Synthesizable VHDL Design for FPGAs

Eduardo Augusto Bezerra
Djones Vinicius Lettnin

Universidade Federal de Santa Catarina
Florianópolis, Brazil

August 2013

Contents

Chapter 3. Hierarchical Design

This chapter introduces the concept of hierarchical design in digital systems. A full VHDL project made of several components is developed as a case study, and its components are glued together using the VHDL reserved words *component* and *portmap*. These VHDL statements are used in all remaining projects throughout this book. At the end of the chapter, the students should be able:

- to identify and comprehend the use of components in VHDL;
- to understand the concept of hierarchical design, and its implementation in VHDL;
- to add a component to the proposed case study;
- to simulate and test both, the original and the modified version of the case study;
- to prototype the case study using an FPGA board.

3.1 Hierarchical Design in VHDL

A digital system can be seen as a set of components interconnected in order to perform some required functionality. As applications grow in complexity, so do the number of components in a design. To deal with system complexity, VHDL offers mechanisms for the design organization. The language has appropriate structures allowing complex components to be built from less complex components. The hierarchical design methodology is not a novelty in traditional hardware and software systems. In a similar way, in VHDL a component oriented solution results not only in a more organized design, but also in more opportunities for components reusability. Components already employed in other systems may have been tested and verified, shortening the design cycle.

In order to introduce the VHDL structure and statements used in the hierarchical design methodology, consider the development of a digital system to implement the following equation:

F = (F1 and F2) or F3

Equation 3.1

where *F1*, *F2* and *F3* are outputs in the following equations:

F1 = A or B or C

Equation 3.2

F2 = B xor C

Equation 3.3

F3 = "to be defined"

Equation 3.4

As shown in Figure 3.1, the circuit to be designed in VHDL has four components: *C1* implements Equation 3.2; *C2* implements Equation 3.3; *C3* implements Equation 3.4 (TBD – *to be defined*); and *C4* implements Equation 3.1. Component *C1* has three inputs (*A*, *B*, *C*) and one output (*F1*). Component *C2* has two inputs (*B*, *C*) and one output (*F2*).

Component *C3* has three inputs (*A*, *B*, *C*) and one output (*F3*). Component *C4* has three inputs (*F1*, *F2*, *F3*) and one output (*F*).

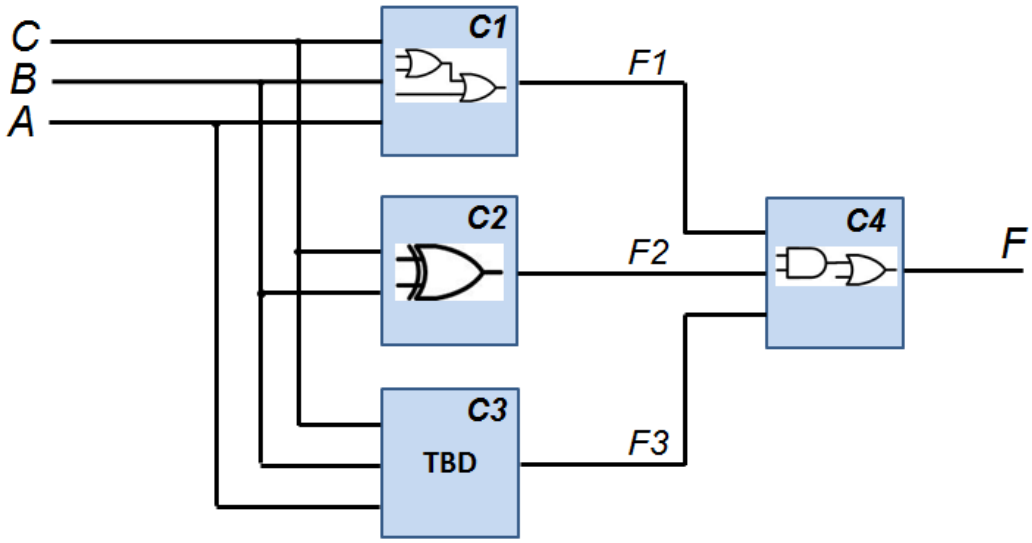

Figure 3.1. Digital system design based on components.

In VHDL, each of these components is implemented through an *Entity/Architecture* construction. Component *C1* is shown in Figure 3.2, component *C2* in Figure 3.3, and component *C4* in Figure 3.4. Component *C3* functionality will be defined next in the "Laboratory Assignment" section.

