Laboratory 4

XILINX-ISE Methodology. Schematic description. Logic gates.

4.1 Objectives

The objectives of this laboratory are:

- Getting familiar with the Xilinx WebPack ISE environment;
- Using Xilinx Schematic Editor to create logic circuit drawings.

4.2 Usage of ISE WebPack Schematic Editor

XILINX ISE (Integrated Software Environment) is a digital system design environment for Xilinx's programmable digital circuits. Xilinx is offering a wide range of applications for design using FPGA or CPLD circuits. The design of the a digital system can be done using the Schematic editor or one of the two supported hardware description languages (VHDL or Verilog).

For example, we consider the design of a combinatorial system which implements the following logic function:

$$F = AB + CD \tag{4.1}$$

4.2.1 Starting ISE

To launch *ISE WebPack* double click on the icon associated with the program or follow the steps:

Start —> All Programs —> Xilinx ISE —> ISE —> Project Navigator.

The main window, *Project Navigator*, offers an interface which organizes all files and applications in such way that the user has a better overview of the design process. The main window is divided into 4 panes:

- **Source panel** presents a list of all files associated with the current project;
- **Process pane** holds a list of all processes and operations which are available for the selected file;

- **Transcript pane** shows the status of the processes, as well as warning and error messages resulted by running different processes;

- **Editor pane** is used to view the content of a file.

