



# Quartus® Prime Introduction Using Verilog Designs

*For Quartus® Prime 17.0*

## Contents

|           |   |           |
|-----------|---|-----------|
| <b>1</b>  | <b>Introduction</b>   | <b>2</b>  |
| <b>2</b>  | <b>Background</b>   | <b>3</b>  |
| <b>3</b>  | <b>Getting Started</b>  | <b>4</b>  |
| 3.1       | Quartus® Prime Online Help . . . . .  | 6         |
| <b>4</b>  | <b>Starting a New Project</b>   | <b>6</b>  |
| <b>5</b>  | <b>Design Entry Using Verilog Code</b>  | <b>13</b> |
| 5.1       | Using the Quartus® Prime Text Editor . . . . .  | 13        |
| 5.1.1     | Using Verilog Templates . . . . .   | 16        |
| 5.2       | Adding Design Files to a Project . . . . .  | 16        |
| <b>6</b>  | <b>Compiling the Designed Circuit</b>   | <b>18</b> |
| 6.1       | Errors . . . . .  | 19        |
| <b>7</b>  | <b>Pin Assignment</b>   | <b>22</b> |
| <b>8</b>  | <b>Programming and Configuring the FPGA Device</b>  | <b>26</b> |
| 8.1       | JTAG* Programming for the DE0-CV, DE0-Nano, DE10-Lite, and DE2-115 Boards . . . . .           | 26        |
| 8.2       | JTAG* Programming for the DE0-Nano-SoC, DE1-SoC Board, DE10-Nano, and DE10-Standard . . . . . | 28        |
| <b>9</b>  | <b>Simulating the Designed Circuit</b>  | <b>29</b> |
| 9.1       | Performing the Simulation . . . . .   | 33        |
| 9.1.1     | Functional Simulation . . . . .   | 33        |
| 9.1.2     | Timing Simulation . . . . .   | 34        |
| <b>10</b> | <b>Testing the Designed Circuit</b>   | <b>35</b> |

## 1 Introduction

This tutorial presents an introduction to the Quartus® Prime 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 Prime software. The design process is illustrated by giving step-by-step instructions for using the Quartus Prime software to implement a very simple circuit in an Intel® FPGA device.

The Quartus Prime 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 Verilog design entry method, in which the user specifies the desired circuit in the Verilog hardware description language. Three versions of this tutorial are available; one uses the Verilog hardware description language, another uses the VHDL hardware description language, and the third 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 Intel DE-series Development and Education board connected to a computer that has Quartus Prime 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 Prime version 17.0 Standard Edition; other versions of the software may be slightly different.

## 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.

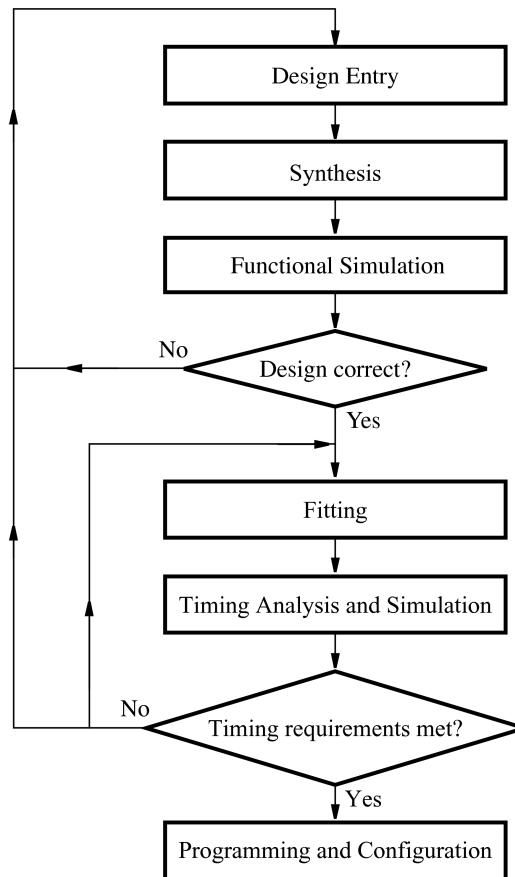


Figure 1. Typical CAD flow.