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity C1 is
port (A: in std_logic;
      B: in std_logic;
      C: in std_logic;
      F: out std_logic
     );
end C1;

architecture c1_estr of C1 is
begin
  F <= A or B or C;
end c1_estr;
```

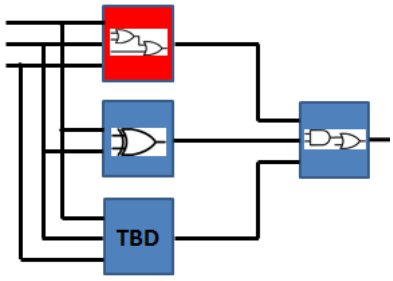

Figure 3.2. VHDL implementation for component C1.

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity C2 is
port (A: in std_logic;
      B: in std_logic;
      F: out std_logic
     );
end C2;

architecture c2_estr of C2 is
begin
  F <= A xor B;
end c2_estr;
```

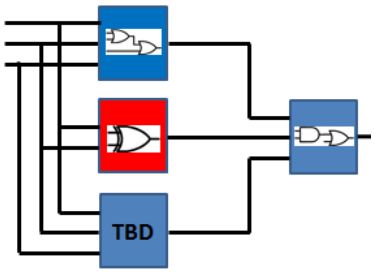

Figure 3.3. VHDL implementation for component C2.

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity C4 is
port (A: in std_logic;
      B: in std_logic;
      C: in std_logic;
      F: out std_logic
     );
end C4;

architecture c4_estr of C4 is
begin
  F <= (A and B) or C;
end c4_estr;
```

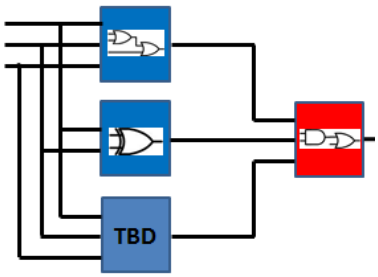

Figure 3.4. VHDL implementation for component C4.

Having all the required components, the next step in a hierarchical design is to glue the components together. As can be observed in Figure 3.1, the four components are connected through wires. The three inputs to *C4* are the wires named *F1*, *F2* and *F3*, which are also the outputs for *C1*, *C2* and *C3*. The inputs for *C1*, *C2* and *C3* are the signals *A*, *B* and *C*, and the circuit output is signal *F*. In VHDL, all these connections are defined using the *port map* statement. In order to use this statement, first it is necessary to list the components to be connected, using the *component* statement. Figure 3.5 has the VHDL code used to generate the circuit shown in Figure 3.1. This VHDL code is usually known

as the “top” component in a design hierarchy. The “top” component is the higher level one, and it has the purpose of instantiating and connecting lower hierarchy level components.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity top is
5     port ( A, B, C : in  std_logic;
6            F       : out std_logic
7     );
8   end top;
9
10  architecture top_map of top is
11     signal F1, F2, F3: std_logic;
12     component C1
13        port (   A : in  std_logic;
14                 B : in  std_logic;
15                 C : in  std_logic;
16                 F : out std_logic
17        );
18     end component;
19     component C2
20        port (   A : in  std_logic;
21                 B : in  std_logic;
22                 F : out std_logic
23        );
24     end component;
25     --
26     -- ADD C3 HERE
27     --
28     component C4
29        port (   A : in  std_logic;
30                 B : in  std_logic;
31                 C : in  std_logic;
32                 F : out std_logic
33        );
34     end component;
35  begin           -- begin architecture
36     L0: C1 port map (A, B, C, F1);
37     L1: C2 port map (B, C, F2);
38     --
39     -- ADD C3 CONNECTIONS HERE
40     --
41     L3: C4 port map (F1, F2, F3, F);
42  end top map;    -- end architecture
```

Figure 3.5. VHDL code for inferring and connecting Figure 3.1 components (top-level component).

In Figure 3.5, components *C1* to *C4* are listed in the *architecture* section, just before the *begin* statement (lines 12 to 34). The VHDL syntax for defining the components is similar to the entity declaration. In fact, a good practice in order to avoid coding mistakes is to copy the desired component's entity section to the top-level file (Figure 3.5), replacing the word "entity" by "component", removing the "is" word, and adding an "end component" at the end of the declaration. Changes in a component header should be performed just in its entity, in order to avoid double changes mistakes. After have made the needed changes in an entity, just copy the whole entity section to the architecture section of the top-level file, and perform the modifications as stated before.

The component declarations in the architecture section of a top-level file inform the synthesis tool which components are to be used in the design. However, so far no component instantiation of connection has been made. In order to "place" the components in the design, and to connect them, the *port map* VHDL statement must be used. In Figure 3.5, line 36, an instance (one copy) of component *C1* is placed in the circuit. On line 37, an instance of *C2* is place in the design, and on line 41 an instance of *C4* is placed in the design. The connections are established according to the arguments in the *port maps* parameters list. On line 36, the first parameter (*A*) tells the synthesis tool to connect the first input signal of component C1 (*A* on line 13) to the *A* signal listed in the top-level entity (line 5). The parameter *B* on line 36 results in the connection between the second input signal of *C1* (*B* on line 14) and the *B* signal listed in the top-level entity (line 5). Parameter *C* on line 36 is the connection from the *C* signal listed in the top-level entity (line 5) to the third input signal of *C1* (*C* on line 15). Finally, the *F1* parameter connects *C1* output to one of *C4* inputs (see line 41).