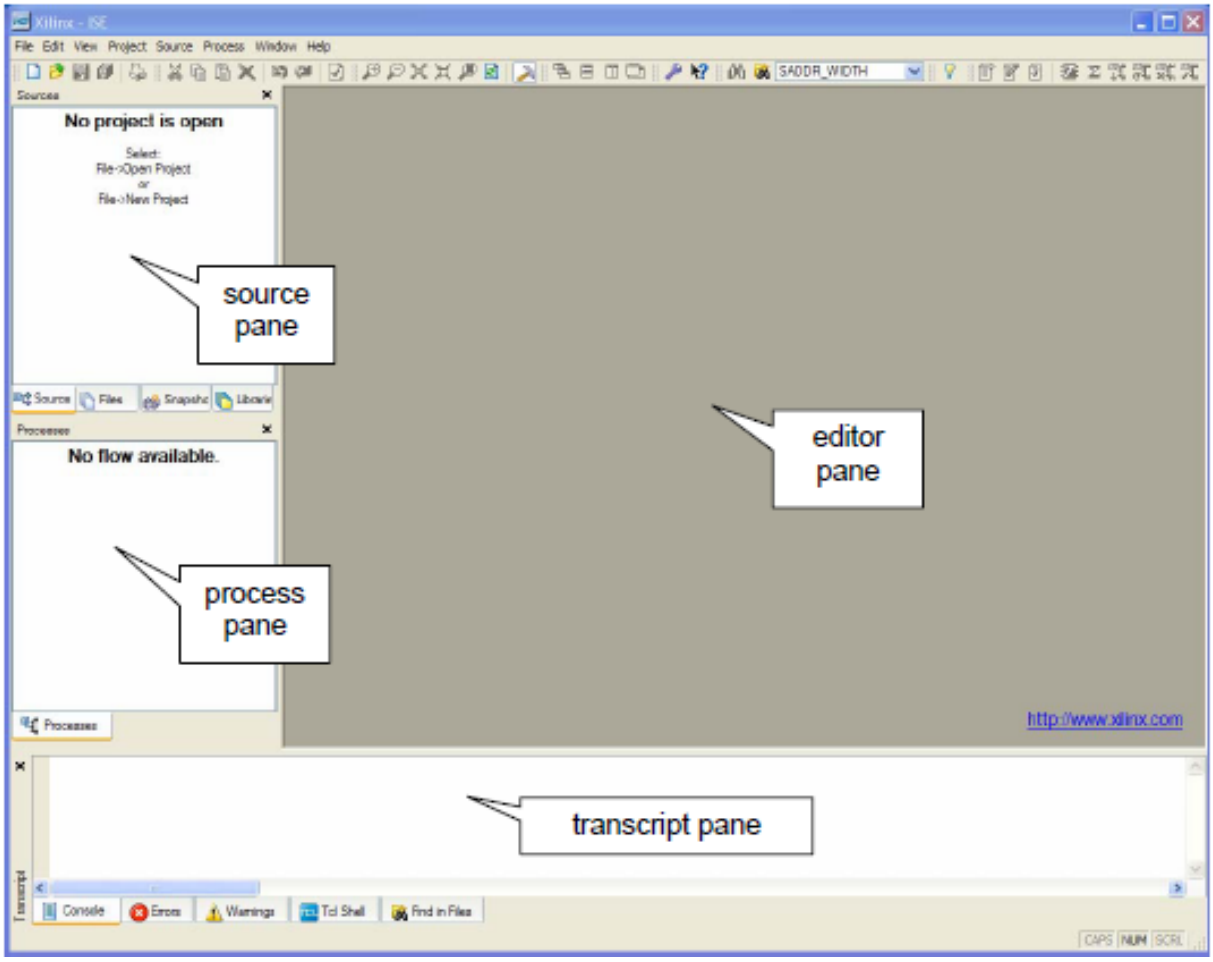

Figure 4.1 The main window of XILINX ISE Webpack.

4.2.2 Creating a new project

To create a new project access:

File —> New Project

Complete the fields for project name, location and for *Top-level source* select **Schematics**. Click on *Next*. Select (and remember) the directory the project is created in. Name the project as **functionF**. The project consists of a set of files which going to be created in the project folder.

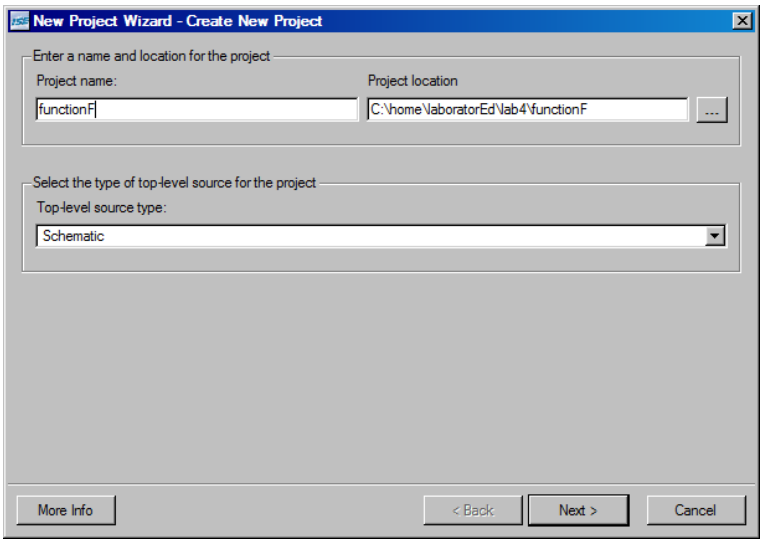

Figure 4.2 *New Project Wizard Create New Project* window.

In the device description window, shown in figure 4.3, complete the fields which describes the Xilinx integrated circuit used in your project. Click *Next*.

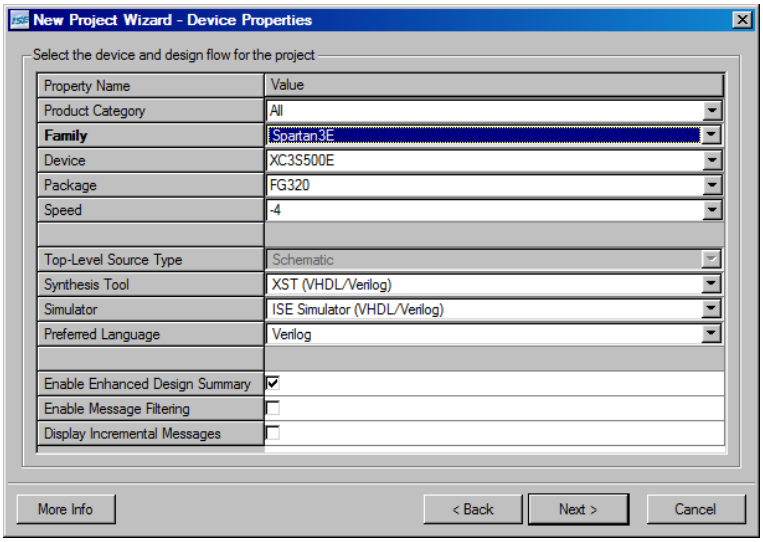

Figure 4.3 *New Project Wizard Device Properties* window.

