Quartus II Introduction Using VHDL Designs

For Quartus II 14.0

1 Introduction

This tutorial presents an introduction to the Quartus® II CAD system. It gives a general overview of a typical CAD flow for designing circuits that are implemented by using FPGA devices, and shows how this flow is realized in the Quartus II software. The design process is illustrated by giving step-by-step instructions for using the Quartus II software to implement a very simple circuit in an Altera FPGA device.

The Quartus II system includes full support for all of the popular methods of entering a description of the desired circuit into a CAD system. This tutorial makes use of the VHDL design entry method, in which the user specifies the desired circuit in the VHDL hardware description language. Two other versions of this tutorial are also available; one uses the Verilog hardware description language and the other is based on defining the desired circuit in the form of a schematic diagram.

The last step in the design process involves configuring the designed circuit in an actual FPGA device. To show how this is done, it is assumed that the user has access to the Altera DE-series Development and Education board connected to a computer that has Quartus II software installed. A reader who does not have access to the DE-series board will still find the tutorial useful to learn how the FPGA programming and configuration task is performed.

The screen captures in the tutorial were obtained using the Quartus II version 14.0; if other versions of the software are used, some of the images may be slightly different.

Contents:

2 Background

Computer Aided Design (CAD) software makes it easy to implement a desired logic circuit by using a programmable logic device, such as a field-programmable gate array (FPGA) chip. A typical FPGA CAD flow is illustrated in Figure 1.

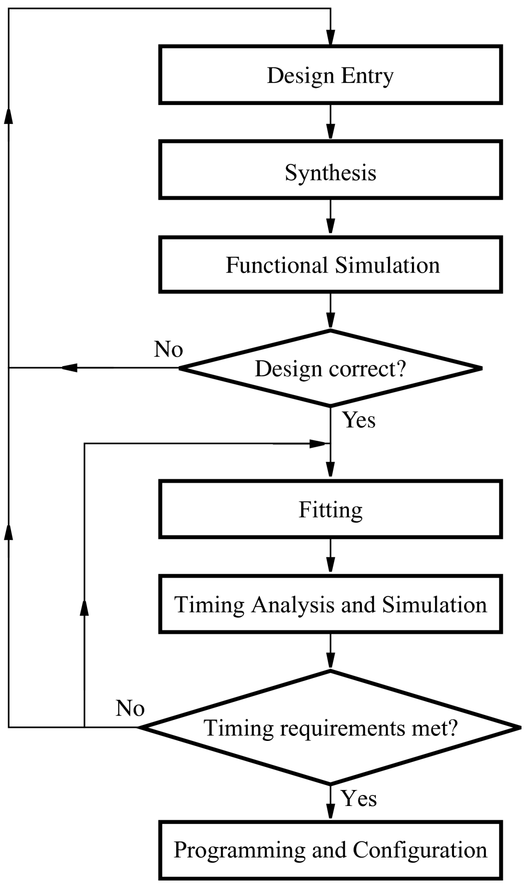


Figure 1. Typical CAD flow.

The CAD flow involves the following steps:

- **Design Entry** – the desired circuit is specified either by means of a schematic diagram, or by using a hardware description language, such as Verilog or VHDL
- **Synthesis** – the entered design is synthesized into a circuit that consists of the logic elements (LEs) provided in the FPGA chip
- **Functional Simulation** – the synthesized circuit is tested to verify its functional correctness; this simulation does not take into account any timing issues

- **Fitting** – the CAD Fitter tool determines the placement of the LEs defined in the netlist into the LEs in an actual FPGA chip; it also chooses routing wires in the chip to make the required connections between specific LEs
- **Timing Analysis** – propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit
- **Timing Simulation** – the fitted circuit is tested to verify both its functional correctness and timing
- **Programming and Configuration** – the designed circuit is implemented in a physical FPGA chip by programming the configuration switches that configure the LEs and establish the required wiring connections

This tutorial introduces the basic features of the Quartus II software. It shows how the software can be used to design and implement a circuit specified by using the VHDL hardware description language. It makes use of the graphical user interface to invoke the Quartus II commands. Doing this tutorial, the reader will learn about:

- Creating a project
- Design entry using VHDL code
- Synthesizing a circuit specified in VHDL code
- Fitting a synthesized circuit into an Altera FPGA
- Assigning the circuit inputs and outputs to specific pins on the FPGA
- Simulating the designed circuit
- Programming and configuring the FPGA chip on Altera's DE-series board

3 Getting Started

Each logic circuit, or subcircuit, being designed with Quartus II software is called a *project*. The software works on one project at a time and keeps all information for that project in a single directory (folder) in the file system. To begin a new logic circuit design, the first step is to create a directory to hold its files. To hold the design files for this tutorial, we will use a directory *introtutorial*. The running example for this tutorial is a simple circuit for two-way light control.

Start the Quartus II software. You should see a display similar to the one in Figure 2. This display consists of several windows that provide access to all the features of Quartus II software, which the user selects with the computer mouse. Most of the commands provided by Quartus II software can be accessed by using a set of menus that are located below the title bar. For example, in Figure 2 clicking the left mouse button on the menu named File opens the menu shown in Figure 3. Clicking the left mouse button on the entry Exit exits from Quartus II software. In general, whenever the mouse is used to select something, the *left* button is used. Hence we will not normally specify which button to press. In the few cases when it is necessary to use the *right* mouse button, it will be specified explicitly.

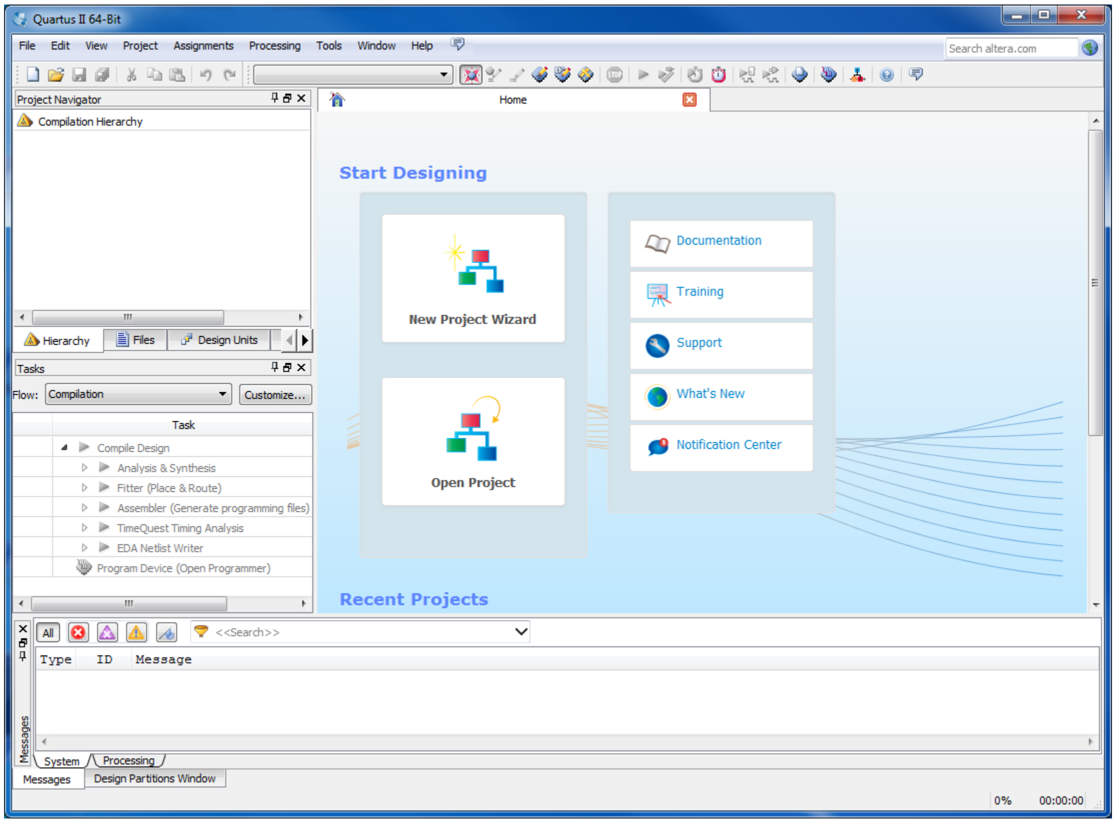


Figure 2. The main Quartus II display.

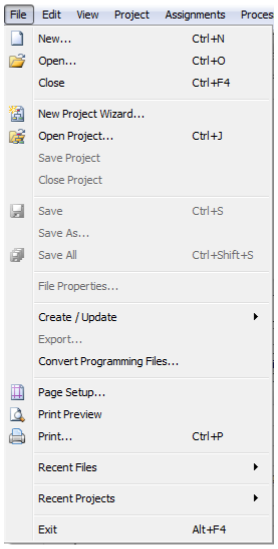


Figure 3. An example of the File menu.

For some commands it is necessary to access two or more menus in sequence. We use the convention Menu1 > Menu2 > Item to indicate that to select the desired command the user should first click the left mouse button on Menu1, then within this menu click on Menu2, and then within Menu2 click on Item. For example, File > Exit uses the mouse to exit from the system. Many commands can be invoked by clicking on an icon displayed in one of the toolbars. To see the command associated with an icon, position the mouse over the icon and a tooltip will appear that displays the command name.

3.1 Quartus II Online Help

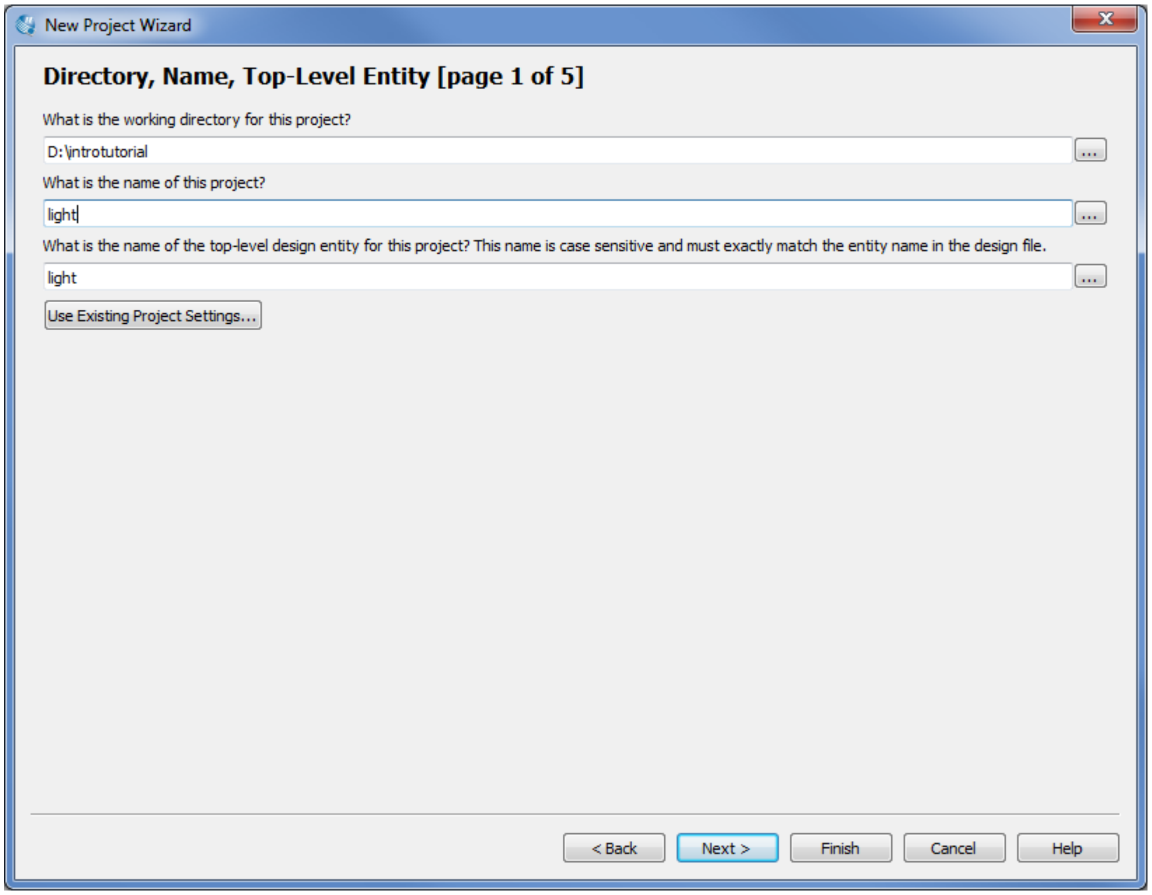
Quartus II software provides comprehensive online documentation that answers many of the questions that may arise when using the software. The documentation is accessed from the Help menu. To get some idea of the extent of documentation provided, it is worthwhile for the reader to browse through the Help menu.

The user can quickly search through the Help topics by selecting Help > Search, which opens a dialog box into which keywords can be entered. Another method, context-sensitive help, is provided for quickly finding documentation for specific topics. While using most applications, pressing the F1 function key on the keyboard opens a Help display that shows the commands available for the application.

4 Starting a New Project

To start working on a new design we first have to define a new *design project*. Quartus II software makes the designer's task easy by providing support in the form of a *wizard*. Create a new project as follows:

1. Select File > New Project Wizard and click Next to reach the window in Figure 4, which asks for the name and directory of the project.


New Project Wizard

Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?

D:\introtutorial

What is the name of this project?