An important remark is that in the parameters list of *port map* statements, only entity and architecture signals are allowed. This means that all signals listed in the components declarations cannot be used. For instance, the *A*, *B*, and *C* parameters used on line 36 are the signals listed on line 5 of the entity, and not the *A*, *B* and *C* signals that are listed on lines 13, 14 and 15, respectively. The *F1* parameter is defined as an internal signal, in the architecture section.

The synthesis tool makes the relation between the informed parameter in the *port map* and the respective component's signal, according to their order in the argument's list and in the component's declaration. For instance, on line 37, C2 inputs *A* and *B* are connected to the top-level inputs *B* and *C* (listed on line 5). *F2* on line 37 is connected to *F2* on line 41. Therefore, the signals are mapped in the same order as defined in the component declaration.

VHDL has another syntax for signals mapping, which is the "explicit mapping". For example, line 37 can be rewritten as follows:

```
L1: C2 port map (F=>F2, A=>B, B=>C);
```

In this alternative syntax, there is no need to keep the parameters order, as the => operator forces the signals mapping. Thus, *A*, *B* and *F* signals of *C2* are explicitly connected to *B*, *C* and *F2*, where *B* and *C* are top-level entity signals, and *F2* is an internal signal.

Another important remark is that each entity/architecture construction needs its own library declaration. Also, it is a good coding practice to keep each component in a

separate file. The top-level entity/architecture (see Figure 3.5) should also be kept in another file, and the synthesis tool can find all components of a design just looking for the "component" statements in this file.

Figure 3.6 shows the circuit generated from the VHDL code in Figure 3.5. As stated on line 36, *C1* has A, B and C as inputs, and *F1* as its output. On line 37, *B* and *C* are defined as the inputs for *C2*, and *F2* is its output. Component *C4* is instantiated on line 41, having *F1*, *F2* and *F3* as inputs, and *F* as output. All 4 components are in the box named "top" in Figure 3.6, and this box represents the VHDL code in Figure 3.5 (see the entity's name).

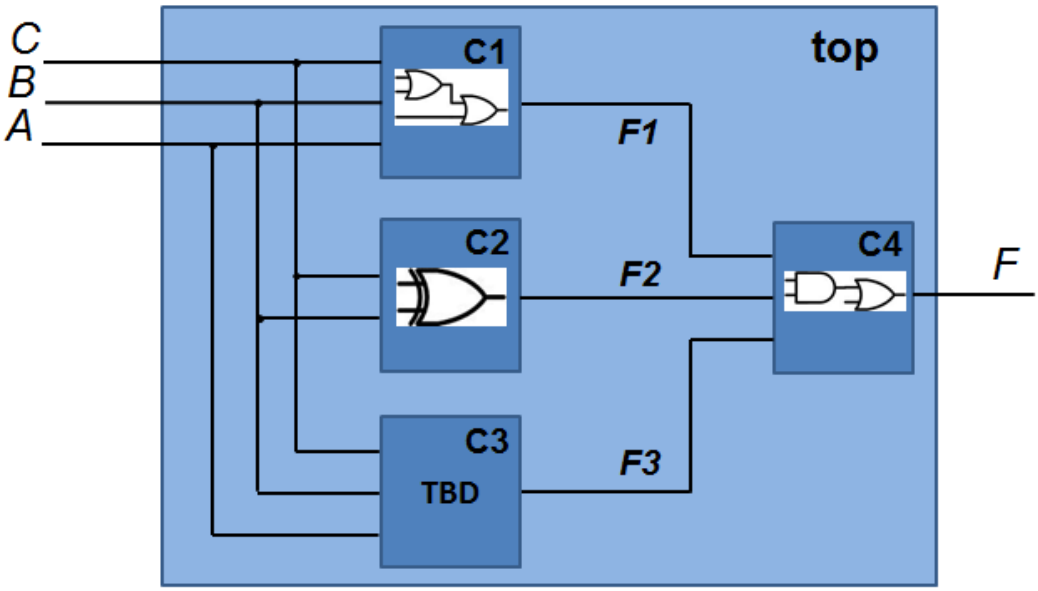

Figure 3.6. Final circuit generated from VHDL code shown in Figure 3.5.

3.2 Laboratory Assignment

The laboratory objectives are:

- to practice the basic concepts of hierarchical design in VHDL;
- to build a basic design to be used as a skeleton in the remaining lab sessions in this book;
- to carry on the training and understanding of a complete FPGA design flow.

Laboratory Session

This lab has as its main task the design of the circuit shown in Figure 3.6, including all the aforementioned components. The whole circuit is built from five VHDL files, each of them comprising an entity and an architecture section. The four components *C1*, *C2*, *C3* and *C4*, are saved in files *c1.vhd*, *c2.vhd*, *c3.vhd*, and *c4.vhd*, respectively. Component *C3* has to be implemented according to the specifications provided next. The top-

level component (see Figure 3.5), performs all the instantiations and connections and it is saved in file *top.vhd*.