The development board used in the laboratory contains an FGPA circuit with the following features:

Device Family: Spartan3E
Device: XC3S500E
Package: FG320
Speed Grade: -4

The next 2 windows which are going to pop-up, *New Project Wizard Create New Source* and *New Project Wizard Add Existing Sources* are shown in figures 4.4 respectively 4.5. At this stage the user can create new source files or existing ones can be added to the project. Both creating new

source files or adding existing ones can be done even after the creation of the project (later stages) by selecting:

Project —> New Source...

Project —> Add Source...

For now, bypass these 2 windows by clicking on *Next* two times.

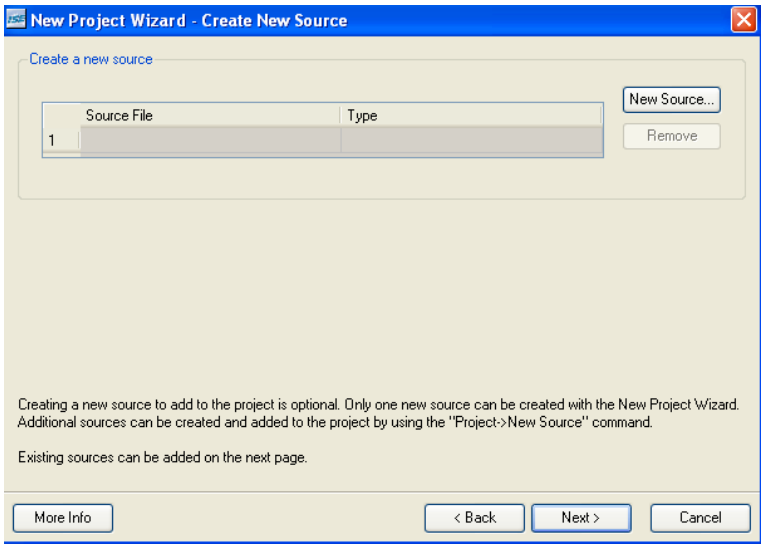

Figure 4.4 *New Project Wizard Create New Source* window.

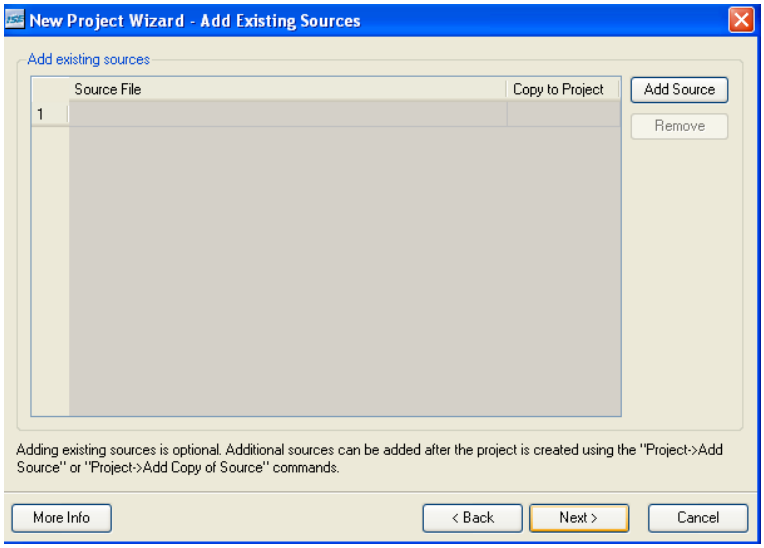

Figure 4.5 *New Project Wizard Add Existing Sources* window.

In the last window of the set-up process, *New Project Wizard Project Summary*, you will find a summary of the newly created project including details of the circuit type used. Click on *Finish*.