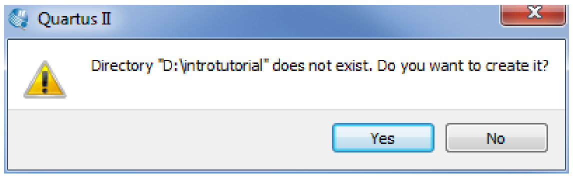
light

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.

light

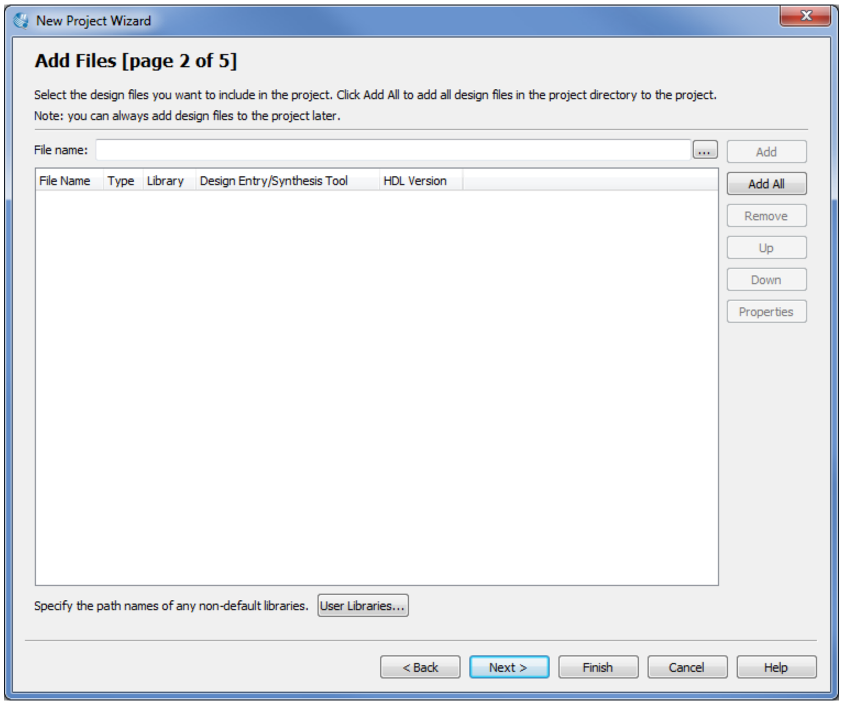
Figure 4. Creation of a new project.

2. Set the working directory to be *introtutorial*; of course, you can use some other directory name of your choice if you prefer. The project must have a name, which is usually the same as the top-level design entity that will be included in the project. Choose *light* as the name for both the project and the top-level entity, as shown in Figure 4. Press Next. Since we have not yet created the directory *introtutorial*, Quartus II software displays the pop-up box in Figure 5 asking if it should create the desired directory. Click Yes, which leads to the window in Figure 6.


Quartus II

Directory "D:\introtutorial" does not exist. Do you want to create it?

Figure 5. Quartus II software can create a new directory for the project.

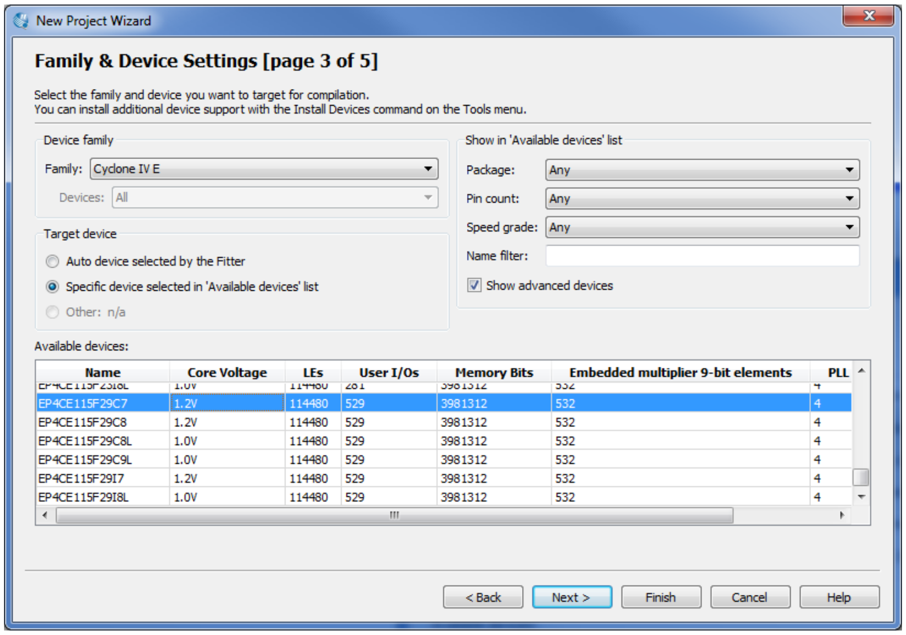


Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.
Note: you can always add design files to the project later.

Specify the path names of any non-default libraries.

Figure 6. The wizard can include user-specified design files.

3. The wizard makes it easy to specify which existing files (if any) should be included in the project. Assuming that we do not have any existing files, click Next, which leads to the window in Figure 7.


New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

Device family
Family: Cyclone IV E
Devices: All

Target device
Auto device selected by the Fitter
[✓ Specific device selected in 'Available devices' list]
Other: n/a

Show in 'Available devices' list
Package: Any
Pin count: Any
Speed grade: Any
Name filter:
[✓] Show advanced devices

Available devices:

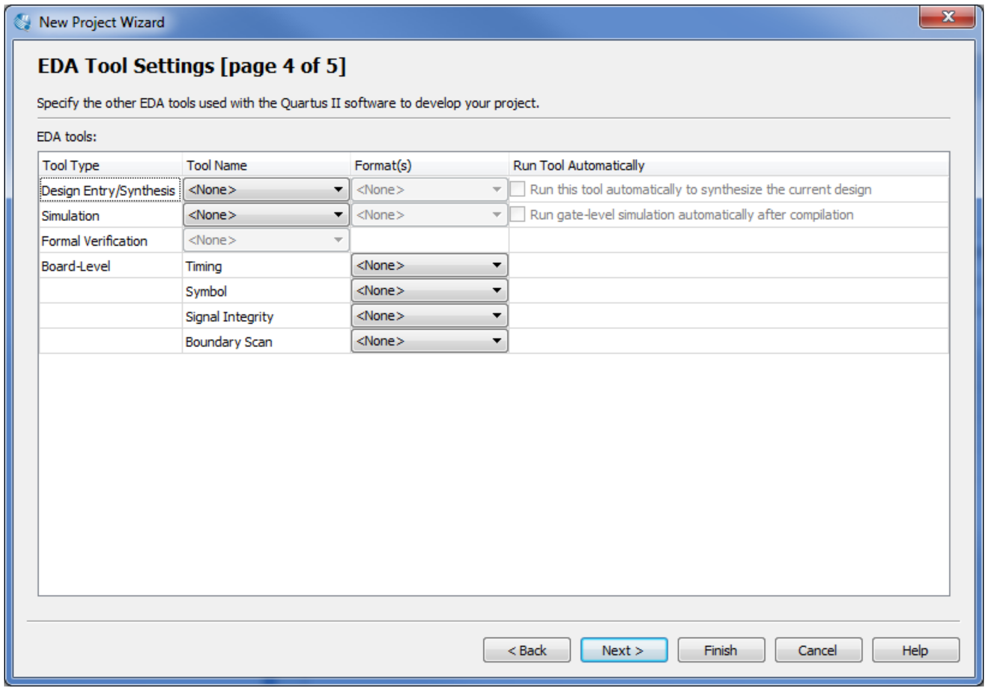
Name	Core Voltage	LEs	User I/Os	Memory Bits	Embedded multiplier 9-bit elements	PLL
EP4CE115F29C7	1.2V	114480	529	3981312	532	4
EP4CE115F29C8	1.2V	114480	529	3981312	532	4
EP4CE115F29C8L	1.0V	114480	529	3981312	532	4
EP4CE115F29C9L	1.0V	114480	529	3981312	532	4
EP4CE115F29I7	1.2V	114480	529	3981312	532	4
EP4CE115F29I8L	1.0V	114480	529	3981312	532	4

Figure 7. Choose the device family and a specific device.

4. We have to specify the type of device in which the designed circuit will be implemented. Choose the Cyclone-series device family for your DE-series board. We can let Quartus II software select a specific device in the family, or we can choose the device explicitly. We will take the latter approach. From the list of available devices, choose the appropriate device name for your DE-series board. A list of devices names on DE-series boards can be found in Table 1. Press Next, which opens the window in Figure 8.

Board	Device Name
DE0-Nano	Cyclone IVE EP4CE22F17C6
DE1-SoC	Cyclone V SoC 5CSEMA5F31C6
DE2-115	Cyclone IVE EP4CE115F29C7

Table 1. DE-series FPGA device names



EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

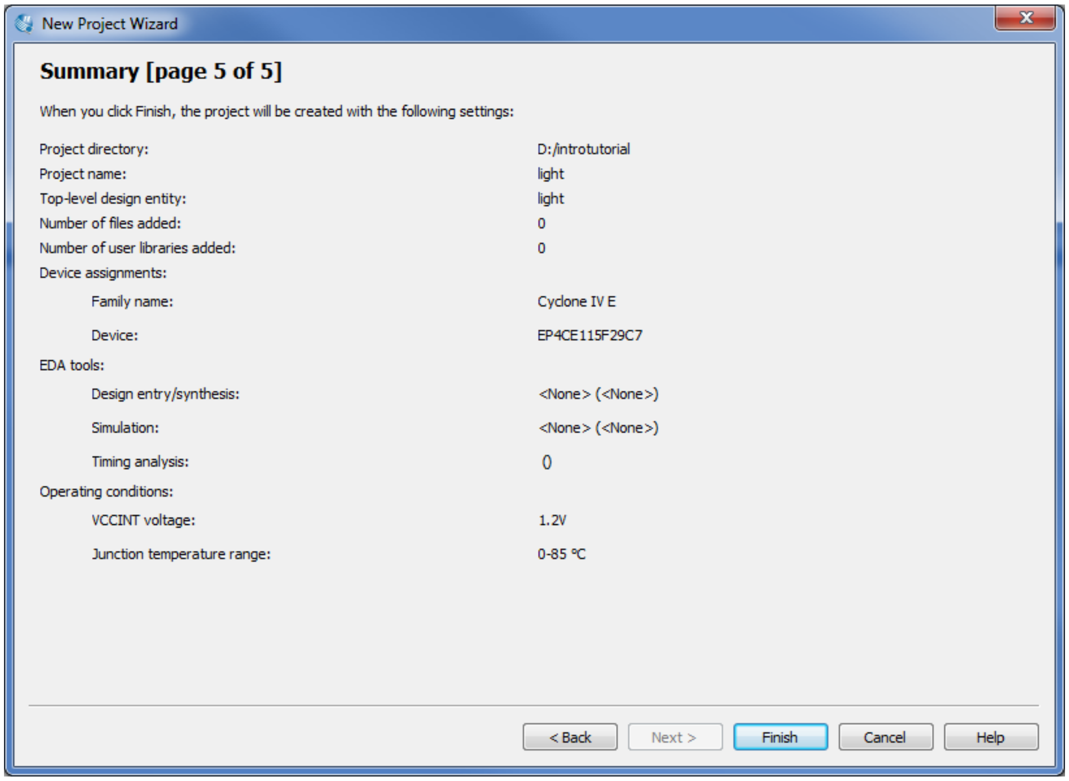
EDA tools:

Tool Type	Tool Name	Format(s)	Run Tool Automatically
Design Entry/Synthesis	<None>	<None>	Run this tool automatically to synthesize the current design
Simulation	<None>	<None>	Run gate-level simulation automatically after compilation
Formal Verification	<None>		
Board-Level	Timing	<None>	
	Symbol	<None>	
	Signal Integrity	<None>	
	Boundary Scan	<None>	

Figure 8. Other EDA tools can be specified.

5. The user can specify any third-party tools that should be used. A commonly used term for CAD software for electronic circuits is *EDA tools*, where the acronym stands for Electronic Design Automation. This term is used in Quartus II messages that refer to third-party tools, which are the tools developed and marketed by companies other than Altera. Since we will rely solely on Quartus II tools, we will not choose any other tools. Press Next.

6. A summary of the chosen settings appears in the screen shown in Figure 9. Press Finish, which returns to the main Quartus II window, but with *light* specified as the new project, in the display title bar, as indicated in Figure 10.



Summary [page 5 of 5]

When you click Finish, the project will be created with the following settings:

Project directory:	D:/introtutorial
Project name:	light
Top-level design entity:	light
Number of files added:	0
Number of user libraries added:	0
Device assignments:	
Family name:	Cyclone IV E
Device:	EP4CE115F29C7
EDA tools:	
Design entry/synthesis:	<None> (<None>)
Simulation:	<None> (<None>)
Timing analysis:	0
Operating conditions:	
VCCINT voltage:	1.2V
Junction temperature range:	0-85 °C

Figure 9. Example summary of a DE2-115 board project settings.