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

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

### 3 Getting Started

Each logic circuit, or subcircuit, being designed with Quartus Prime 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 Prime 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 Prime software, which the user selects with the computer mouse. Most of the commands provided by Quartus Prime 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 Prime 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.

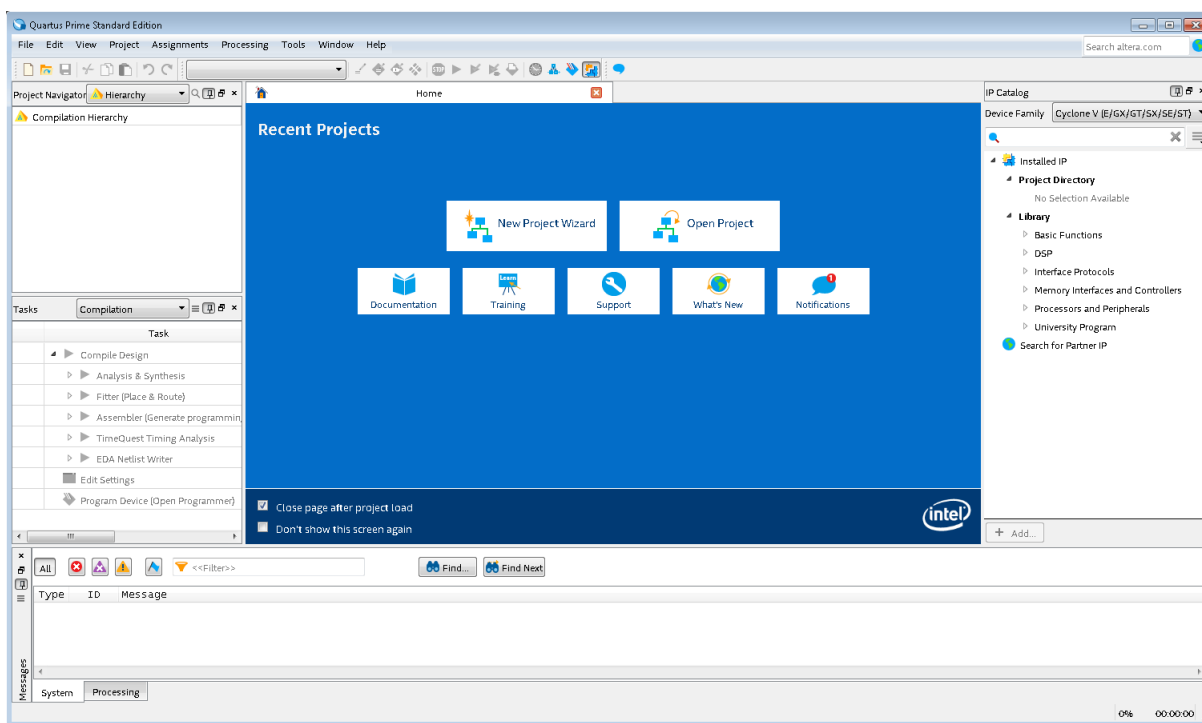


Figure 2. The main Quartus Prime display.

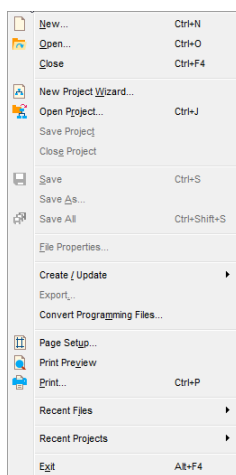


Figure 3. An example of the File menu.

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 the command name will be shown in the status bar at the bottom of the screen.

### 3.1 Quartus® Prime Online Help

Quartus Prime 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 using the search box in the top right corner of the main Quartus display. 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 Prime 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.
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 Prime 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.