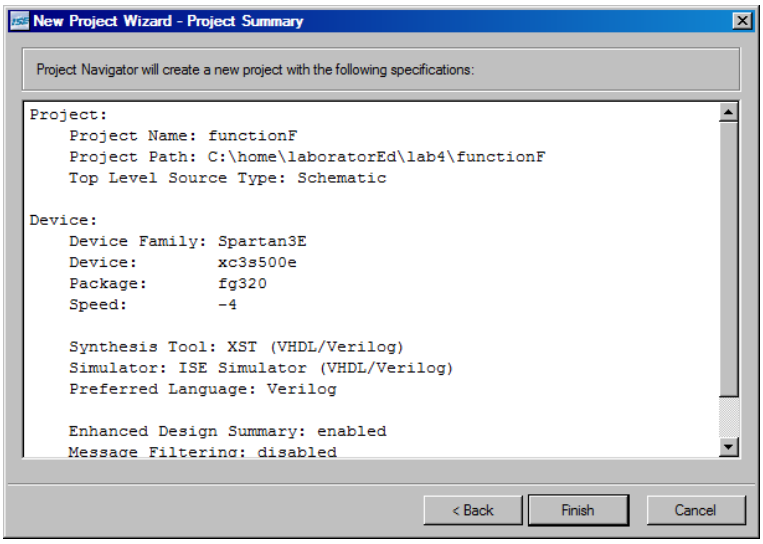

Figure 4.6 *New Project Wizard Project Summary* window.

4.2.3 Creating a new schematic file (.sch)

To create a schematic source file, select *Sources* model **xc3s500e-4fg320** from the source pane, then and click *Processes* in Processes pane and double click on *Create New Source*.

An another way to do the above mentioned operation is to access *Project* —> *New Source...*

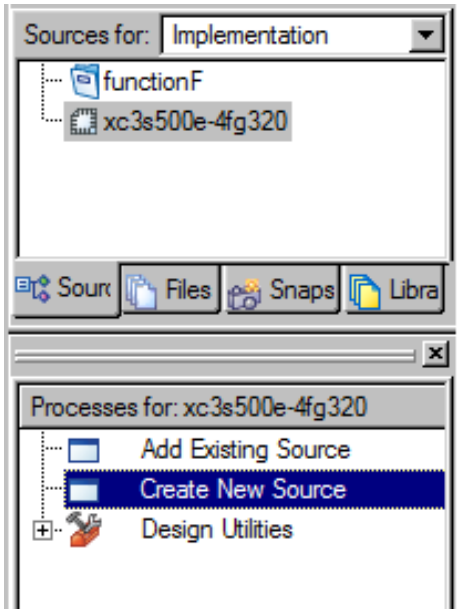

Figure 4.7 *Sources* pane and *Processes* pane.

In *New Source Wizard Select Source Type* window select **Schematic** and complete the fields for file name and location. Click on *Next* and then *Finish*. Name the file as **functionF**. The file will be saved with the **.sch** extension. Make sure you have selected the **Add to project** option.

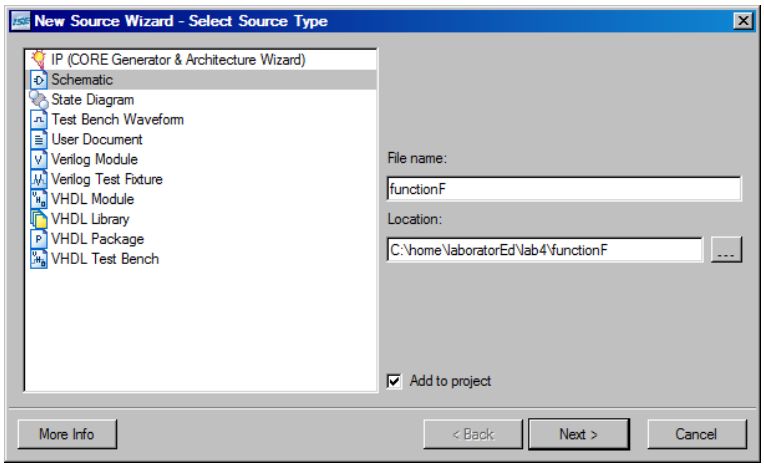

Figure 4.8 *New Source Wizard Select Source Type* window.

4.2.4 Adding components

The schematic drawing with logic gates associated to our function is shown in the figure 4.9.

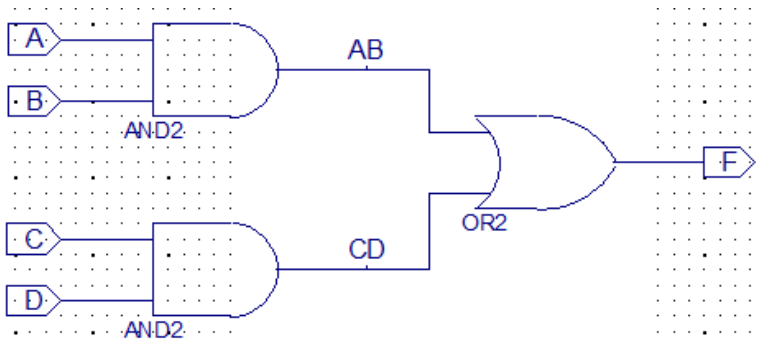

Figure 4.9 Schematic drawing with logic gates associated to the *F* function.