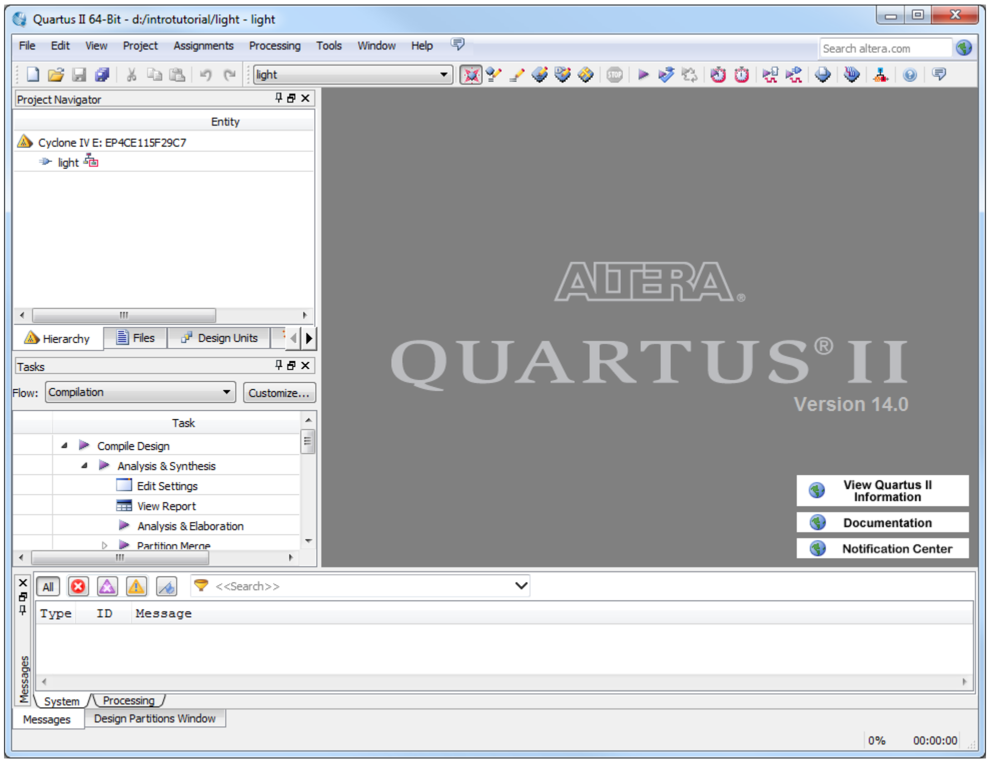
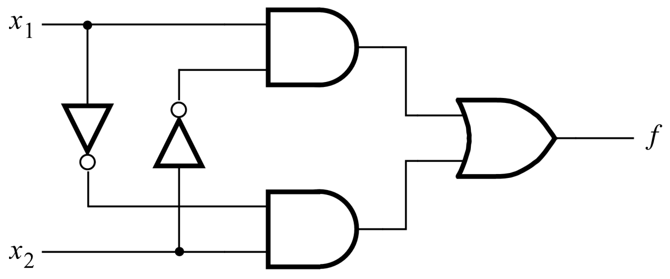


Figure 10. The Quartus II display for created project on a DE2-115 board.

5 Design Entry Using VHDL Code

As a design example, we will use the two-way light controller circuit shown in Figure 11. The circuit can be used to control a single light from either of the two switches, x_1 and x_2, where a closed switch corresponds to the logic value 1. The truth table for the circuit is also given in the figure. Note that this is just the Exclusive-OR function of the inputs x_1 and x_2, but we will specify it using the gates shown.



x_1	x_2	f
0	0	0
0	1	1
1	0	1
1	1	0

Figure 11. The light controller circuit.

The required circuit is described by the VHDL code in Figure 12. Note that the VHDL entity is called *light* to match the name given in Figure 4, which was specified when the project was created. This code can be typed into a file by using any text editor that stores ASCII files, or by using the Quartus II text editing facilities. While the file can be given any name, it is a common designers' practice to use the same name as the name of the top-level VHDL entity. The file name must include the extension *vhd*, which indicates a VHDL file. So, we will use the name *light.vhd*.

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY light IS
    PORT ( x1, x2 : IN     STD_LOGIC ;
                f     : OUT  STD_LOGIC ) ;
END light ;
ARCHITECTURE LogicFunction OF light IS
BEGIN
    f <= (x1 AND NOT x2) OR (NOT x1 AND x2);
END LogicFunction ;
```

Figure 12. VHDL code for the circuit in Figure 11.

5.1 Using the Quartus II Text Editor

This section shows how to use the Quartus II Text Editor. You can skip this section if you prefer to use some other text editor to create the VHDL source code file, which we will name *light.vhd*.

Select File > New to get the window in Figure 13, choose VHDL File, and click OK. This opens the Text Editor window. The first step is to specify a name for the file that will be created. Select File > Save As to open the pop-up box depicted in Figure 14. In the box labeled Save as type choose VHDL File. In the box labeled File name type *light*. Put a checkmark in the box Add file to current project. Click Save, which puts the file into the directory *introtutorial* and leads to the Text Editor window shown in Figure 15. Enter the VHDL code in Figure 12 into the Text Editor and save the file by typing File > Save, or by typing the shortcut Ctrl-s.

Most of the commands available in the Text Editor are self-explanatory. Text is entered at the *insertion point*, which is indicated by a thin vertical line. The insertion point can be moved either by using the keyboard arrow keys or by using the mouse. Two features of the Text Editor are especially convenient for typing VHDL code. First, the editor can display different types of VHDL statements in different colors, which is the default choice. Second, the editor can automatically indent the text on a new line so that it matches the previous line. Such options can be controlled by the settings in Tools > Options > Text Editor.

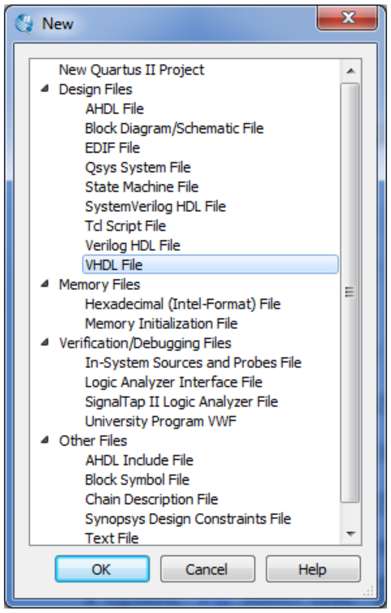
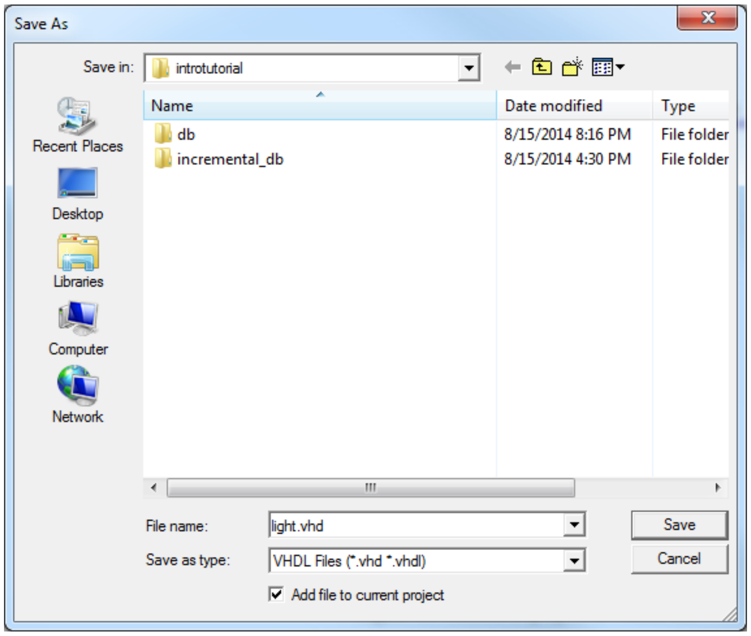


Figure 13. Choose to prepare a VHDL file.



Name	Date modified	Type
db	8/15/2014 8:16 PM	File folder
incremental_db	8/15/2014 4:30 PM	File folder

File name: light.vhd

Save as type: VHDL Files (*.vhd *.vhdl)

☑ Add file to current project

Figure 14. Name the file.

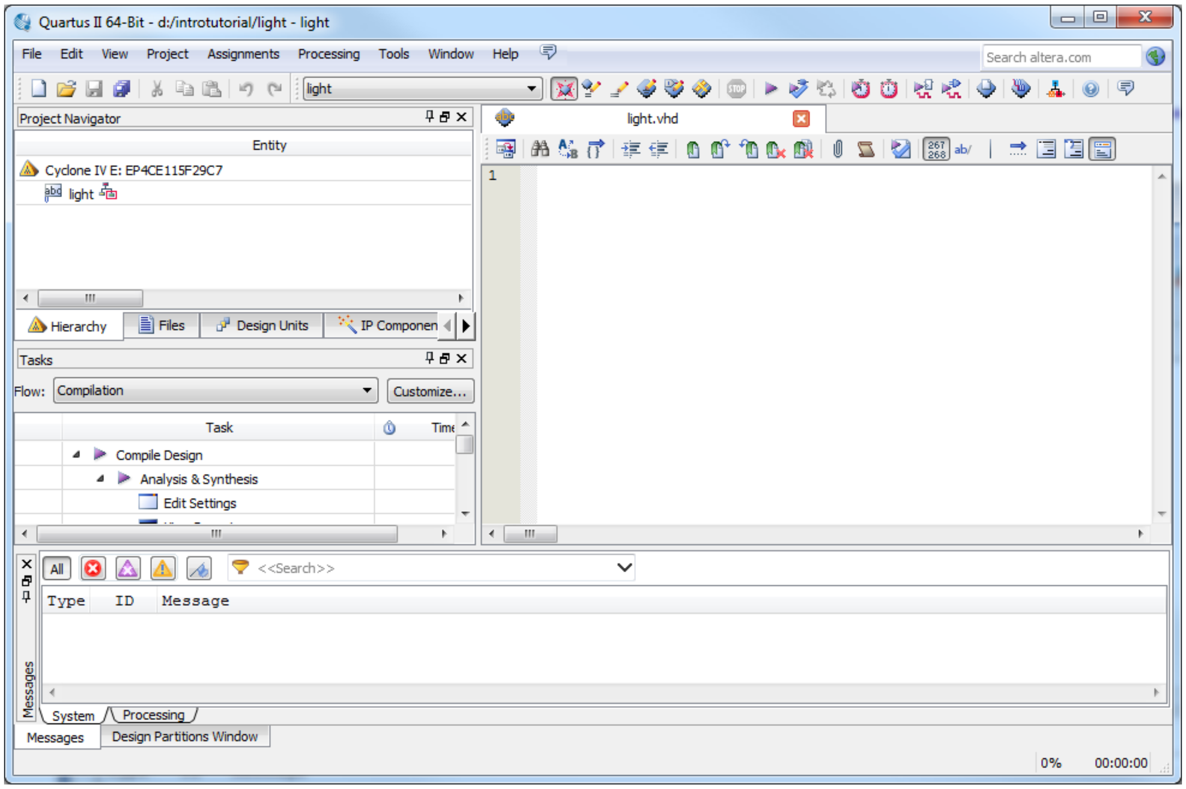

Quartus II 64-Bit - d:/introtutorial/light - light
File Edit View Project Assignments Processing Tools Window Help
light
Project Navigator
Entity
Cyclone IV E: EP4CE115F29C7
light
Hierarchy Files Design Units IP Componen
Tasks
Flow: Compilation
Customize...
Task
Compile Design
Analysis & Synthesis
Edit Settings
light.vhd
Type ID Message
System Processing
Messages Design Partitions Window
0% 00:00:00

Figure 15. Text Editor window.

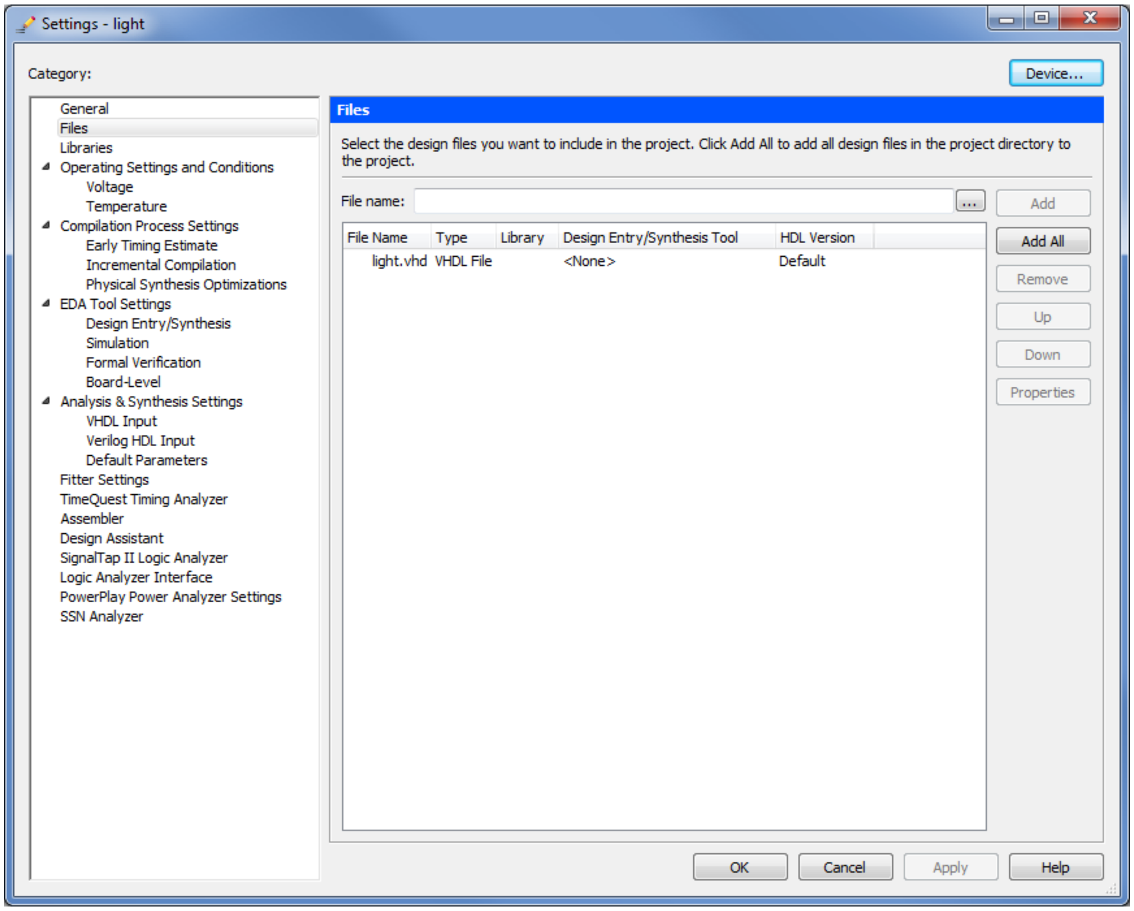
5.1.1 Using VHDL Templates

The syntax of VHDL code is sometimes difficult for a designer to remember. To help with this issue, the Text Editor provides a collection of *VHDL templates*. The templates provide examples of various types of VHDL statements, such as an **ENTITY** declaration, a **CASE** statement, and assignment statements. It is worthwhile to browse through the templates by selecting Edit > Insert Template > VHDL to become familiar with this resource.

5.2 Adding Design Files to a Project

As we indicated when discussing Figure 6, you can tell Quartus II software which design files it should use as part of the current project. To see the list of files already included in the *light* project, select Assignments > Settings, which leads to the window in Figure 16. As indicated on the left side of the figure, click on the item Files. An alternative way of making this selection is to choose Project > Add/Remove Files in Project.

If you used the Quartus II Text Editor to create the file and checked the box labeled Add file to current project, as described in Section 5.1, then the *light.vhd* file is already a part of the project and will be listed in the window in Figure 16. Otherwise, the file must be added to the project. So, if you did not use the Quartus II Text Editor, then place a copy of the file *light.vhd*, which you created using some other text editor, into the directory *introtutorial*. To add this file to the project, click on the **...** button next to the box labelled File name in Figure 16 to get the pop-up window in Figure 17. Select the *light.vhd* file and click Open. The selected file is now indicated in the File name box in of Figure 16. Click Add then OK to include the *light.vhd* file in the project. We should mention that in many cases the Quartus II software is able to automatically find the right files to use for each entity referenced in VHDL code, even if the file has not been explicitly added to the project. However, for complex projects that involve many files it is a good design practice to specifically add the needed files to the project, as described above.

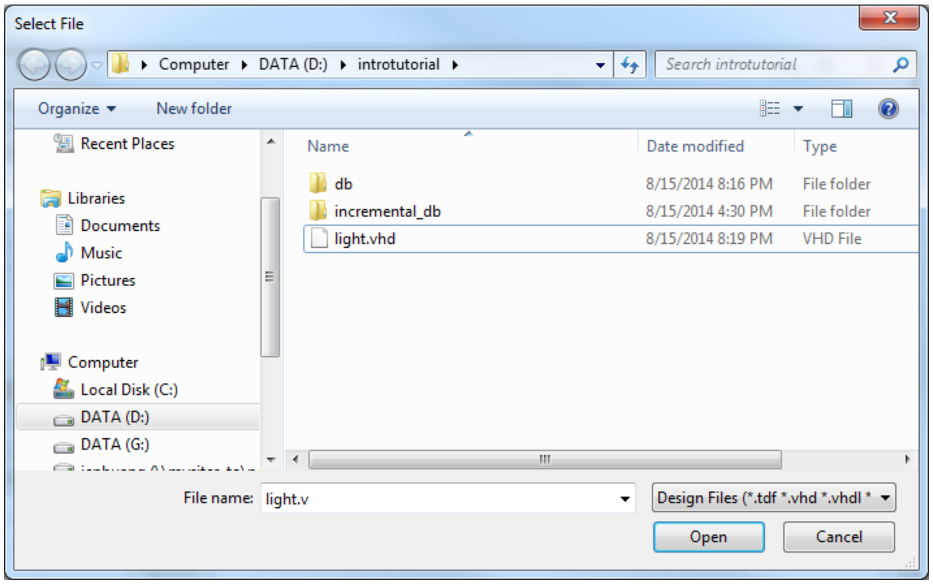


Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

File name:

File Name	Type	Library	Design Entry/Synthesis Tool	HDL Version
light.vhd	VHDL File		<None>	Default

Figure 16. Settings window.



Name	Date modified	Type
db	8/15/2014 8:16 PM	File folder
incremental_db	8/15/2014 4:30 PM	File folder
light.vhd	8/15/2014 8:19 PM	VHD File

File name: light.v

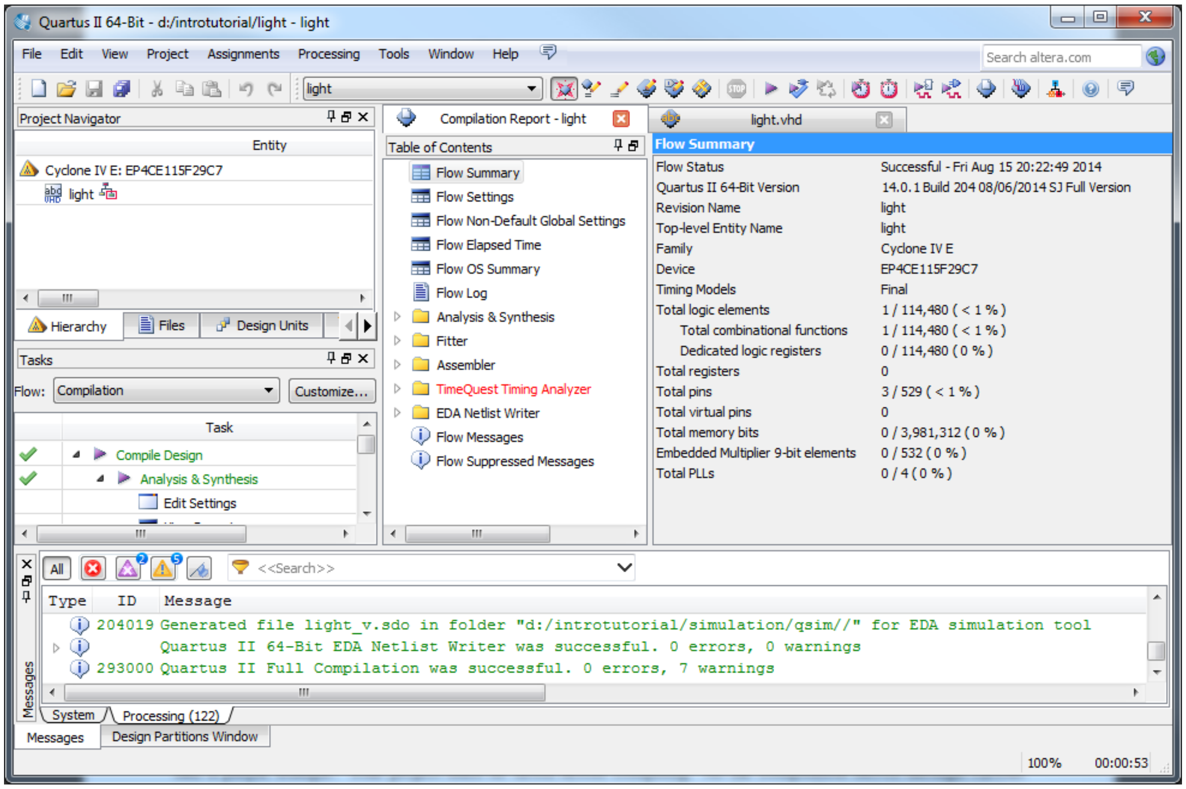
Design Files (*.tdf *.vhd *.vhdl *

Figure 17. Select the file.

6 Compiling the Designed Circuit

The VHDL code in the file *light.vhd* is processed by several Quartus II tools that analyze the code, synthesize the circuit, and generate an implementation of it for the target chip. These tools are controlled by the application program called the *Compiler*.

Run the Compiler by selecting Processing > Start Compilation, or by clicking on the toolbar icon ▶ that looks like a purple triangle. Your project must be saved before compiling. As the compilation moves through various stages, its progress is reported in a window on the left side of the Quartus II display. Successful (or unsuccessful) compilation is indicated in a pop-up box. Acknowledge it by clicking OK, which leads to the Quartus II display in Figure 18. In the message window, at the bottom of the figure, various messages are displayed. In case of errors, there will be appropriate messages given.



Flow Summary

Flow Status	Successful - Fri Aug 15 20:22:49 2014
Quartus II 64-Bit Version	14.0.1 Build 204 08/06/2014 SJ Full Version
Revision Name	light
Top-level Entity Name	light
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	1 / 114,480 (< 1 %)
Total combinational functions	1 / 114,480 (< 1 %)
Dedicated logic registers	0 / 114,480 (0 %)
Total registers	0
Total pins	3 / 529 (< 1 %)
Total virtual pins	0
Total memory bits	0 / 3,981,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 532 (0 %)
Total PLLs	0 / 4 (0 %)