The tasks to be completed in this laboratory session, using Altera's Quartus II EDA tool, are as follows:

- Create a new project with all the components shown in Figure 3.6 – **Steps 1, 2**;
- Create the missing component (*C3*) using the VHDL design entry editor – **Step 3**;
- Modify the top-level component provided in Figure 3.5, in order to add *C3* to the circuit – **Step 4**;
- Modify the top-level component provided in Figure 3.5, in order to adapt its inputs and outputs to the signal labels used in the DE2 (prototyping board) interface – **Step 5**;
- Perform the synthesis – **Step 6**;
- Perform the simulation and fix errors, if any – **Step 7**;
- Prototype and test the circuit in the FPGA board – **Step 8**.

Step 1 – Creating a new project

This step is essentially the same as "Step 1" described in Chapter 1. The major difference is in the project's name. In this new design, instead of "halfadder" use "top" as the project's name (page 1 of 5 in the New Project Wizard – see Figure 1.13), as this is the name of the entity in Figure 3.5 (top-level entity).

Step 2 – Adding VHDL files to the project

Chose *File –> New* and *VHDL File*, as shown in Figure 3.7. This will open an empty text editor.

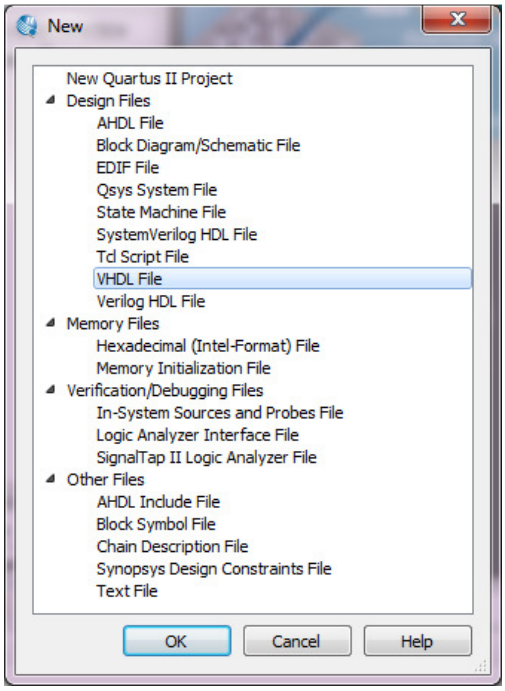

Figure 3.7. New VHDL file.

Type in the text editor the VHDL code for *C1* as shown in Figure 3.2, and save it: *File –> Save* (or just Ctrl-S). The default name to save the file is top.vhd, as it is the name provided as the "project name" in Step 1. Before saving it, change the name to c1.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. Again, at this point it is important to be sure that the entity's name is also "C1" otherwise, the synthesis tool will not be able to find the entity to be synthesized.

Repeat the same procedure to create c2.vhd, c4.vhd, and top.vhd:

1. *File –> New –> VHDL File*;
2. Type de VHDL code for each component and the top-level – use Figure 3.3 for c2.vhd, Figure 3.4 for c4.vhd, and Figure 3.5 for top.vhd;
3. Use *File –> Save* for saving each file, always changing the name according to what is being saved;
4. Be sure to select the "Add file to current project" checking box;

Alternatively, if you already have c1.vhd, c2.vhd, c3.vhd, c4.vhd and top.vhd, just select *Project –> Add/Remove Files in Project*, as shown in Figure 3.8. This option will open a window allowing the import of VHDL files into the new project.