On the upper left corner of *Sources* window you can find a list of the available symbols and a list of symbol categories. To easily find a component (in this case logic gates) select *Categories* **Logic**. In *Symbols* list you will find logic gates's symbols only. If you wish to find a symbol by it's name, just try to type the first letters of the symbol's name in the *Symbol Name Filter* field, thus narrowing down the number of results. In *Symbols* list you will find all symbols whose name starts with the letters you typed in to the *Symbol Name Filter* filter field.

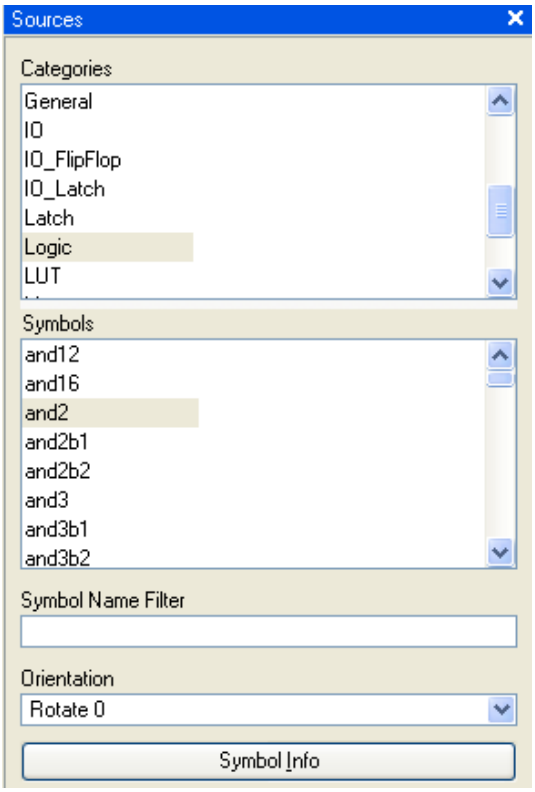

Figure 4.10 Symbols window.

Select the symbol you need. To place it in your schematic, click on the location you wish to it to be placed on. Place all the needed symbols this way.

4.2.5 Editing connections

Once the components are placed, these needs to be connected between each other with simple wires or buses.

In order to make a connection click:

Add –> Wire (CTRL+W)

or

click on *Add Wire* icon  .

To make a connection between the pins of two components click on the pin of the first component then click on the pin of the second component.

A net can be given a name by selecting:

Add –> Net Name (CTRL+D)

or

click on *Add Net Name* icon  .

Create the connections between the components as shown in figure 4.9. Give the net the names as you can see in the above mentioned figure.

4.2.6 Adding input/output ports

To add input/output ports click on:

Add –> I/O Marker (CTRL+G)

or

use *Add I/O Marker* icon  .

Place I/O ports as shown in figure 4.9. Rename the the implicitly named ports with the recommended designations shown in figure 4.9. Right click on the port and then click *Rename port*.

Notice that I/O port names are case sensitive.

4.2.7 Verifying the schematic

In order to verify the correctness of the schematic (from electric point of view) click on

Tools –> Check Schematic

or

click on *Check Schematic*  .

If there are any errors regarding the wire connections, ports or components, they will be reported through the error messages displayed in the Console section.

If there are no errors, the displays message is:

```
Start DRC ...
No error or warning is detected
```

4.2.8 Introducing graphical and text elements

Additionally to the functional elements of the schematic, graphical and text elements can be introduced: block for the title of the schematic, circles, arches, lines, rectangles or text.

To add a title block the procedure is similar to the one used to add components. In the *Sources* window, at *Categories* select **General**. At *Symbols* select **title** component. After the block is placed, you can edit it's parameters by double clicking on it.

In the *Object Properties* window, figure 4.11, the user can modify the name, title and add additional fields (e.g. additional info about the schematic) or modify other parameters.