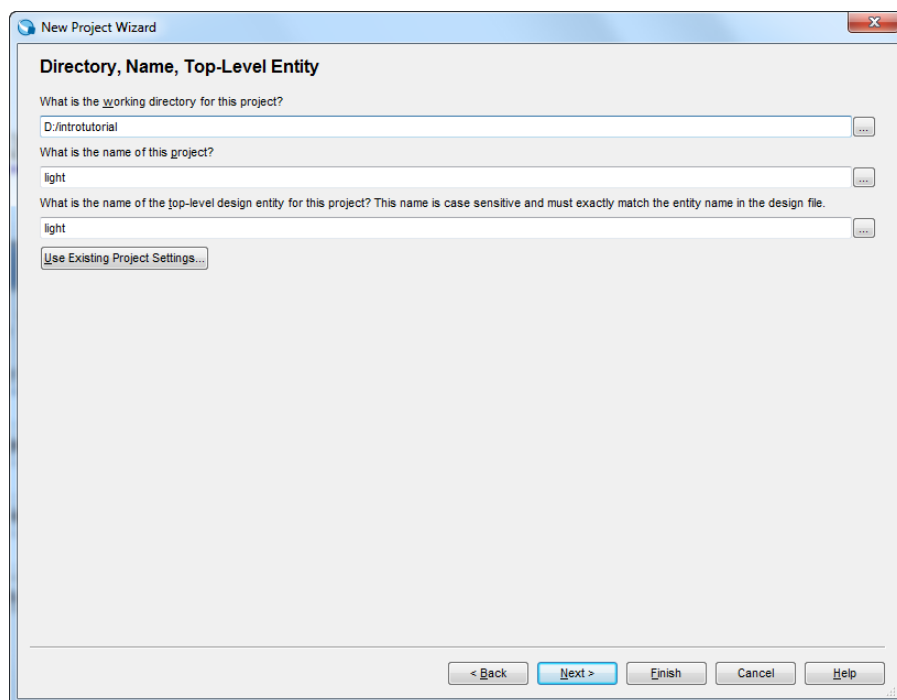


Figure 4. Creation of a new project.



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

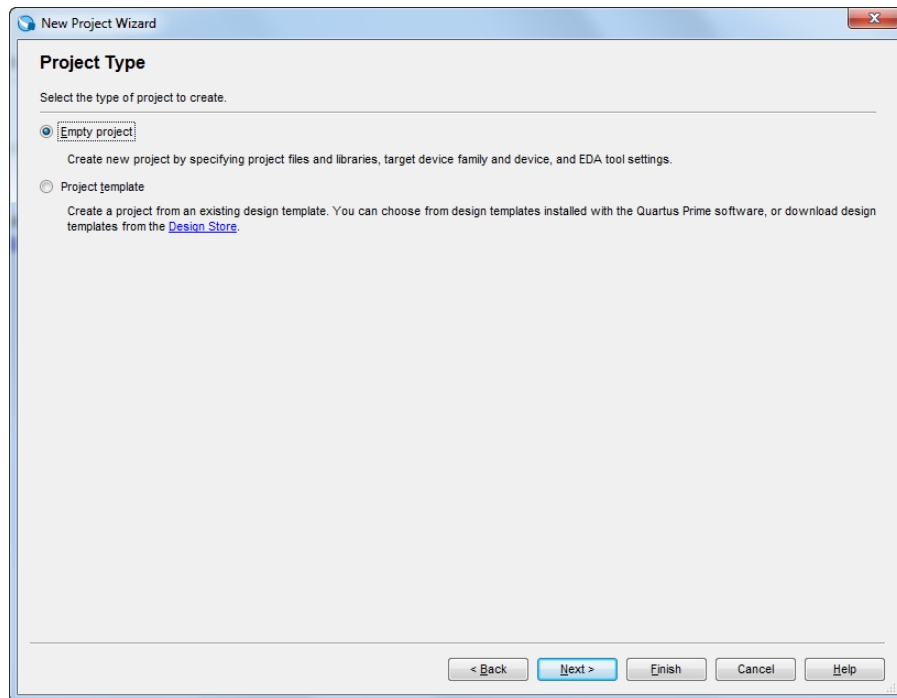


Figure 6. Choosing the project type.