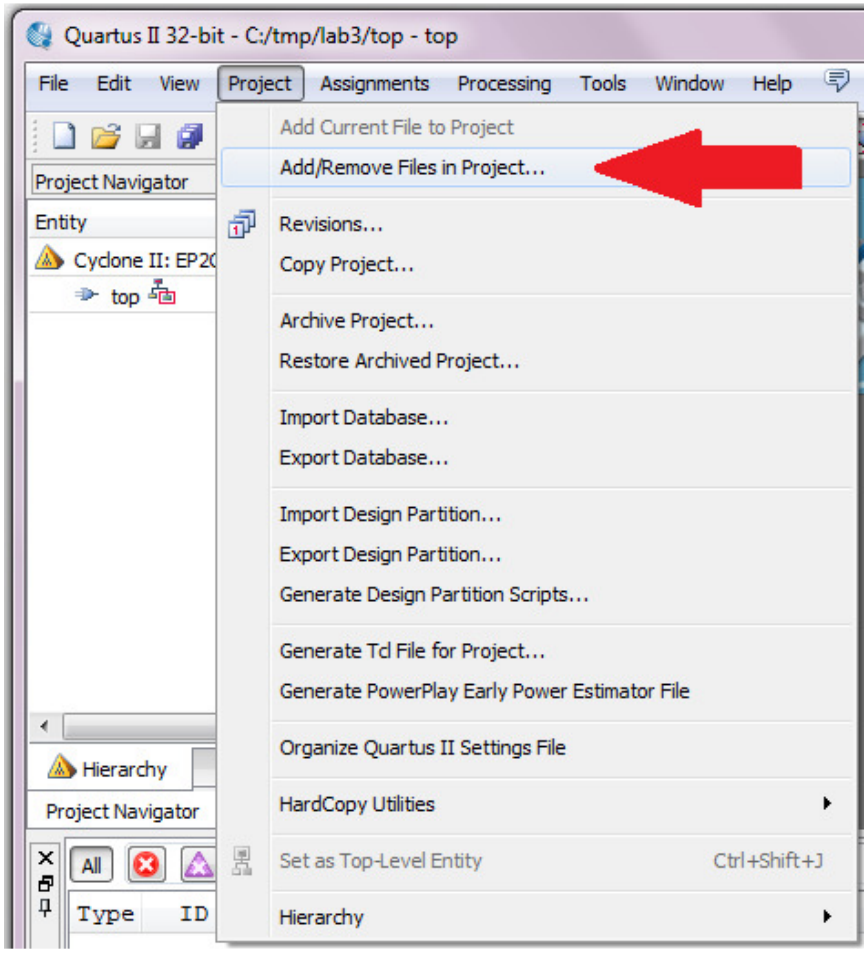

Figure 3.8. Adding existing files to the project.

Step 3 – Creating and adding *C3* to the project

Using C1, C2 or C4 as a model, write in VHDL the circuit presented in Figure 3.9. Follow "Step 2" instructions to create and add this component as c3.vhd to the project.

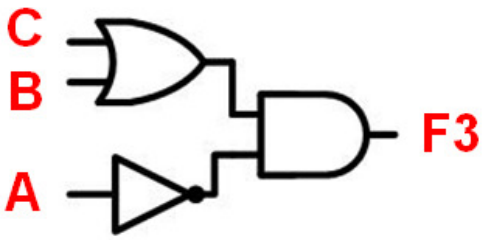

Figure 3.9. C3 schematic diagram.

Step 4 – Editing top.vhd to add *C3* to the project

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. Add *C3* component interface (entity) to the designated location in Figure 3.5 (see line 26), and make the necessary modifications in order to have the wright syntax for components definition in VHDL. Use components *C1*, *C2* and *C4* of Figure 3.5 as examples.

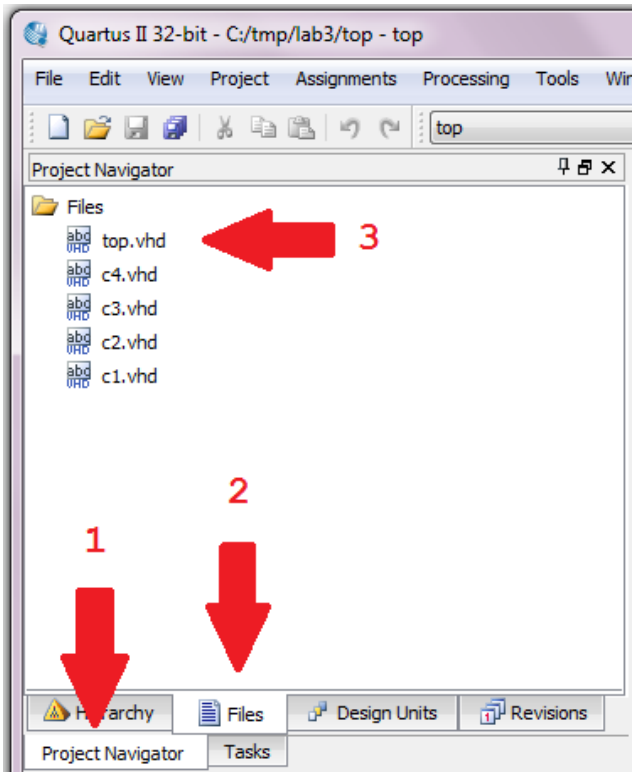

Figure 3.10. Making changes to top.vhd.

Create an instance of *C3* using the port map statement on line 39 of Figure 3.5. Use the port maps of components *C1*, *C2* and *C4* as examples, but be careful to make the correct connections according to Figure 3.6 and Figure 3.9. Notice that in Figure 3.9, *B* and *C* are the inputs for the OR gate, and *A* is the input for the NOT gate. *F3* is the component output. All these connections should be performed by the port map statement.

Step 5 – Adapting the design to work in the DE2 FPGA board

At this point, the circuit is ready for synthesis (see Chapter 2, Step 3) and simulation (see Chapter 2, Step 4), but it will not work on DE2 FPGA board. As explained in Chapter 2, and introduced in Figure 2.8, a circuit to be prototyped in DE2 board should employ specific signal names in its entity.