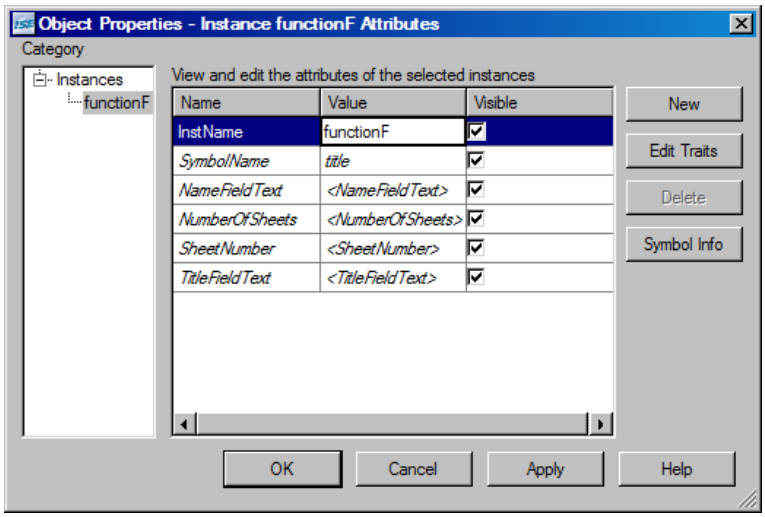

Figure 4.11 *Object Properties* window.

Other graphical elements:

Rectangles: click *Add –> Rectangle* or click on the *Add Rectangle* button;
Circles: click *Add –> Circle* or click on the*Add Circle* button;
Lines: click *Add –> Line (CTRL+L)* or click on the *Add Line* button;
Arches: click *Add –> Arc* or click on the *Add Arc* button.

4.2.9 Saving the schematic

To save the schematic click on:

File –> Save (CTRL+S)

or

click *Save*.

Save your schematic in **functionF.sch** file.

4.2.10 Creating and using custom symbol

In order to facilitate your future work you can create your own symbols or you can modify the existing ones. The newly created symbols will be stored in the local symbol library. If you intend to reuse a design in the future as part of an another design is a good idea to create a symbol for it. If you do so, the whole design can be used by simply placing it's symbol in the new design which contains it.

To create a new symbol, click on *Tools –> Symbol Wizard*. In the first dialogue window you can select the shape for the symbol, as well as the naming convention for the I/O pins. For ease, select *Pin Name Source = Using Schematic*. The next window, *Symbol Wizard Pin Page* as shown in figure 4.12, the I/O pins will appear with the name used in the source schematic. If you chose *Specify manually* you have to manually introduce name for each I/O pin.

After finishing editing the symbol, click *Next –> Next –> Finish*.

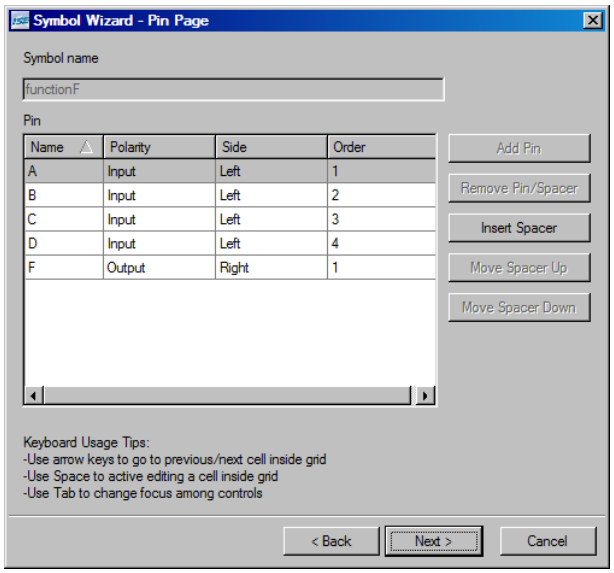

Figure 4.12 *Symbol Wizard Pin Page* window.

For the end, the symbol have to be saved as a **.sym** file. The name of the symbol should also be the name of the file. In order to be able to use the newly created symbol in a new project, you have to copy the **.sym** file and the according **.sch** file in the root folder of the destination project.