```
Type  ID     Message
      204019 Generated file light_v.sdo in folder "d:/introtutorial/simulation/qsim//" for EDA simulation tool
             Quartus II 64-Bit EDA Netlist Writer was successful. 0 errors, 0 warnings
      293000 Quartus II Full Compilation was successful. 0 errors, 7 warnings
```

Figure 18. Display after a successful compilation.

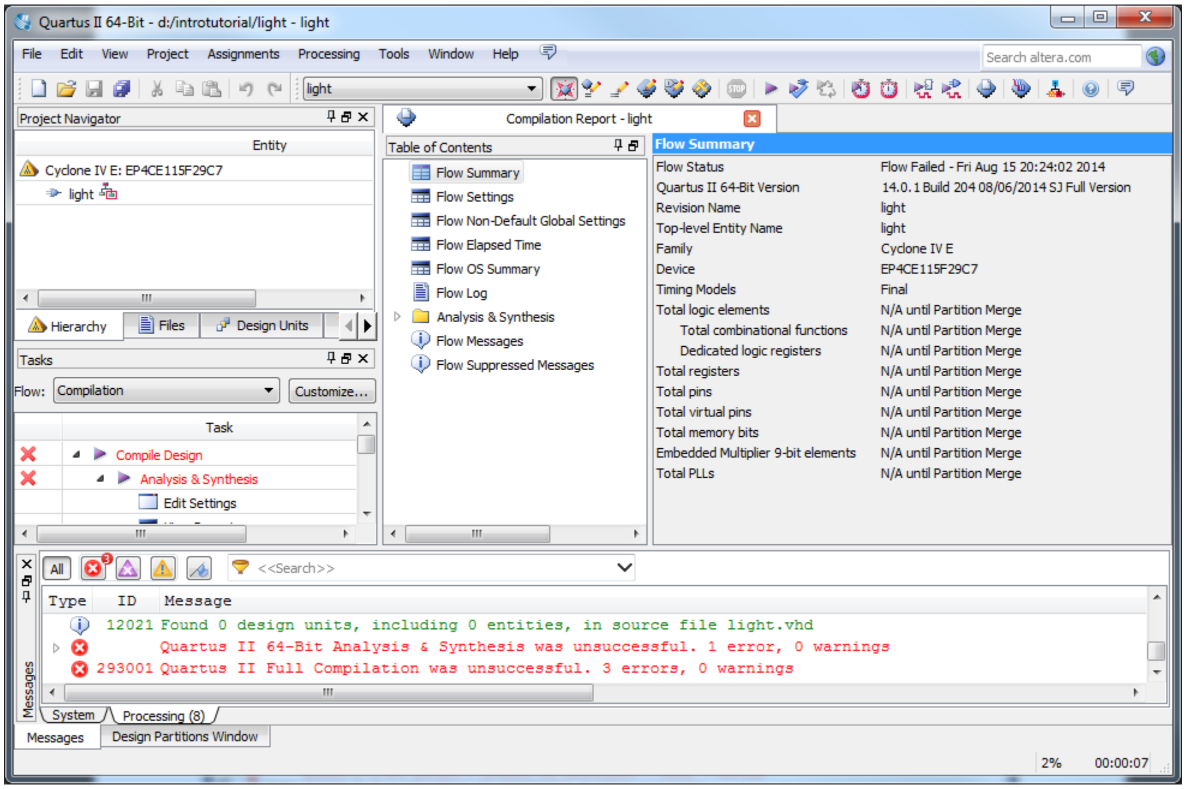
When the compilation is finished, a compilation report is produced. A tab showing this report is opened automatically, as seen in Figure 18. The tab can be closed in the normal way, and it can be opened at any time either by selecting Processing > Compilation Report or by clicking on the icon . The report includes a number of sections listed on the left side. Figure 18 displays the Compiler Flow Summary section, which indicates that only one logic element and three pins are needed to implement this tiny circuit on the selected FPGA chip.

6.1 Errors

Quartus II software displays messages produced during compilation in the Messages window. If the VHDL design file is correct, one of the messages will state that the compilation was successful and that there are no errors.

If the Compiler does not report zero errors, then there is at least one mistake in the VHDL code. In this case a message corresponding to each error found will be displayed in the Messages window. Double-clicking on an error message will highlight the offending statement in the VHDL code in the Text Editor window. Similarly, the Compiler may display some warning messages. Their details can be explored in the same way as in the case of error messages. The user can obtain more information about a specific error or warning message by selecting the message and pressing the F1 function key.

To see the effect of an error, open the file *light.vhd*. Remove the semicolon in the statement that defines the function *f*, illustrating a typographical error that is easily made. Compile the erroneous design file by clicking on the icon. A pop-up box will ask if the changes made to the *light.vhd* file should be saved; click Yes. After trying to compile the circuit, Quartus II software will display a pop-up box indicating that the compilation was not successful. Acknowledge it by clicking OK. The compilation report summary, given in Figure 19, now confirms the failed result. In the Table of Contents panel, expand the Analysis & Synthesis part of the report and then select Messages to have the messages displayed as shown in Figure 20. The Compilation Report can be dispayed as a separate window as in Figure 20 by right-clicking its tab and selecting Detach Window, and can be reattached by clicking Window > Attach Window. Double-click on the first error message. Quartus II software responds by opening the *light.vhd* file and highlighting the statement which is affected by the error, as shown in Figure 21. Correct the error and recompile the design.

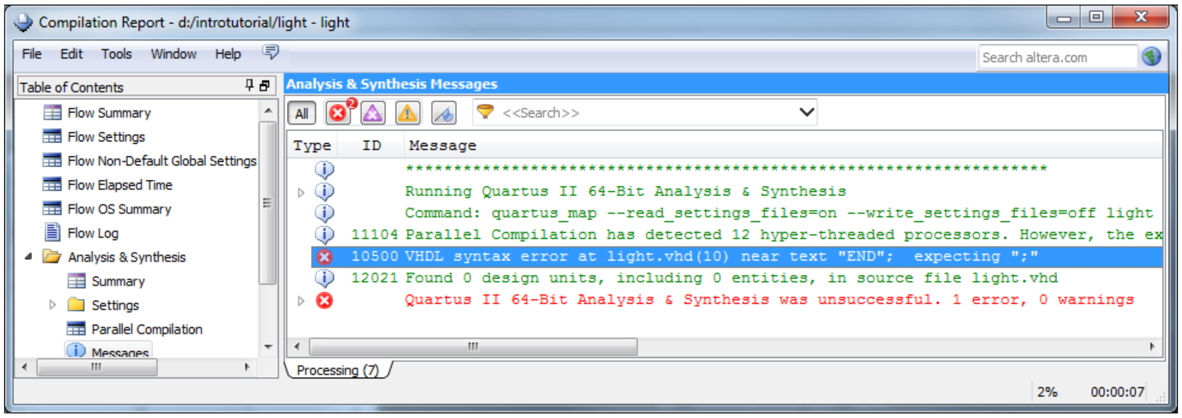


Flow Summary

Flow Status	Flow Failed - Fri Aug 15 20:24:02 2014
Quartus II 64-Bit Version	14.0.1 Build 204 08/06/2014 SJ Full Version
Revision Name	light
Top-level Entity Name	light
Family	Cyclone IV E
Device	EP4CE115F29C7
Timing Models	Final
Total logic elements	N/A until Partition Merge
Total combinational functions	N/A until Partition Merge
Dedicated logic registers	N/A until Partition Merge
Total registers	N/A until Partition Merge
Total pins	N/A until Partition Merge
Total virtual pins	N/A until Partition Merge
Total memory bits	N/A until Partition Merge
Embedded Multiplier 9-bit elements	N/A until Partition Merge
Total PLLs	N/A until Partition Merge

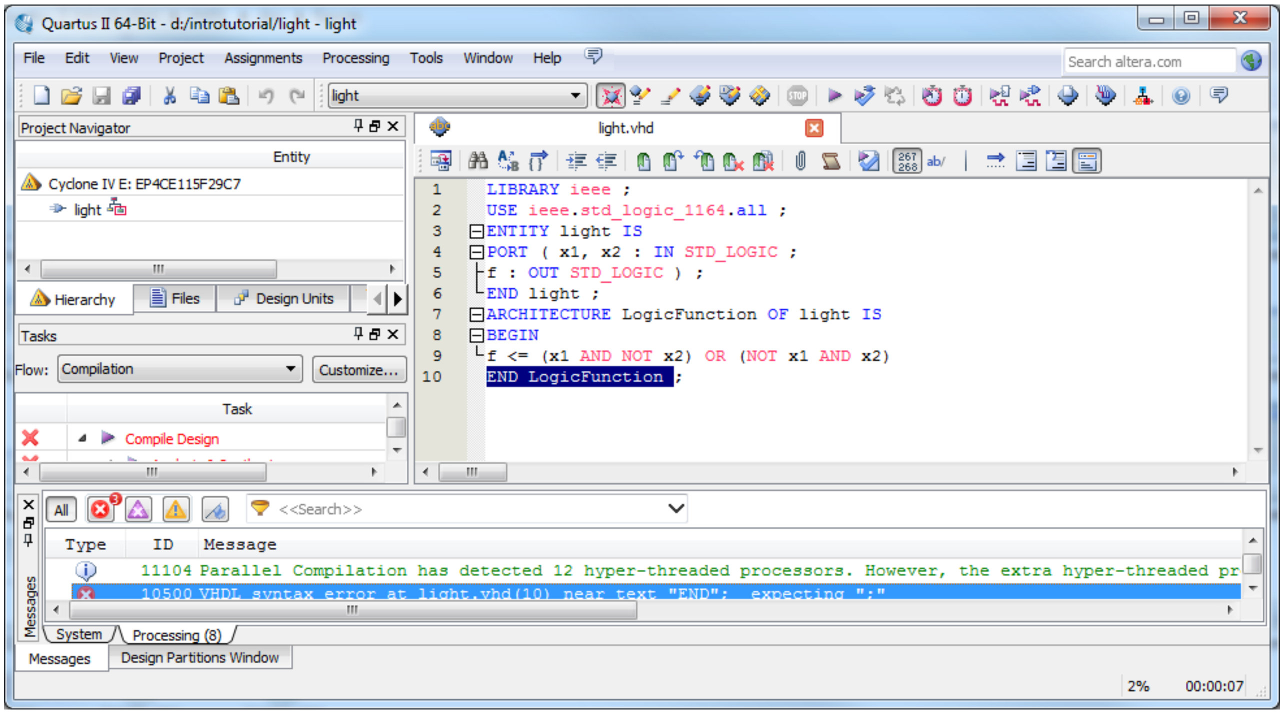
```
Type  ID     Message
      12021  Found 0 design units, including 0 entities, in source file light.vhd
             Quartus II 64-Bit Analysis & Synthesis was unsuccessful. 1 error, 0 warnings
      293001 Quartus II Full Compilation was unsuccessful. 3 errors, 0 warnings
```

Figure 19. Compilation report for the failed design.