Figure 3.11 shows the available switches and LEDs on DE2 board. In the design shown in Figure 3.12, *A*, *B* and *C* inputs will be provided by DE2 switches 0, 1 and 2, respectively. The *F* output will be presented in red LED 0.

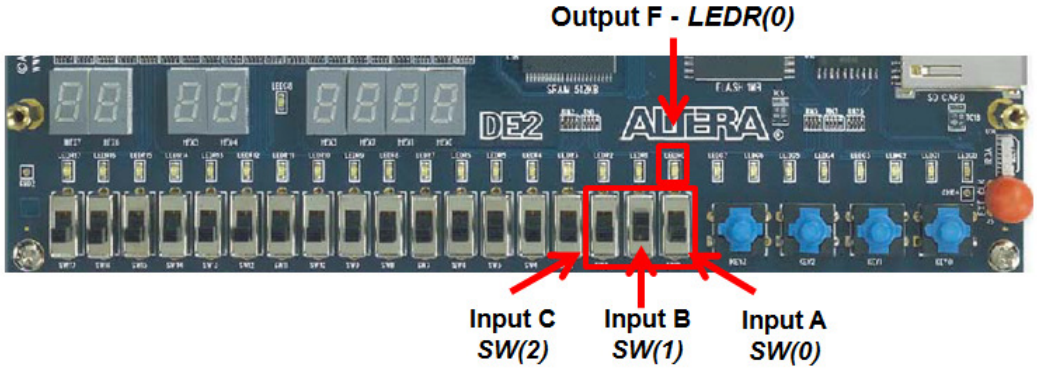

Figure 3.11. DE2 input and output resources.

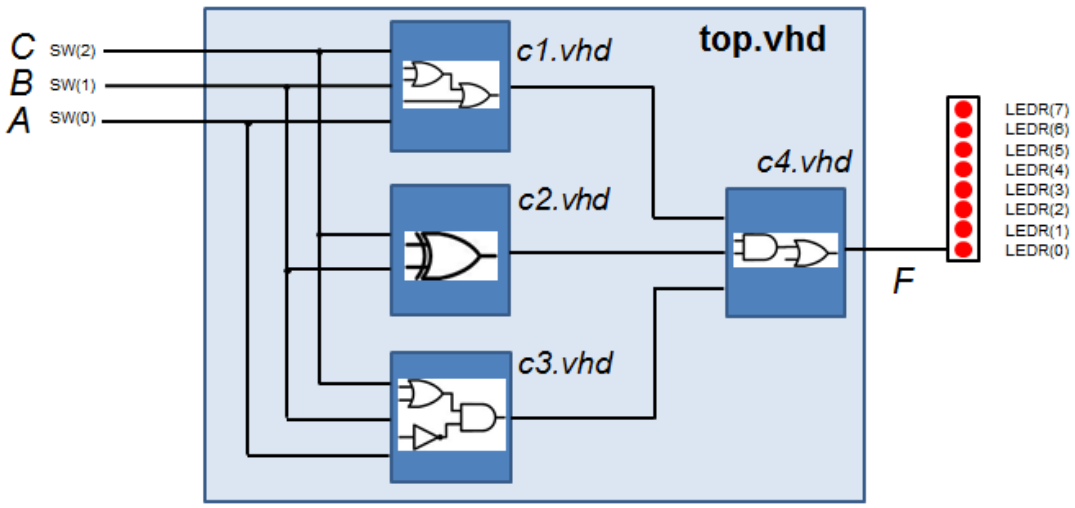

Figure 3.12. Final circuit with DE2 board connections.

In order to adapt inputs and outputs of top.vhd component to the DE2 board, replace its entity signals in Figure 3.5, lines 5 and 6 by:

```
4:    entity top is
5:      port ( SW  : in  std_logic_vector(17 downto 0);
6:             LEDR: out std_logic_vector(17 downto 0)
7:      );
8:    end top;
```

SW is an 18 bits vector, where each bit represents a DE2 switch. In a similar way, LEDR is also an 18 bits vector but representing the 18 red LEDs. In this new entity there is no longer *A*, *B*, *C* and *F* signals. So, all occurrences of *A*, *B* and *C* in Figure 3.5 should be replaced by SW(0), SW(1) and SW(2), respectively. All occurrences of *F* should be replaced by LEDR(0). The port map statements for *C1* and *C4* for instance, should look like:

L0: C1 port map (SW(0), SW(1), SW(2), F1);
L3: C4 port map (F1, F2, F3, LEDR(0));

Only the top-level entity signals *A*, *B*, and *C*, and the internal signal *F* should be replaced by SW and LEDR, respectively. The remaining *A*, *B*, *C* and *F* signals belonging to the components should not be changed.

Step 6 – 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.

Step 7 – Simulation

If the simulation tool has not been configured already, follow the instructions in Chapter 1, "Step 4". 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), and SW(2) in the Objects window, drag and drop them in the Wave window. Do the same for LEDR(0), F1, F2, and F3. At the end, the Wave window should look like Figure 3.14.