3. The Project Type window, shown in Figure 6, allows you to choose from the Empty project and the Project template options. For this tutorial, choose Empty project as we will be creating a project from scratch, and press Next which leads to the window in Figure 7.



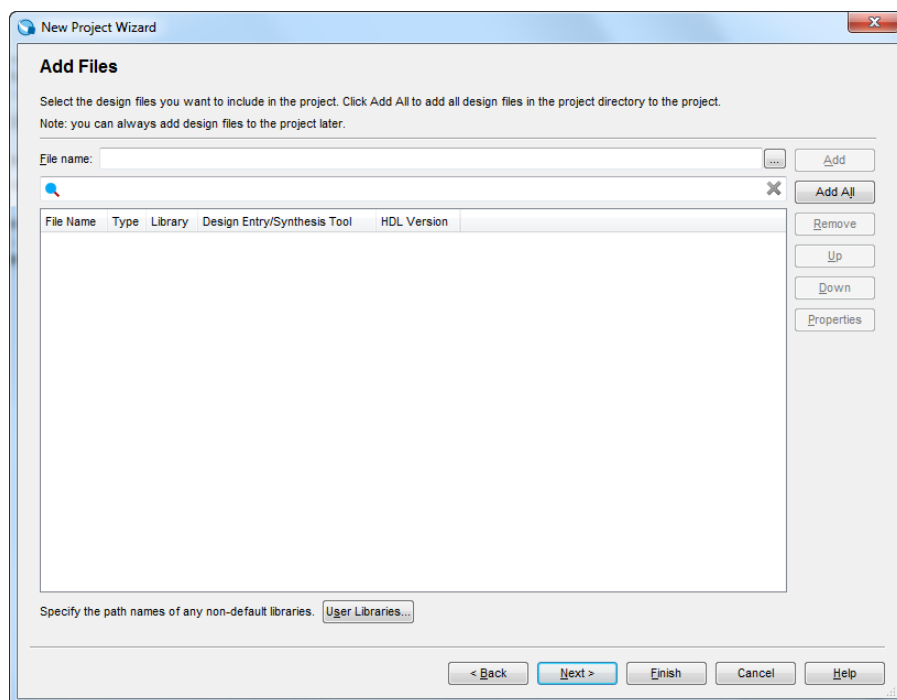


Figure 7. The wizard can include user-specified design files.

4. 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 8.

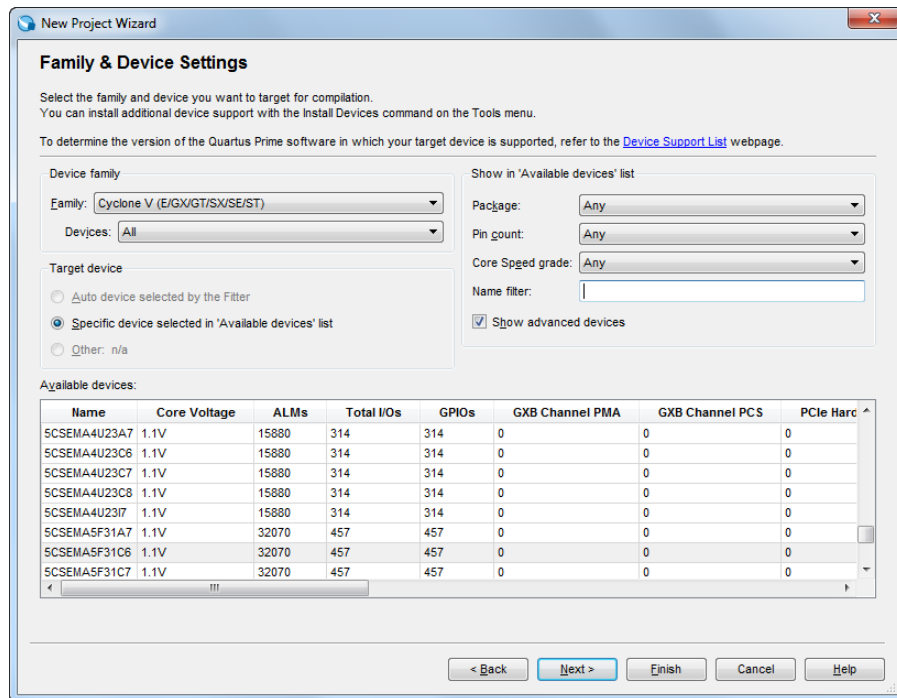


Figure 8. Choose the device family and a specific device.