```
Type  ID    Message
            ******************************************************************
            Running Quartus II 64-Bit Analysis & Synthesis
            Command: quartus_map --read_settings_files=on --write_settings_files=off light
      11104 Parallel Compilation has detected 12 hyper-threaded processors. However, the ex
      10500 VHDL syntax error at light.vhd(10) near text "END";  expecting ";"
      12021 Found 0 design units, including 0 entities, in source file light.vhd
            Quartus II 64-Bit Analysis & Synthesis was unsuccessful. 1 error, 0 warnings
```

Figure 20. Error messages.



```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY light IS
PORT ( x1, x2 : IN STD_LOGIC ;
f : OUT STD_LOGIC ) ;
END light ;
ARCHITECTURE LogicFunction OF light IS
BEGIN
f <= (x1 AND NOT x2) OR (NOT x1 AND x2)
END LogicFunction ;
```

```
Type  ID    Message
      11104 Parallel Compilation has detected 12 hyper-threaded processors. However, the extra hyper-threaded pr
      10500 VHDL syntax error at light.vhd(10) near text "END";  expecting ";"
```

Figure 21. Identifying the location of the error.

7 Pin Assignment

During the compilation above, the Quartus II Compiler was free to choose any pins on the selected FPGA to serve as inputs and outputs. However, the DE-series board has hardwired connections between the FPGA pins and the other components on the board. We will use two toggle switches, labeled SW_0 and SW_1, to provide the external inputs, x_1

and x_2, to our example circuit. These switches are connected to the FPGA pins listed in Table 2. We will connect the output f to the green light-emitting diode labeled $LEDG_0$. Its FPGA pin assignment can also be found in Table 2. Note that on the DE0-Nano board the lights are called *LED*, and on the DE1-SoC board they are called *LEDR*.

Component	DE0-Nano	DE2-115	DE1-SoC
SW_0	PIN_M1	PIN_AB28	PIN_AB12
SW_1	PIN_T8	PIN_AC28	PIN_AC12
$LEDG_0$, LED_0, or $LEDR_0$	PIN_A15	PIN_E21	PIN_V16

Table 2. DE-Series Pin Assignments

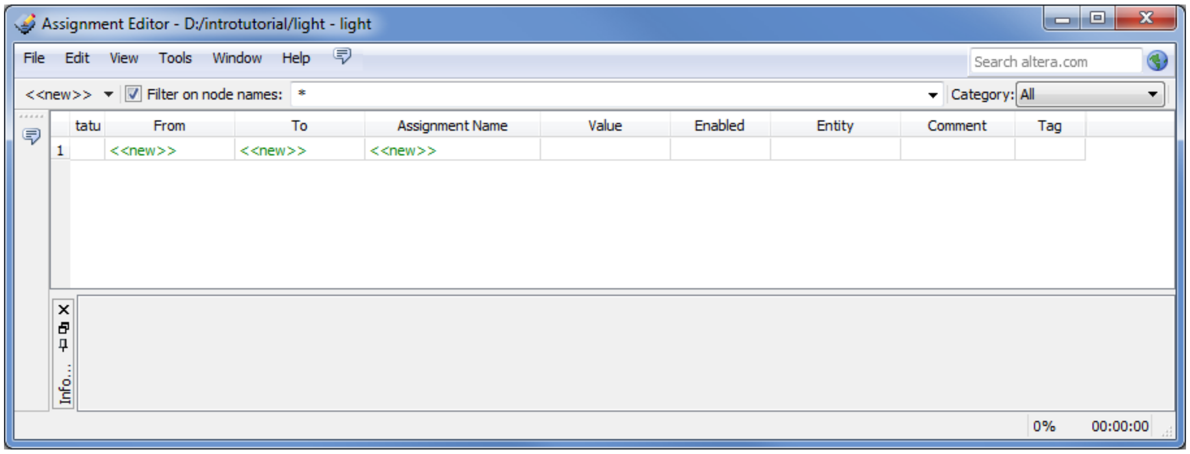
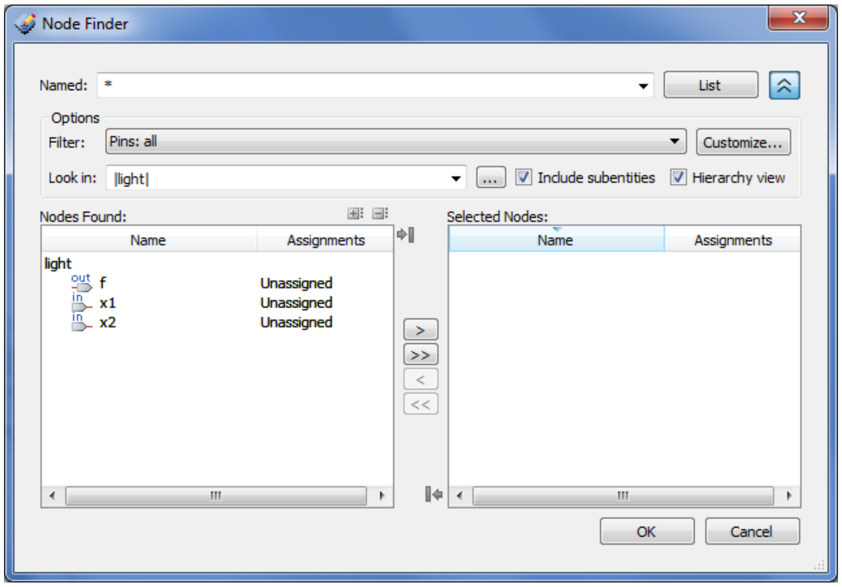

Assignment Editor - D:/introtutorial/light - light
File Edit View Tools Window Help
Search altera.com
<<new>> Filter on node names: *
Category: All
tatu From To Assignment Name Value Enabled Entity Comment Tag
1 <<new>> <<new>> <<new>>
Info...
0% 00:00:00

Figure 22. The Assignment Editor window.

Pin assignments are made by using the Assignment Editor. Select Assignments > Assignment Editor to reach the window in Figure 22 (shown here as a detached window). In the Category drop-down menu select All. Click on the <<new>> button located near the top left corner to make a new item appear in the table. Double click the box under the column labeled To so that the Node Finder button appears. Click on the button (not the drop down arrow) to reach the window in Figure 23. Click on to show more search options. In the Filter drop-down menu select Pins: all. Then click the List button to display the input and output pins to be assigned: f, $x1$, and $x2$. Click on $x1$ as the first pin to be assigned and click the > button; this will enter $x1$ in the Selected Nodes box. Click OK. $x1$ will now appear in the box under the column labeled To. Alternatively, the node name can be entered directly by double-clicking the box under the To column and typing in the node name.

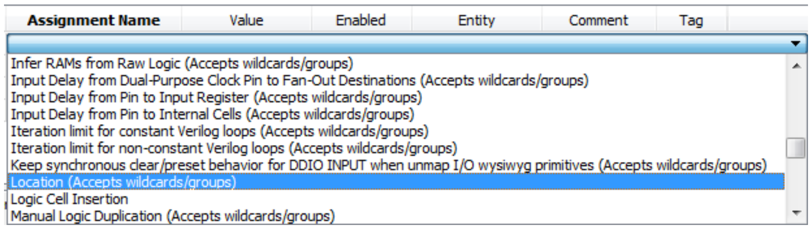
Follow this by double-clicking on the box to the right of this new $x1$ entry, in the column labeled Assignment Name. Now, the drop-down menu in Figure 24 appears. Scroll down and select Location (Accepts wildcards/groups). Instead of scrolling down the menu to find the desired item, you can just type the first letter of the item in the Assignment Name box. In this case the desired item happens to be the first item beginning with L. Finally, double-click the box in the column labeled Value. Type the pin assignment corresponding to SW_0 for your DE-series board, as listed in Table 2.

Use the same procedure to assign input *x*2 and output *f* to the appropriate pins listed in Table 2. An example using a DE2-115 board is shown in Figure 25. To save the assignments made, choose File > Save. You can also simply close the Assignment Editor window, in which case a pop-up box will ask if you want to save the changes to assignments; click Yes. Recompile the circuit, so that it will be compiled with the correct pin assignments.



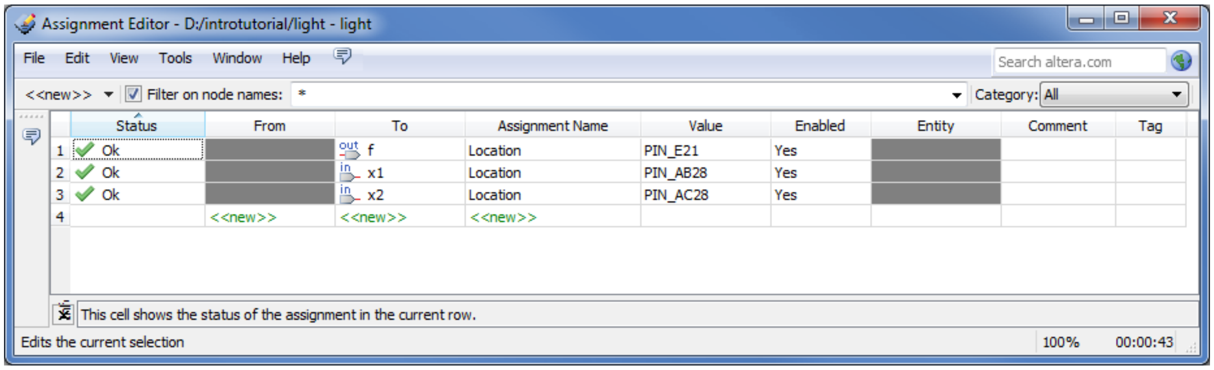
Name	Assignments
light	
f	Unassigned
x1	Unassigned
x2	Unassigned

Figure 23. The Node Finder displays the input and output names.



Infer RAMs from Raw Logic (Accepts wildcards/groups)
Input Delay from Dual-Purpose Clock Pin to Fan-Out Destinations (Accepts wildcards/groups)
Input Delay from Pin to Input Register (Accepts wildcards/groups)
Input Delay from Pin to Internal Cells (Accepts wildcards/groups)
Iteration limit for constant Verilog loops (Accepts wildcards/groups)
Iteration limit for non-constant Verilog loops (Accepts wildcards/groups)
Keep synchronous clear/preset behavior for DDIO INPUT when unmap I/O wysiwyg primitives (Accepts wildcards/groups)
Location (Accepts wildcards/groups)
Logic Cell Insertion
Manual Logic Duplication (Accepts wildcards/groups)

Figure 24. The available assignment names for a DE2-115 board.


Assignment Editor - D:/introtutorial/light - light

	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓ Ok		f	Location	PIN_E21	Yes			
2	✓ Ok		x1	Location	PIN_AB28	Yes			
3	✓ Ok		x2	Location	PIN_AC28	Yes			
4		<<new>>	<<new>>	<<new>>					

This cell shows the status of the assignment in the current row.

Edits the current selection

Figure 25. The complete assignment on a DE2-115 board.

The DE-series board has fixed pin assignments. Having finished one design, the user will want to use the same pin assignment for subsequent designs. Going through the procedure described above becomes tedious if there are many pins used in the design. A useful Quartus II feature allows the user to both export and import the pin assignments from a special file format, rather than creating them manually using the Assignment Editor. A simple file format that can be used for this purpose is the *Quartus II Settings File (QSF)* format. The format for the file for our simple project (on a DE2-115 board) is