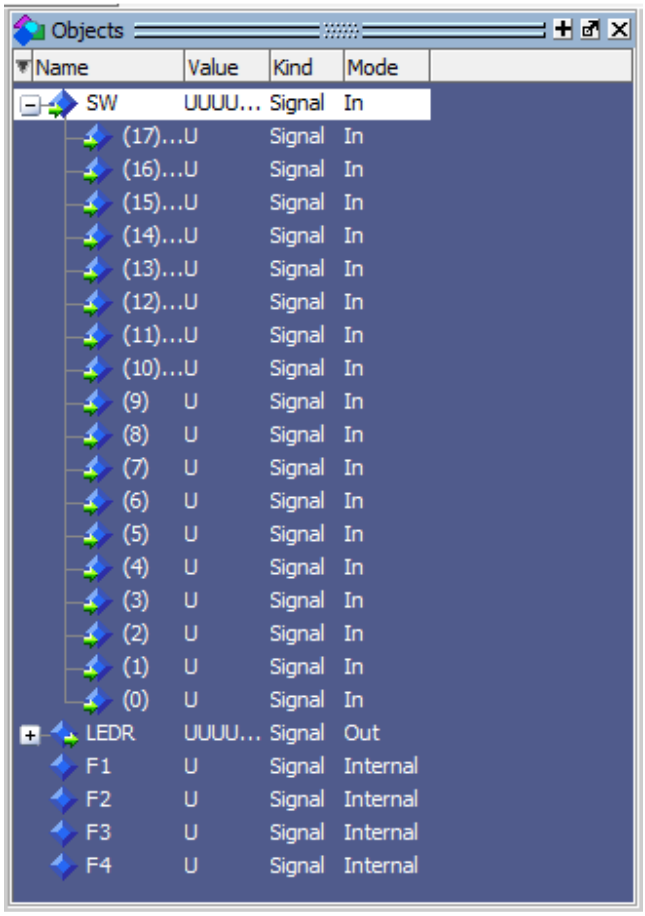

Figure 3.13. Simulation signals selection.

- Set SW(0), SW(1) and SW(2) 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) and SW(2) to the remaining seven possible combinations, always running for 100 ps after each input combination.
- The expected simulation results are shown in Figure 3.15.

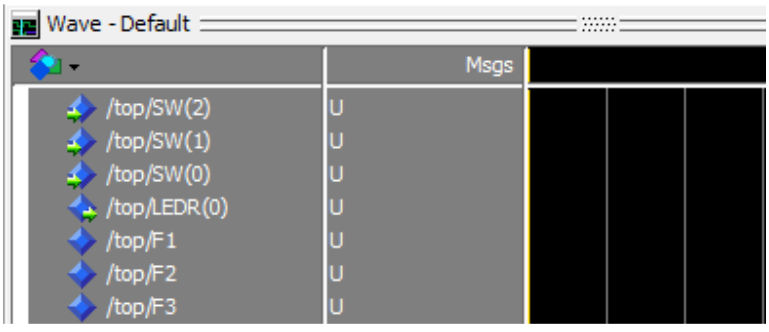

Figure 3.14. Wave window.

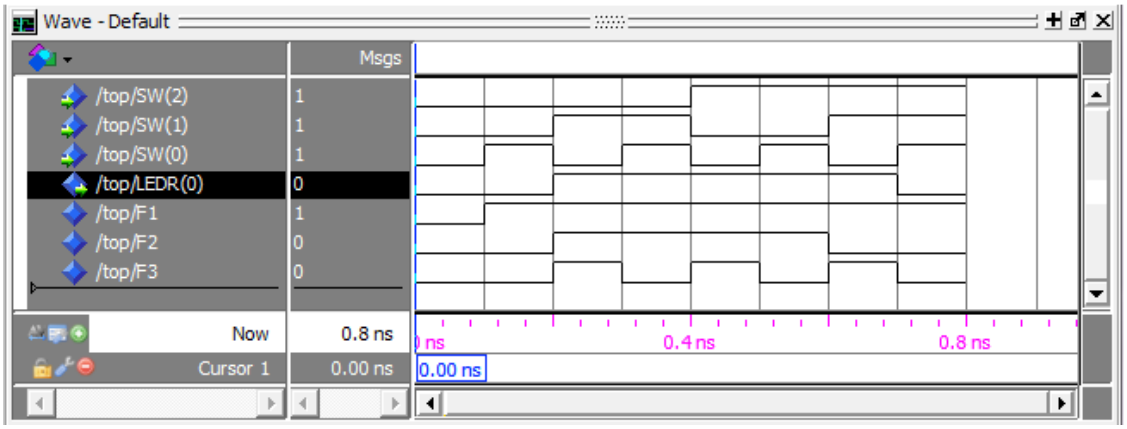

Figure 3.15. Expected simulation results.

Step 8 – 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) and SW(2) as inputs, and observing the results in LEDR(0).