4.3 Testing the design on the development board

In order to be able to test the design on the development board we need to assign every I/O with of the schematic to a physical pin of the FPGA. For example, we connect the inputs to the four available switches and the output to an LED. This way we can trigger the circuit and observe the output. The mapping in done based on a special file, having the **.ucf** extension (*User Constraints File*). This file contains the association between design I/Os and physical pins of the integrated circuit (FPGA).

The UCF syntax is the following:

```
NET "portName"  LOC = "pinLocation";   # comments
```

The **functionF.ucf** constraints file looks like the following (for our sample project):

```
# LED
NET "F"  LOC = "F12";  # LED0

# SWITCH
NET "A"  LOC = "L13";  # SW0
NET "B"  LOC = "L14";  # SW1
NET "C"  LOC = "H18";  # SW2
NET "D"  LOC = "N17";  # SW3
```

Using a simple text editor (**Notepad**), create and edit the content of **functionF.ucf**. After you have created the file, you have to add it to the design, by clicking *Add Source* and selecting the **functionF.ucf**.

View the content of the file in the *Sources* sub-pane. The window is similar to the one shown in figure 4.13.

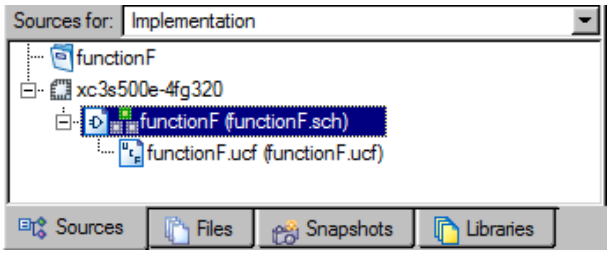

Figure 4.13 Source files of the **functionF** project.

As all the necessary files have been added we can proceed to the implementation of the design. Select the **functionF.sch** file by clicking on it. In the *Processes for: functionF* window (figure 4.14) you can see the implementation stages for the design (FPGA specific). These implementation stages are:

- **Synthesize - XST** - conversion of the schematic or textual description of the design into a list of XILINX technology specific primitives. The software tool will generate status reports, a graphical representation of the design, as well as a post-synthesis simulation model.
- **Implement Design** - this step is divided in 3 sub-steps.
  - **Translate** - conversion of the project from different description types (schematics, graphs, description languages, etc.) to an unitary representation mode.
  - **Map** - mapping of the logical operators to the existing physical resources of the FPGA.
  - **Place & Route** - placement of the resources on the available locations and routing the interconnecting wires to implement the desired function.
- **Generate Programming File** - generation of the programming file which contains the information necessary to configure all the existing resources of the FPGA. The file has the **.bit** extension.
- **Configure Target Device** - launching a software application which transfers the **.bit** file from the computer to the FPGA circuit. As a workaround, the content of **.bit** file can be converted to ROM memory image, from which the FPGA can be configured as well.

These design stages can be performed one-by-one or all at once. After every step, numerous reports will be exported, so the designer can monitor the progress and it's correctness.

For this sample project ignore the intermediate stages and click on the **Generate programming file**. All the necessary steps will run successively and at the end the tool will generate the programming file (**.bit**). The generated programming file, **functionF.bit**, will be loaded to the *Spartan3E* development board and will configure the FPGA to perform the desired function.

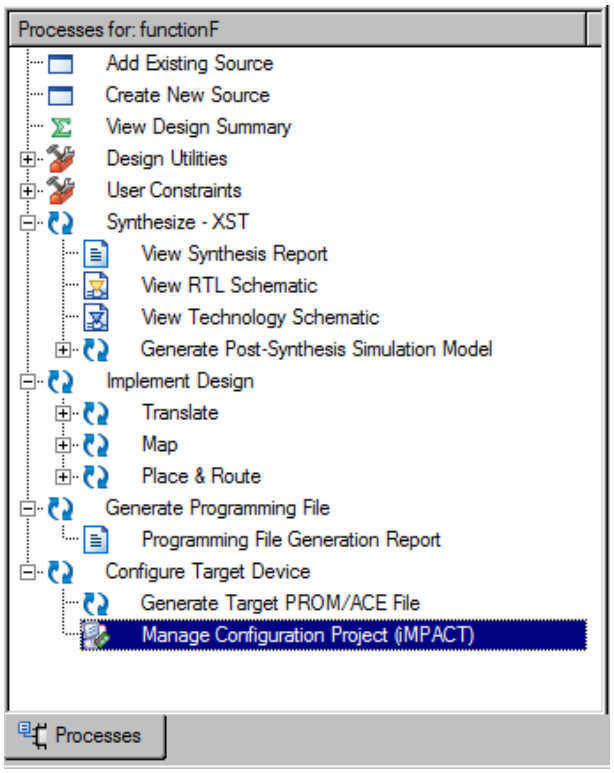

Figure 4.14 FPGA specific design implementation stages.