- 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 Prime 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 9.

| Board         | Device Name                   |
|---------------|-------------------------------|
| DE0-CV        | Cyclone® V 5CEBA4F23C7        |
| DE0-Nano      | Cyclone® IVE EP4CE22F17C6     |
| DE0-Nano-SoC  | Cyclone® V SoC 5CSEMA4U23C6   |
| DE1-SoC       | Cyclone® V SoC 5CSEMA5F31C6   |
| DE2-115       | Cyclone® IVE EP4CE115F29C7    |
| DE10-Lite     | Max® 10 10M50DAF484C7G        |
| DE10-Standard | Cyclone® V SoC 5CSXFC6D6F31C6 |
| DE10-Nano     | Cyclone® V SE 5CSEBA6U2317    |

Table 1. DE-series FPGA device names

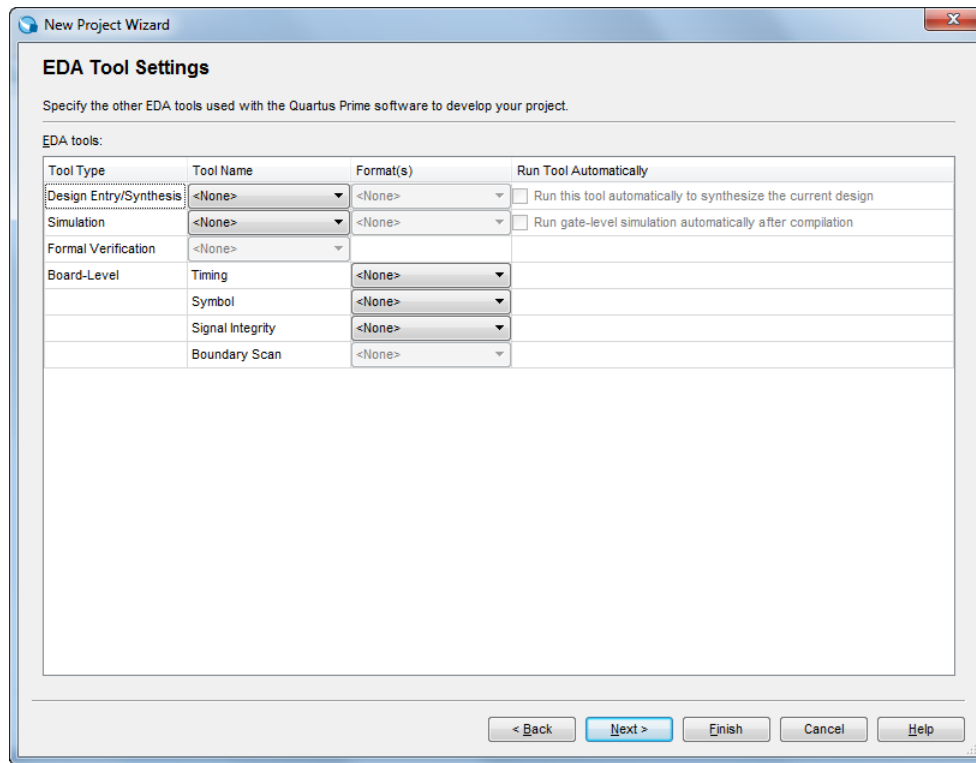


Figure 9. Other EDA tools can be specified.

6. 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 Prime messages that refer to third-party tools, which are the tools developed and marketed by companies other than Intel. Since we will rely solely on Quartus Prime tools, we will not choose any other tools. Press Next.
7. A summary of the chosen settings appears in the screen shown in Figure 10. Press Finish, which returns to the main Quartus Prime window, but with *light* specified as the new project, in the title bar, as indicated in Figure 11.