Fill in the truth table provided next in Table 3.1. The second column should be filled in with the results obtained from the evaluation of Equation 3.1. The third column can be obtained straight from the simulation results shown in Figure 3.15. The fourth 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 3.1. Truth table for the case study.

Input SW(2..0) C B A	Output *F = (F1 and F2) or F3 where: F1 = A or B or C F2 = B xor C F3 = (B or C) and (not A)*	Output Obtained in Step 7 Simulation LEDR(0)	Output Obtained in Step 8 FPGA LEDR(0)
0 0 0			
0 0 1			
0 1 0			
0 1 1			
1 0 0			
1 0 1			
1 1 0			
1 1 1			

The Quartus II tool "RTL viewer" can be used to check if a circuit has been generated according to its requirements. Figure 3.16 shows the menu used to start the tool, and Figure 3.17 shows the circuit's block diagram generated by Quartus II. In this block diagram, the developer can travel through the design hierarchy by double-clicking on the components.

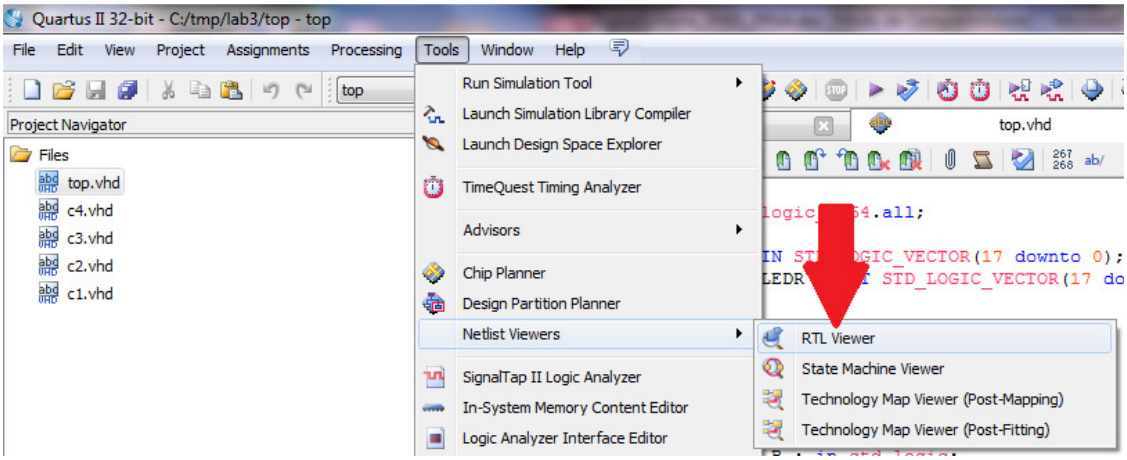

Figure 3.16. Quartus II RTL viewer tool.

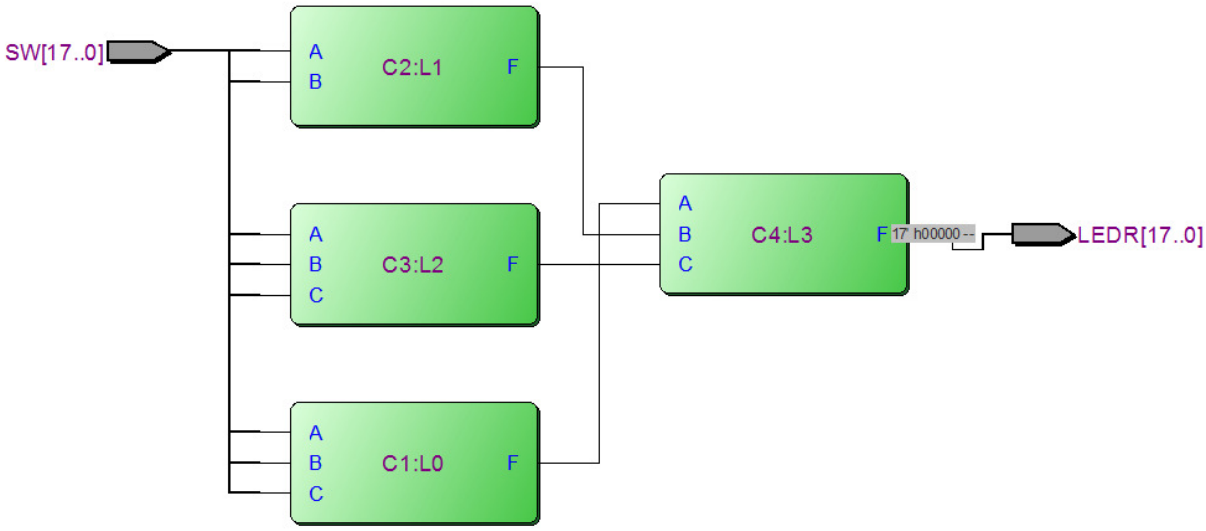

Figure 3.17. Final circuit in RTL viewer.