After processing the input files, the tool will generate a file containing the Verilog description of the design, which will be saved as a **.vf**. For our sample project, the content of the file will contain the instances of the logic gates implementing the desired function:

```
‘timescale 1ns / 1ps

module functionF(A,
                 B,
                 C,
                 D,
                 F);

    input A;
    input B;
    input C;
    input D;
   output F;

   wire AB;
   wire CD;

   AND2 XLXI_1 (.I0(B),
                .I1(A),
                .O(AB));
   AND2 XLXI_2 (.I0(D),
                .I1(C),
```

```
                .O(CD));
   OR2 XLXI_3 (.I0(CD),
               .I1(AB),
               .O(F));
endmodule
```

4.3.1 Loading the (.bit) configuration file to the board

To load the configuration file to the development board, expand the *Processes* **Configure Target Device** pane and double click on **Manage Configuration Project (iMPACT)**.

The FPGA is programmed via the JTAG interface. The development board contains 3 JTAG-enabled devices, connected in a chain.:

- **xc3s500e** the FPGA (used for the sample project implementation).
- **xcf04s** EPROM (contains a default program for the FPGA, mainly used to perform the initial tests for the development board).
- **xc2c64a** CPLD (is a hardware module used to perform the programming of the FPGA circuit with the content of the EEPROM, at power-up).

After automatically determining the cable type connected between the computer and the Spartan-3E board, the software tool will show a graphical representation of the chain consisting of the above mentioned 3 circuits, as shown in figure 4.15.

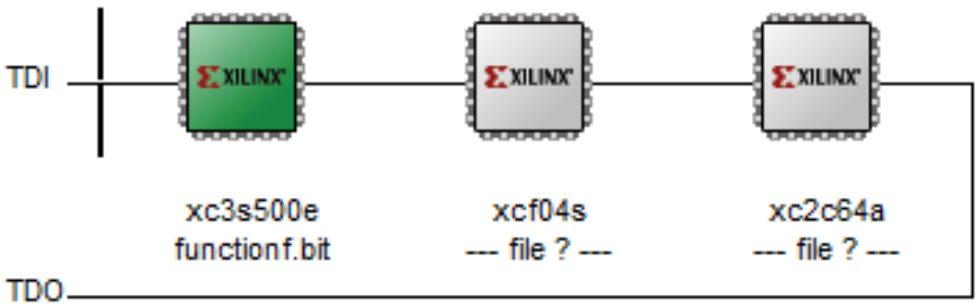

Figure 4.15 Graphical representation of the JTAG chain with the 3 components, FPGA, CPLD and EEPROM.

Every of the 3 JTAG components can be separately programmed through the JTAG interface and has it's own programming file. On the first dialogue window click on *Finish*. In the following window you will select programming file that is going to be loaded on the FPGA (the first item in the chain) and click **Open**. For the next two items of the chain just click on *Bypass*.

To finish the process, right click on the **xc3s500e** FPGA and click **Program**.

Using the switches trigger the inputs of the circuit for all the possible combinations and fill in the 4.1 truth table. Verify the results by comparing them to the ones obtained with analytically solving the function.

Table 4.1

Truth table of the F function.

A	B	C	D	F
0	0	0	0	
0	0	0	1	
0	0	1	0	
0	0	1	1	
0	1	0	0	
0	1	0	1	
0	1	1	0	
0	1	1	1	
1	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

4.4 Knowledge testing

Derive the schematic representation and implement it on the FPGA for the following Boolean functions.

$$\begin{aligned} G &= A + B\overline{C} + D \\ H &= ((A \oplus B)C)D \end{aligned} \tag{4.2}$$

Write a **.ucf** constraint file in which you map the inputs of the functions to the available switches and the output to a LED.

Fill in the 4.2 truth table for both functions and verify the result by comparing them to the results obtained by analytical calculus.

Table 4.2

The truth table for G and H functions.

A	B	C	D	G	H
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		