```
set_location_assignment PIN_AB28 -to x1
set_location_assignment PIN_AC28 -to x2
set_location_assignment PIN_E21 -to f
```

By adding lines to the file, any number of pin assignments can be created. Such *qsf* files can be imported into any design project.

If you created a pin assignment for a particular project, you can export it for use in a different project. To see how this is done, open again the Assignment Editor to reach the window in Figure 25. Select Assignments > Export Assignment which leads to the window in Figure 26. Here, the file *light.qsf* is available for export. Click on OK. If you now look in the directory, you will see that the file *light.qsf* has been created.

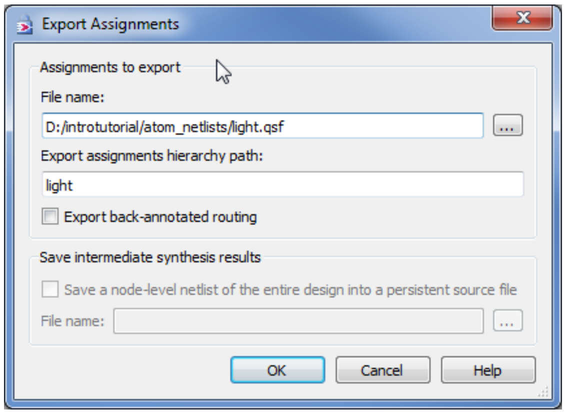

Export Assignments
Assignments to export
File name:
D:/introtutorial/atom_netlists/light.qsf
Export assignments hierarchy path:
light
Export back-annotated routing
Save intermediate synthesis results
Save a node-level netlist of the entire design into a persistent source file
File name:
OK
Cancel
Help

Figure 26. Exporting the pin assignment.

You can import a pin assignment by choosing Assignments > Import Assignments. This opens the dialogue in Figure 27 to select the file to import. Type the name of the file, including the *qsf* extension and the full path to the directory that holds the file, in the File Name box and press OK. Of course, you can also browse to find the desired file.

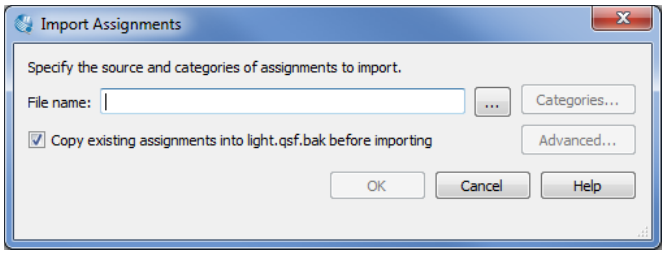

Import Assignments
Specify the source and categories of assignments to import.
File name:
Categories...
Copy existing assignments into light.qsf.bak before importing
Advanced...
OK
Cancel
Help

Figure 27. Importing the pin assignment.

For convenience when using large designs, all relevant pin assignments for the DE-series board are given in individual files. For example, the DE2-115 pin assignments can be found in the *DE2-115.qsf* file, which is available from Altera's University Program website. This file uses the names found in the *DE2-115 User Manual*. If we wanted to make the pin assignments for our example circuit by importing this file, then we would have to use the same names in our Block Diagram/Schematic design file; namely, *SW[0]*, *SW[1]* and *LEDG[0]* for *x1*, *x2* and *f*, respectively. Since these signals are specified in the *DE2-115.qsf* file as elements of vectors *SW* and *LEDG*, we must refer to them in the same way in our design file. For example, in the *DE2-115.qsf* file the 18 toggle switches are called *SW[17]* to *SW[0]*. In a design file they can also be referred to as a vector *SW[17..0]*.

8 Simulating the Designed Circuit

Before implementing the designed circuit in the FPGA chip on the DE-series board, it is prudent to simulate it to ascertain its correctness. Quartus II's Simulation Waveform Editor tool can be used to simulate the behavior of a designed circuit. Before the circuit can be simulated, it is necessary to create the desired waveforms, called *test vectors*, to represent the input signals. It is also necessary to specify which outputs, as well as possible internal points in the circuit, the designer wishes to observe. The simulator applies the test vectors to a model of the implemented circuit and determines the expected response. We will use the Simulation Waveform Editor to draw the test vectors, as follows:

1. In the main Quartus II window, select File > New > Verification/Debugging Files > University Program VWF to open the Simulation Waveform Editor.

2. The Simulation Waveform Editor window is depicted in Figure 28. Save the file under the name *light.vwf*; note that this changes the name in the displayed window. Set the desired simulation to run from 0 to 200 ns by selecting Edit > Set End Time and entering 200 ns in the dialog box that pops up. Selecting View > Fit in Window displays the entire simulation range of 0 to 200 ns in the window, as shown in Figure 29. You may wish to resize the window to its maximum size.

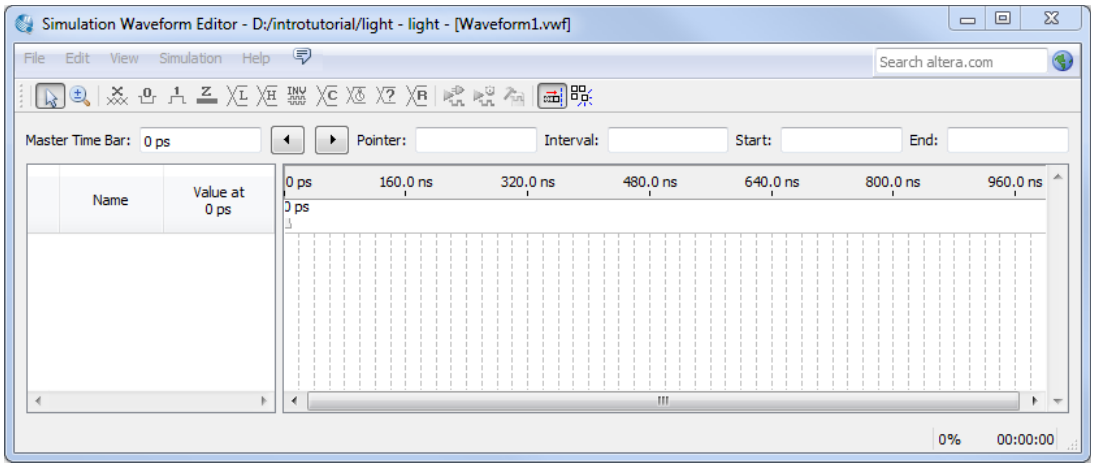


Figure 28. The Waveform Editor window.

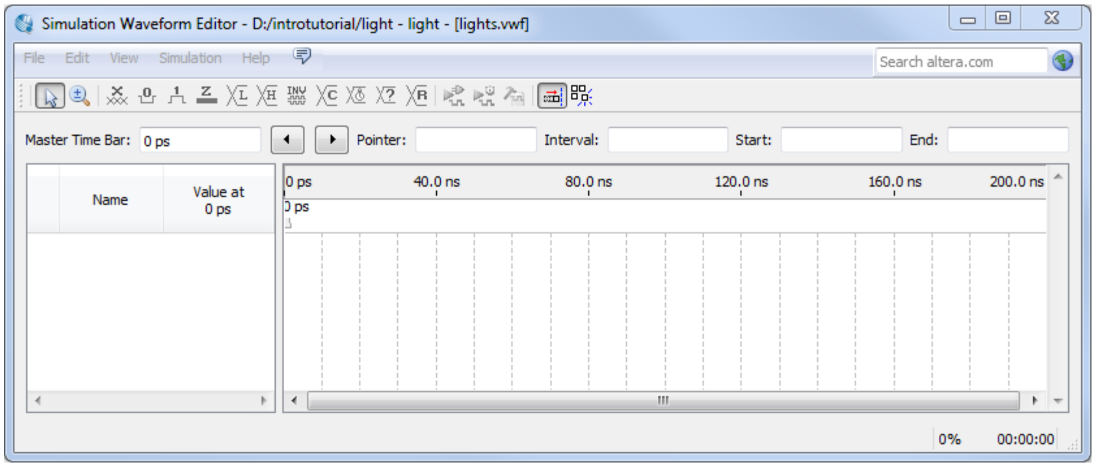
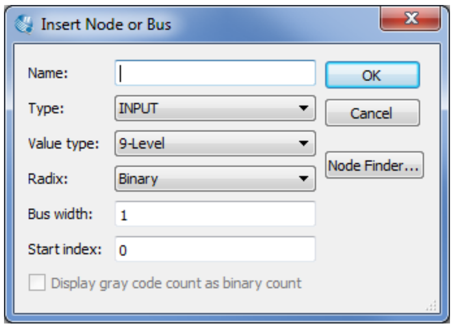

Simulation Waveform Editor - D:/introtutorial/light - light - [lights.vwf]
File Edit View Simulation Help
Search altera.com
Master Time Bar: 0 ps
Pointer:
Interval:
Start:
End:
Name
Value at 0 ps
0 ps 40.0 ns 80.0 ns 120.0 ns 160.0 ns 200.0 ns
0%
00:00:00

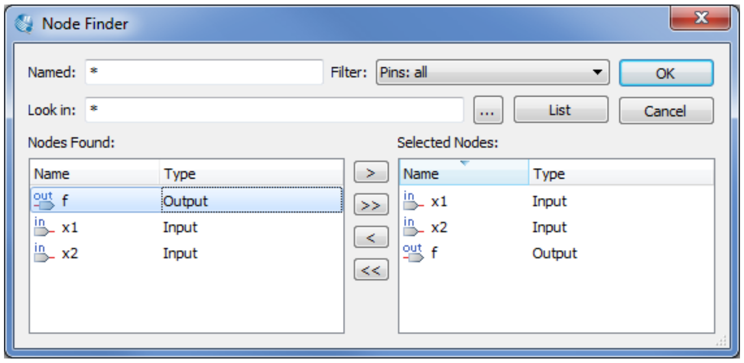
Figure 29. The augmented Waveform Editor window.

3. Next, we want to include the input and output nodes of the circuit to be simulated. Click Edit > Insert > Insert Node or Bus to open the window in Figure 30. It is possible to type the name of a signal (pin) into the Name box, or use the Node Finder to search your project for the signals. Click on the button labeled Node Finder to open the window in Figure 31. The Node Finder utility has a filter used to indicate what type of nodes are to be found. Since we are interested in input and output pins, set the filter to Pins: all. Click the List button to find the input and output nodes as indicated on the left side of the figure.


Insert Node or Bus

Name:
Type: INPUT
Value type: 9-Level
Radix: Binary
Bus width: 1
Start index: 0
Display gray code count as binary count

Figure 30. The Insert Node or Bus dialogue.



Named: * Filter: Pins: all

Look in: *

Nodes Found:

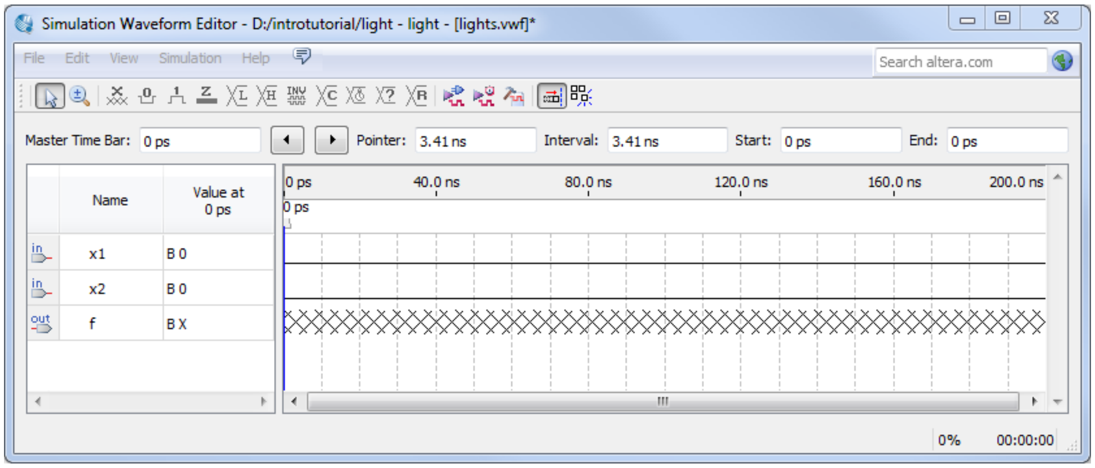
Name	Type
f	Output
x1	Input
x2	Input

Selected Nodes:

Name	Type
x1	Input
x2	Input
f	Output

Figure 31. Selecting nodes to insert into the Waveform Editor.

Click on the *x1* signal in the Nodes Found box in Figure 31, and then click the > sign to add it to the Selected Nodes box on the right side of the figure. Do the same for *x2* and *f*. Click OK to close the Node Finder window, and then click OK in the window of Figure 30. This leaves a fully displayed Waveform Editor window, as shown in Figure 32. If you did not select the nodes in the same order as displayed in Figure 32, it is possible to rearrange them. To move a waveform up or down in the Waveform Editor window, click within the node's row (i.e. on its name, icon, or value) and drag it up or down in the Waveform Editor.



