*, and rename c4.vhd to mux4x1.vhd. The file will be saved, also in the default location, in the project folder defined in Step 1. Be sure that the check box "Add file to current project" is checked. At this point it is important to be sure that the entity's name is "mux" otherwise the synthesis tool will not be able to find the entity to be synthesized.

A mux4x1.vhd component should appear in the Files tab of the Project Navigator window. As shown in Figure 4.11, the old c4.vhd component can be removed from the project by right-clicking on it.

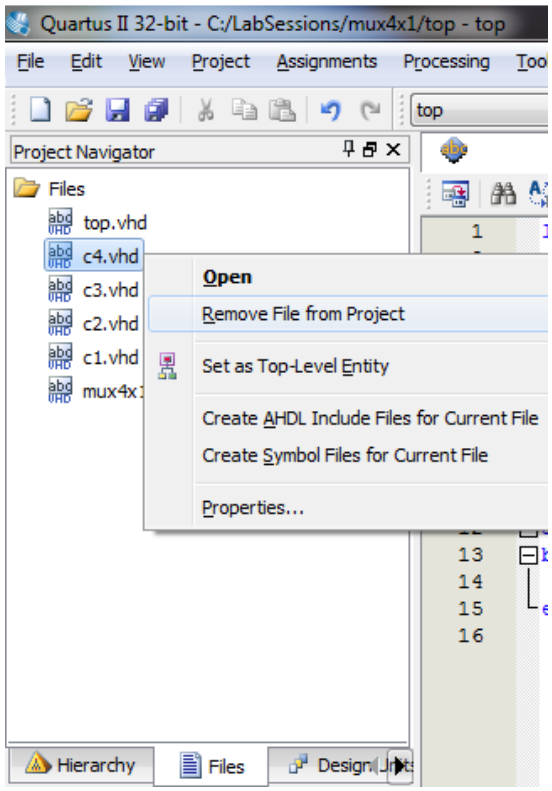


Figure 4.11. Removing c4.vhd from the project.

Step 3 – Editing top.vhd to add Mux to the circuit

As shown in Figure 3.10, select *Project Navigator* (1), *Files* (2), and double-click on top.vhd (3). This will open the top-level component in the VHDL text editor. In Figure 3.5, remove C4 component (lines 28 to 33), and *copy & paste* the entity of the new Mux. Make the necessary modifications in order to have the wright syntax for component definition in VHDL (e.g., replace "entity" by "component", …).

Next, remove line 41 (C4), and create an instance of Mux using the port map statement. Use the port map declarations for components *C1*, *C2* and *C3* as examples, but be careful to make the correct connections according to Figure 4.7. Notice that in Figure 4.7, F1, F2 and F3 are connected to *w*, *x* and *y*, respectively. The *z* input is connected to a '0' constant. The *s* input (mux selection) is connected straight to DE2 signals SW_{17} and SW_{16}. The *m* output is connected to DE2 LEDR(0) signal. All these connections should be performed by the port map statement.

Step 4 – Synthesis

To generate the configuration file (bitstream) to be downloaded to the FPGA board, the first step is to import the *DE2_pin_assignments.qsf* file, as discussed in Chapter 2, Step 5, and shown in Figure 2.7. To perform the synthesis, use the Compile button in Quartus II menu. Fix any errors pointed out by the synthesis tool.

Use the Quartus II tool "RTL viewer" to check if the generated circuit is as expected. Figure 4.12 shows the circuit's block diagram generated by Quartus II.