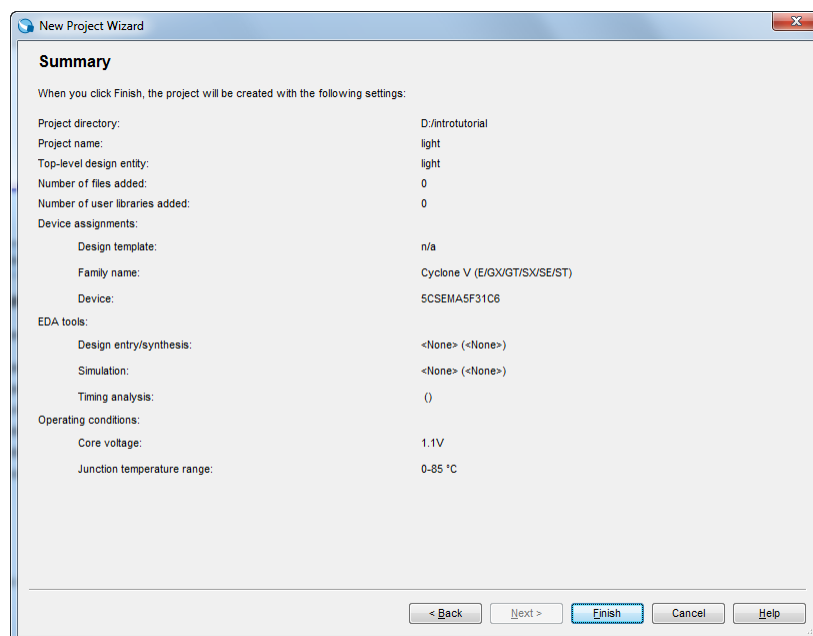


Figure 10. Summary of project settings.