Simulation Waveform Editor - D:/introtutorial/light - light - [lights.vwf]*

Master Time Bar: 0 ps Pointer: 3.41 ns Interval: 3.41 ns Start: 0 ps End: 0 ps

Name	Value at 0 ps
x1	B 0
x2	B 0
f	B X

Figure 32. The nodes needed for simulation.

4. We will now specify the logic values to be used for the input signals *x1* and *x2* during simulation. The logic values at the output *f* will be generated automatically by the simulator. To make it easy to draw the desired waveforms, the Waveform Editor displays (by default) vertical guidelines and provides a drawing feature that snaps on these lines (which can otherwise be invoked by choosing Edit > Snap to Grid). Observe also a solid vertical line, which can be moved by pointing to its top and dragging it horizontally. This reference line is used in analyzing the timing of a circuit; move it to the *time* = 0 position. The waveforms can be drawn using the Selection Tool, which is activated by selecting the icon in the toolbar.

To simulate the behavior of a large circuit, it is necessary to apply a sufficient number of input valuations and observe the expected values of the outputs. In a large circuit the number of possible input valuations may be huge, so in practice we choose a relatively small (but representative) sample of these input valuations. However, for our tiny circuit we can simulate all four input valuations given in Figure 11. We will use four 50-ns time intervals to apply the four test vectors.

We can generate the desired input waveforms as follows. Click on the waveform for the *x1* node. Once a waveform is selected, the editing commands in the Waveform Editor can be used to draw the desired waveforms. Commands are available for setting a selected signal to 0, 1, unknown (X), high impedance (Z), weak low (L), weak high (H), a count value (C), an arbitrary value, a random value (R), inverting its existing value (INV), or defining a clock waveform. Each command can be activated by using the Edit > Value command, or via the toolbar for the Waveform Editor. The Value menu can also be opened by right-clicking on a selected waveform.

Set *x1* to 0 in the time interval 0 to 100 ns, which is probably already set by default. Next, set *x1* to 1 in the time interval 100 to 200 ns. Do this by pressing the mouse at the start of the interval and dragging it to its end, which highlights the selected interval, and choosing the logic value 1 in the toolbar. Make *x2* = 1 from 50 to 100 ns and also from 150 to 200 ns, which corresponds to the truth table in Figure 11. This should produce the image in Figure 33. Observe that the output *f* is displayed as having an unknown value at this time, which is indicated by a hashed pattern; its value will be determined during simulation. Save the file.

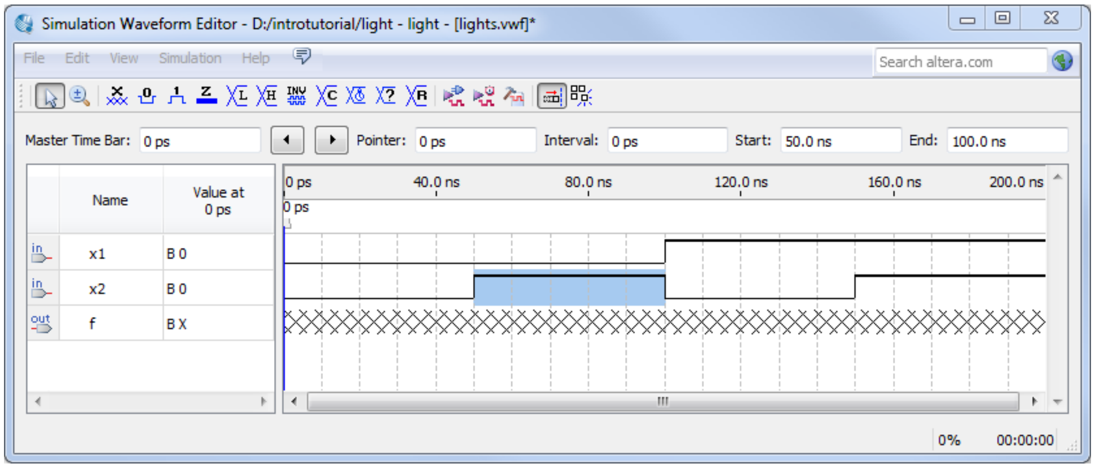


Figure 33. Setting of test values.

8.1 Performing the Simulation

A designed circuit can be simulated in two ways. The simplest way is to assume that logic elements and interconnection wires in the FPGA are perfect, thus causing no delay in propagation of signals through the circuit. This is called *functional simulation*. A more complex alternative is to take all propagation delays into account, which leads to *timing simulation*. Typically, functional simulation is used to verify the functional correctness of a circuit as it is being designed.

8.1.1 Functional Simulation

Before running a functional simulation it is necessary to run Analysis and Synthesis on your design by selecting the icon in the main Quartus II window. Note that Analysis and Synthesis gets run as a part of the main compilation flow. If you compiled your design in Section 6, then it is not necessary to run Analysis and Synthesis again.

To perform the functional simulation, select Simulation > Run Functional Simulation or select the icon. A pop-up window will show the progress of the simulation then automatically close when it is complete. At the end of the simulation, a second Waveform Editor window will open the results of the simulation as illustrated in Figure 34. Observe that the output f is as specified in the truth table of Figure 11.

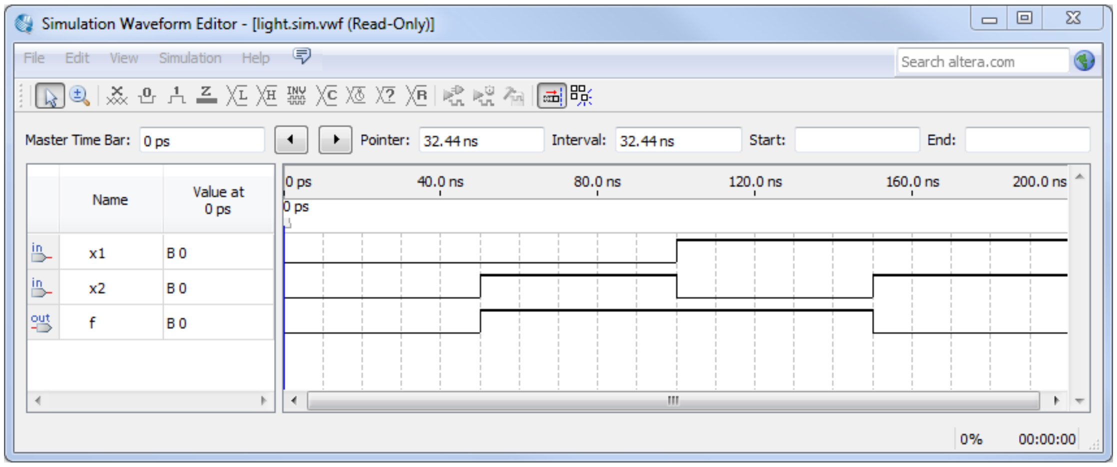


Figure 34. The result of functional simulation.

8.1.2 Timing Simulation

Having ascertained that the designed circuit is functionally correct, we should now perform the timing simulation to see how it will behave when it is actually implemented in the chosen FPGA device. Before running a timing simulation, it is necessary to compile your design by selecting the icon in the main Quartus II window. Unlike functional simulations, timing simulations require the full compilation of your design, not just Analysis and Synthesis.

To perform the timing simulation, select Simulation > Run Timing Simulation or select the icon. The simulation should produce the waveforms in Figure 35. Observe that there is a delay of about 5 ns in producing a change in the signal f from the time when the input signals, x_1 and x_2, change their values. This delay is due to the propagation delays in the logic element and the wires in the FPGA device.

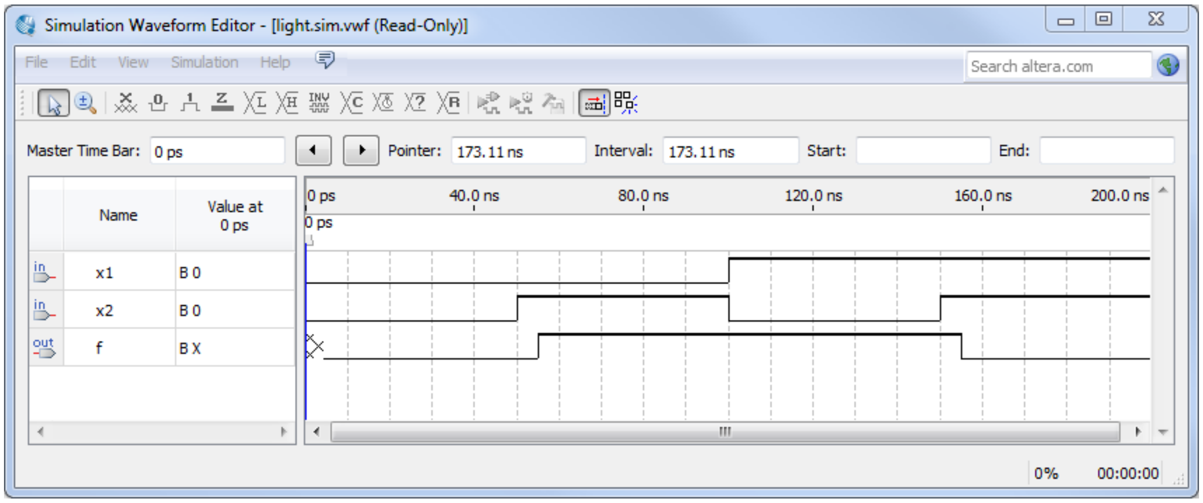


Figure 35. The result of timing simulation.

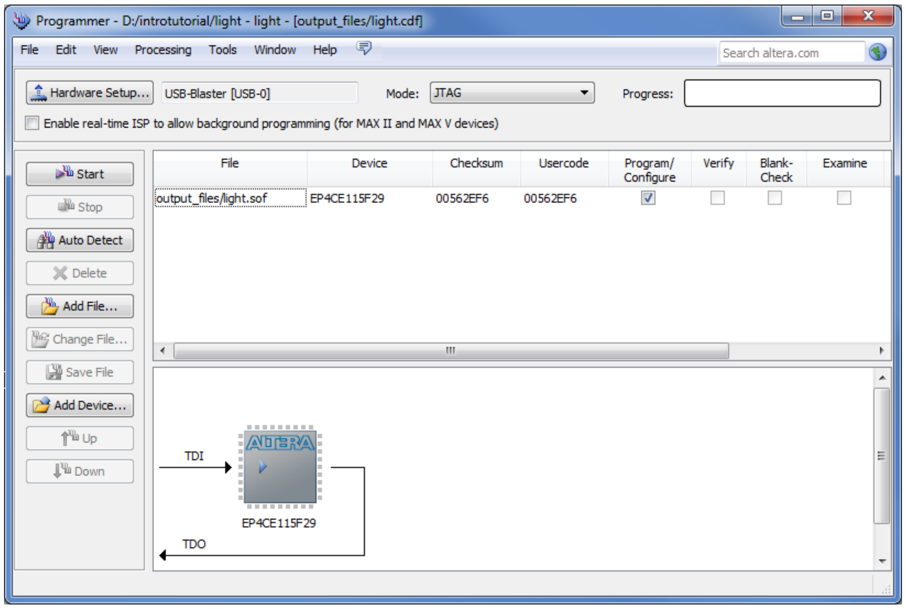
9 Programming and Configuring the FPGA Device

The FPGA device must be programmed and configured to implement the designed circuit. The required configuration file is generated by the Quartus II Compiler's Assembler module. Altera's DE-series board allows the configuration to be done in two different ways, known as JTAG and AS modes. The configuration data is transferred from the host computer (which runs the Quartus II software) to the board by means of a cable that connects a USB port on the host computer to the leftmost USB connector on the board. To use this connection, it is necessary to have the USB-Blaster driver installed. If this driver is not already installed, consult the tutorial *Getting Started with Altera's DE-Series Boards* for information about installing the driver. Before using the board, make sure that the USB cable is properly connected and turn on the power supply switch on the board.