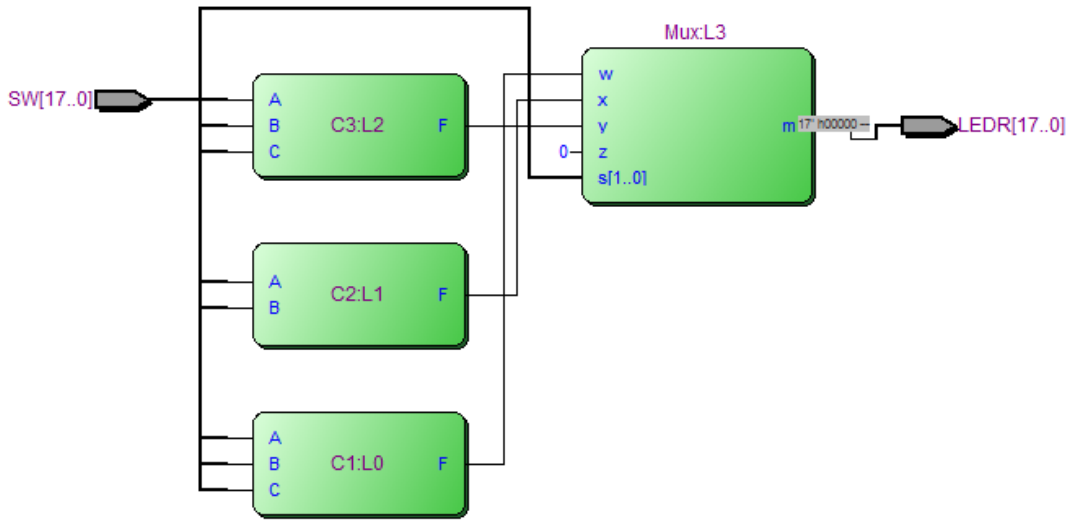


Figure 4.12. RTL viewer block diagram.

Step 5 – Simulation

To start the simulator in Quartus II, select *Tools –> Run Simulation Tool –> RTL Simulation*. Wait for ModelSim to open and perform the following steps:

- Simulate –> Start Simulation;
- In the "Start Simulation" window, look for the "Design" tab;
- In the "work" Library, click on the "+" sign, and select the "top" Entity;
- Click OK to start the simulation;
- Look for the Objects window and click on the "+" sign next to the SW label. This will open all 18 bits of vector SW, as shown in Figure 3.13. Select SW(0), SW(1), SW(2), SW(16) and SW(17) in the Objects window, drag and drop them in the Wave window. Do the same for LEDR(0), F1, F2, F3 and F4.
- Set SW(0), SW(1), SW(2), SW(16) and SW(17) to 0 using the "Force" option, as explained in Chapter 1, "Step 4". Set the simulation run length to 100 ps, and press the *Run* button. Next, change SW(0), SW(1), SW(2), SW(16) and SW(17) to the remaining 31 possible combinations, always running for 100 ps after each input combination.

Figure 4.13 shows the simulation results when the selection bits SW_{17} and SW_{16} are set to "00", and the inputs A, B and C (SW_0, SW_1, SW_2) receive all possible 8 values ("000", "001", …, "111"). Every time SW_{17} and SW_{16} are set to "11", the z input is selected and, as shown in Figure 4.7, for this circuit this mux input is fixed in zero. As shown in the simulation results, at 0.8ns SW_{17} and SW_{16} are set to "11", and the $LEDR_0$ output results in "0".

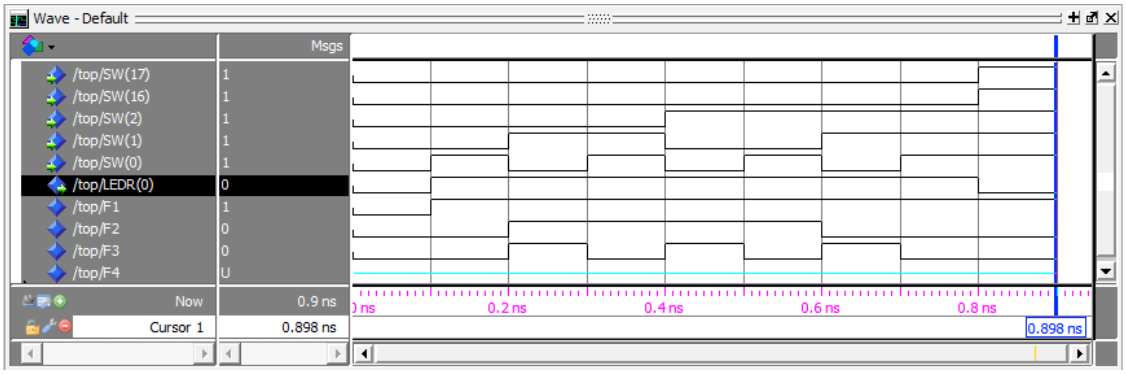


Figure 4.13. Expected simulation results.