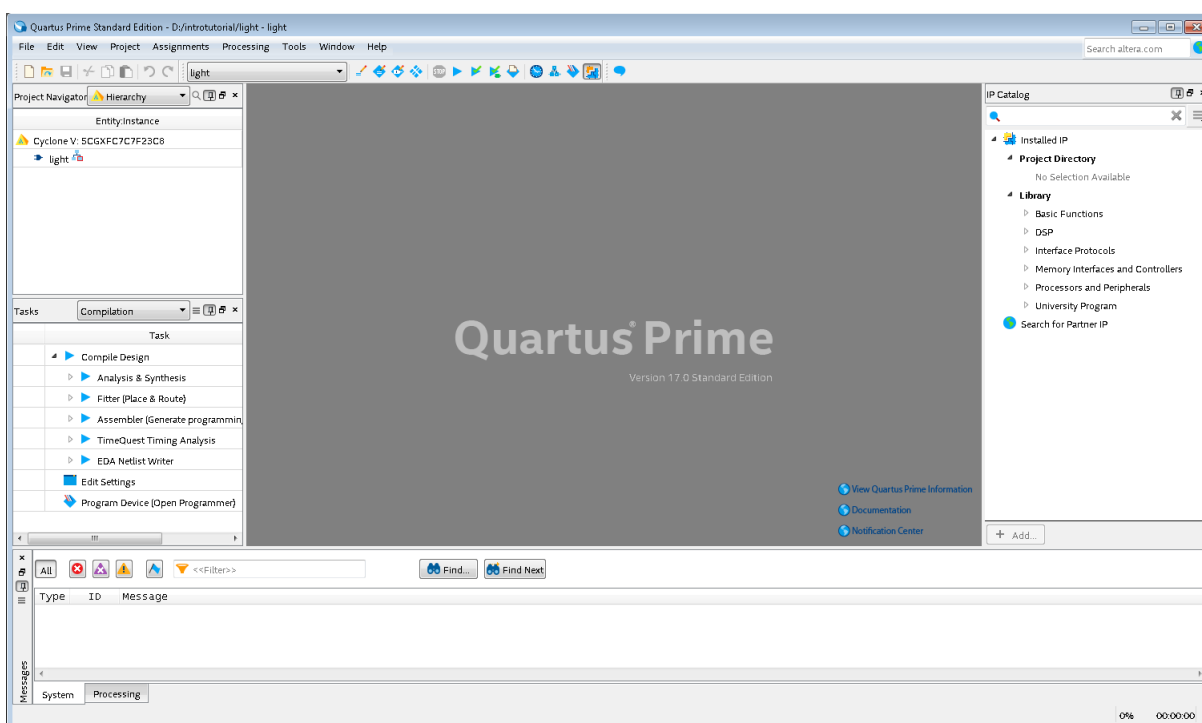


Figure 11. The Quartus Prime window for a created project.

## 5 Design Entry Using Verilog Code

As a design example, we will use the two-way light controller circuit shown in Figure 12. 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.

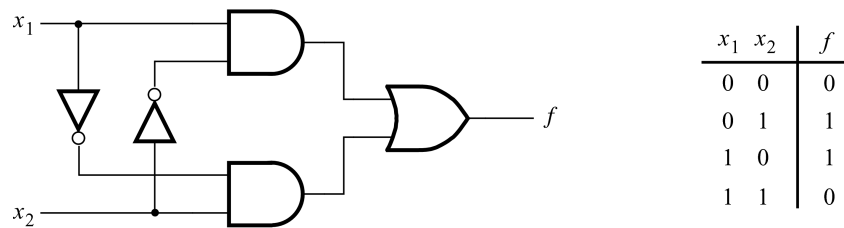


Figure 12. The light controller circuit.

The required circuit is described by the Verilog code in Figure 13. Note that the Verilog module 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 Prime 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 Verilog module. The file name must include the extension *v*, which indicates a Verilog file. So, we will use the name *light.v*.

```

module light (x1, x2, f);
    input x1, x2;
    output f;
    assign f = (x1 & ~x2) | (~x1 & x2);
endmodule

```

Figure 13. Verilog code for the circuit in Figure 11.

### 5.1 Using the Quartus® Prime Text Editor

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

Select **File > New** to get the window in Figure 14, choose **Verilog HDL 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 15. In the box labeled **Save as type** choose **Verilog HDL 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 16. Enter the Verilog code in Figure 13 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 Verilog code. First, the editor can display different types of Verilog 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.

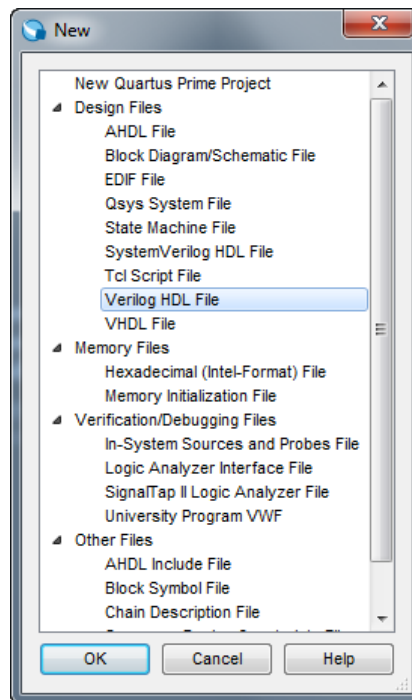


Figure 14. Choose to prepare a Verilog file.