In the JTAG mode, the configuration data is loaded directly into the FPGA device. The acronym JTAG stands for Joint Test Action Group. This group defined a simple way for testing digital circuits and loading data into them, which became an IEEE standard. If the FPGA is configured in this manner, it will retain its configuration as long as the power remains turned on. The configuration information is lost when the power is turned off. The second possibility is to use the Active Serial (AS) mode. In this case, a configuration device that includes some flash memory is used to store the configuration data. Quartus II software places the configuration data into the configuration device on the DE-series board. Then, this data is loaded into the FPGA upon power-up or reconfiguration. Thus, the FPGA need not be configured by the Quartus II software if the power is turned off and on. The choice between the two modes is made by the RUN/PROG switch on the DE-series board. The RUN position selects the JTAG mode, while the PROG position selects the AS mode. This tutorial discusses only the JTAG programming mode.

9.1 JTAG Programming for the DE0-Nano and DE2-115 Boards

For the DE0-Nano and DE2-115 Boards, the programming and configuration task is performed as follows. If using the DE1-SoC board, then the instructions in the following section should be followed. To program the FPGA chip, the RUN/PROG switch on the board must be in the RUN position. Select Tools > Programmer to reach the window in Figure 36. Here it is necessary to specify the programming hardware and the mode that should be used. If not already chosen by default, select JTAG in the Mode box. Also, if the USB-Blaster is not chosen by default, press the Hardware Setup... button and select the USB-Blaster in the window that pops up, as shown in Figure 37.



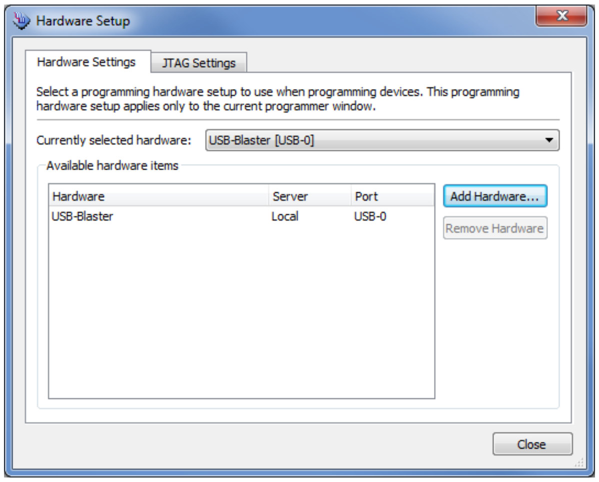
Hardware Setup... USB-Blaster [USB-0]   Mode: JTAG   Progress:

Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine
output_files/light.sof	EP4CE115F29	00562EF6	00562EF6	✓			

Figure 36. The Programmer window.

Observe that the configuration file *light.sof* is listed in the window in Figure 36. If the file is not already listed, then click Add File and select it. This is a binary file produced by the Compiler's Assembler module, which contains the data needed to configure the FPGA device. The extension *.sof* stands for SRAM Object File. Note also that the device selected is EP4CE115F29, which is the FPGA device used on the DE2-115 board. Click on the Program/Configure check box, as shown in Figure 38.



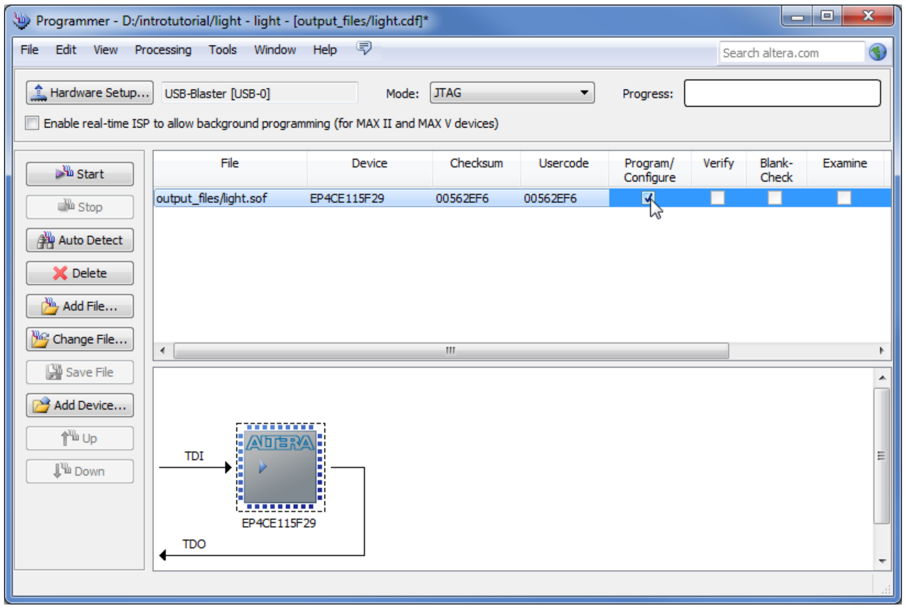
Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: USB-Blaster [USB-0]

Available hardware items

Hardware	Server	Port
USB-Blaster	Local	USB-0

Figure 37. The Hardware Setup window.



USB-Blaster [USB-0] Mode: JTAG

Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

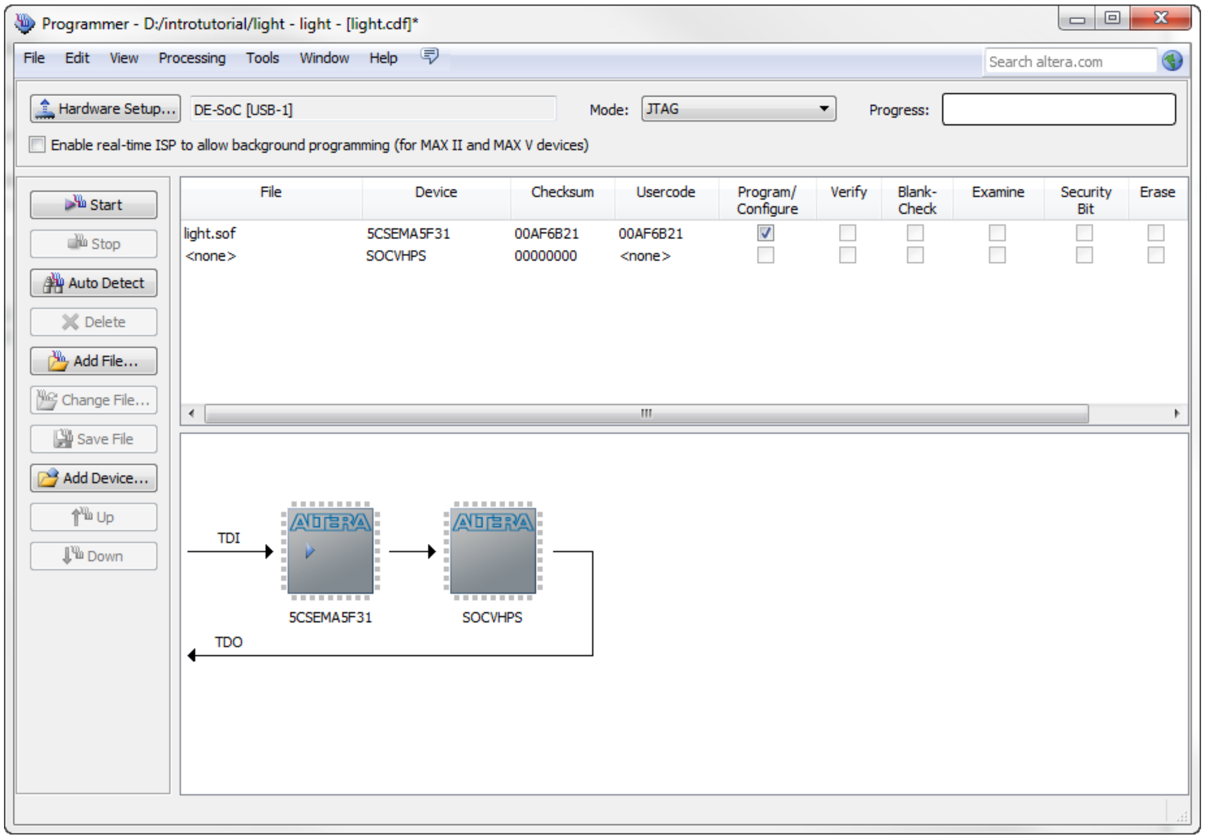
File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank-Check	Examine
output_files/light.sof	EP4CE115F29	00562EF6	00562EF6	✓			

Figure 38. The updated Programmer window.

Now, press Start in the window in Figure 38. An LED on the board will light up when the configuration data has been downloaded successfully. If you see an error reported by Quartus II software indicating that programming failed, then check to ensure that the board is properly powered on.

9.2 JTAG Programming for the DE1-SoC Board

For the DE1-SoC board, the following steps should be used for programming. Select Tools > Programmer to reach the window in Figure 39. Here it is necessary to specify the programming hardware and the mode that should be used. If not already chosen by default, select JTAG in the Mode box. Also, if *DE-SoC* is not chosen by default as the programming hardware, then press the Hardware Setup... button and select the *DE-SoC* in the window that pops up, as shown in Figure 40.



Programmer - D:/introtutorial/light - light - [light.cdf]*

Hardware Setup... DE-SoC [USB-1]

Mode: JTAG

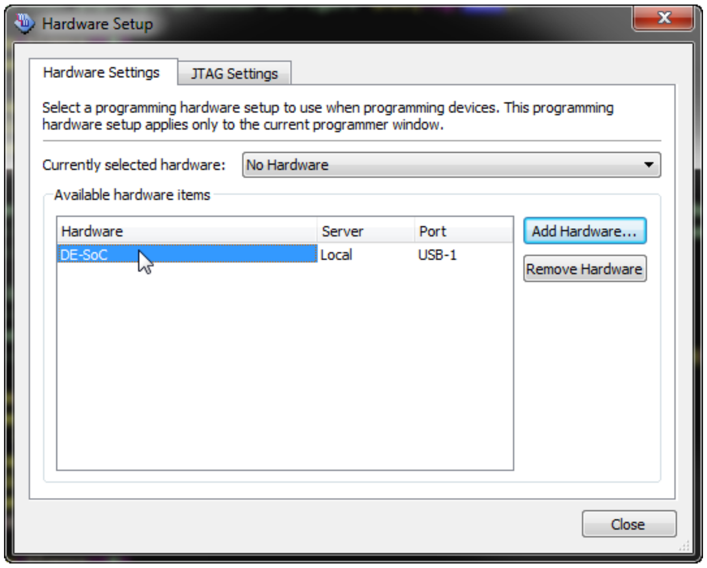
Progress:

☐ Enable real-time ISP to allow background programming (for MAX II and MAX V devices)

File	Device	Checksum	Usercode	Program/ Configure	Verify	Blank- Check	Examine	Security Bit	Erase
light.sof	5CSEMA5F31	00AF6B21	00AF6B21	☑	☐	☐	☐	☐	☐
<none>	SOCVHPS	00000000	<none>	☐	☐	☐	☐	☐	☐

Figure 39. The Programmer window.

Observe that the configuration file *light.sof* is listed in the window in Figure 39. If the file is not already listed, then click Add File and select it. This is a binary file produced by the Compiler's Assembler module, which contains the data needed to configure the FPGA device. The extension *.sof* stands for SRAM Object File. Click on the Program/Configure check box, as shown in Figure 39. This setting is used to select the FPGA in the Cyclone V SoC chip for programming.


Hardware Setup

Hardware Settings | JTAG Settings

Select a programming hardware setup to use when programming devices. This programming hardware setup applies only to the current programmer window.

Currently selected hardware: No Hardware

Available hardware items

Hardware	Server	Port
DE-SoC	Local	USB-1

Figure 40. The Hardware Setup window.

Now, press Start in the Programmer. An LED on the DE1-SoC board will light up while the FPGA device is being programmed. If you see an error reported by Quartus II software indicating that programming failed, then check to ensure that the board is properly powered on.

10 Testing the Designed Circuit

Having downloaded the configuration data into the FPGA device, you can now test the implemented circuit. Flip the RUN/PROG switch to RUN position. Try all four valuations of the input variables x_1 and x_2, by setting the corresponding states of the switches SW_1 and SW_0. Verify that the circuit implements the truth table in Figure 11.

If you want to make changes in the designed circuit, first close the Programmer window. Then make the desired changes in the VHDL design file, compile the circuit, and program the board as explained above.