Step 6 – Prototyping the circuit in the FPGA board

Power on and connect the FPGA board to the host computer through the USB port (always using the "blaster" USB connector). Press the red button to switch on the board, and run the programming tool: Menu *Tools –> Programmer*.

Follow the instructions provided in Chapter 1, Step 5, observing the "hardware set-up" instructions in Figure 1.32 and in Figure 1.33.

Test the circuit in the FPGA board, using switches SW_0, SW_1, SW_2 as inputs, switches SW_{16} and SW_{17} as selection bits, and observing the results in $LEDR_0$.

Fill in the truth table provided next in Table 4.2. The second column is filled in with the selection bits ("00", "01", "10", and "11"). The third column should be filled in with the results obtained from the manual evaluation of equations F1, F2 and F3. In the third column, when the selection bits are "11", the output is always '0' disregarding the input values. The fourth column can be obtained straight from the simulation results, and it is partially filled in according to the waveforms in Figure 4.13. The fifth column should be filled in with the FPGA execution results. In case of mismatches in any line of the truth table, the circuit should be fully revised in order to find the error.

Table 4.2. Truth table for the MUX based circuit.

Inputs		**Outputs**		
$SW_{2..0}$ C B A	**$SW_{17..16}$ Selection**	***F1 = A or B or C*** ***F2 = B xor C*** ***F3 = (B or C) and (not A)***	**Simulation Step 5 $LEDR_0$**	**FPGA Step 6 $LEDR_0$**
0 0 0	**00**	F1 =	0	
0 0 1	**00**	F1 =	1	
0 1 0	**00**	F1 =	1	
0 1 1	**00**	F1 =	1	
1 0 0	**00**	F1 =	1	
1 0 1	**00**	F1 =	1	
1 1 0	**00**	F1 =	1	
1 1 1	**00**	F1 =	1	
0 0 0	**01**	F2 =		
0 0 1	**01**	F2 =		
0 1 0	**01**	F2 =		
0 1 1	**01**	F2 =		
1 0 0	**01**	F2 =		
1 0 1	**01**	F2 =		
1 1 0	**01**	F2 =		
1 1 1	**01**	F2 =		
0 0 0	**10**	F3 =		
0 0 1	**10**	F3 =		
0 1 0	**10**	F3 =		
0 1 1	**10**	F3 =		
1 0 0	**10**	F3 =		
1 0 1	**10**	F3 =		
1 1 0	**10**	F3 =		
1 1 1	**10**	F3 =		
X X X	**11**	0	0	

Version II – Multiplexer in behavioral VHDL

Follow, basically, the same steps as before in order to implement the behavioral VHDL version of the multiplexer based circuit. There are just a couple of remarks to be considered:

Step 1

A new folder may be created in order to hold the behavioral version of the project. A suggestion of a folder to be created is c:\LabSessions\mux4x1_beh.

Step 2

Copy the version I VHDL files (.vhd) to the new folder, and add them to the project as explained before.

Edit the mux4x1.vhd file, and make the necessary changes in order to have a behavioral implementation of the multiplexer. The VHDL code based on *when* / *else* statements is listed in Figure 4.5(c) and in Figure 4.6.

Steps 3, 4, 5 and 6

There are no need for changes in the top.vhd file, as the new component has exactly the same interface (entity) and functionality as the structural version (version I).

Follow the same instructions for synthesis, simulation and FPGA implementation, as defined for version I.