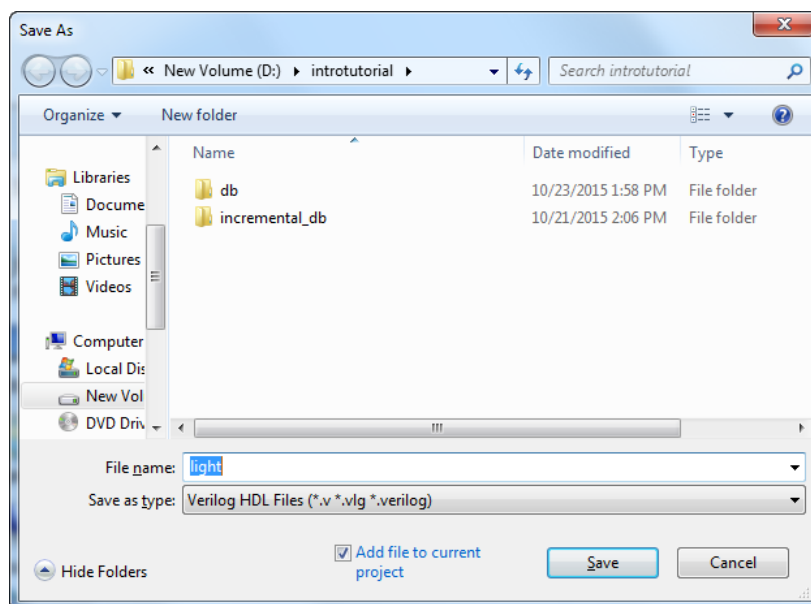


Figure 15. Name the file.

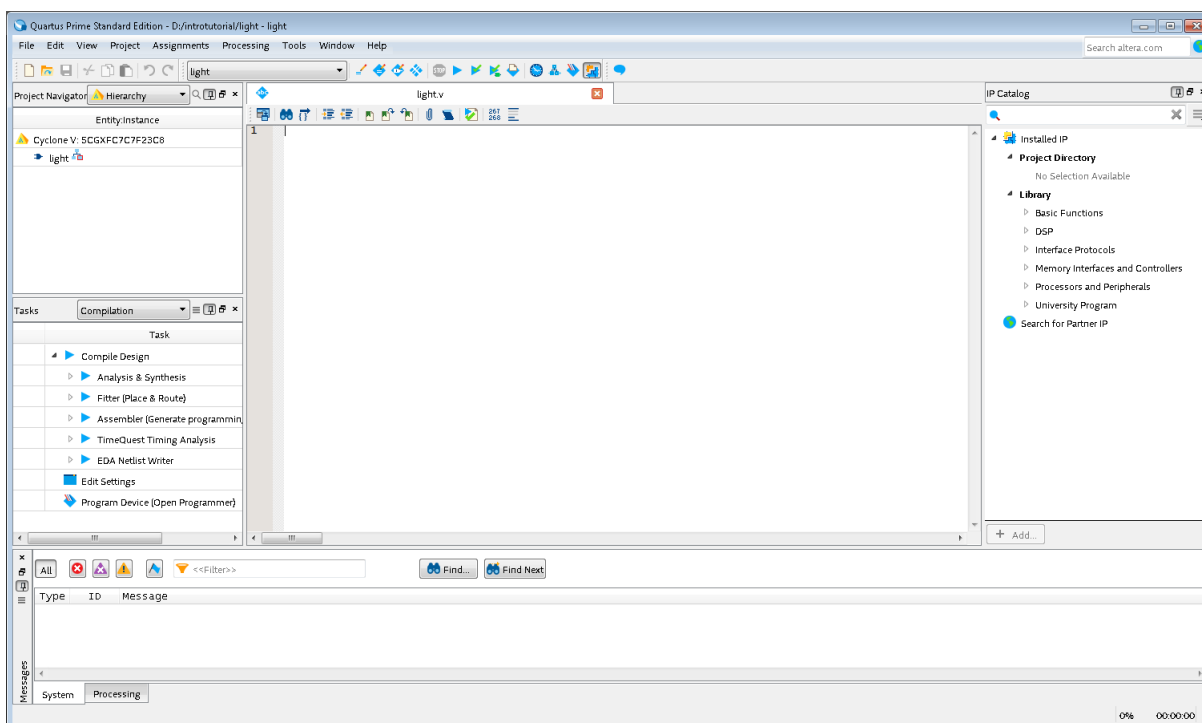


Figure 16. Text Editor window.

### 5.1.1 Using Verilog Templates

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

## 5.2 Adding Design Files to a Project

As we indicated when discussing Figure 7, you can tell Quartus Prime 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 17. 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 Prime 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.v* file is already a part of the project and will be listed in the window in Figure 17. Otherwise, the file must be added to the project. So, if you did not use the Quartus Prime Text Editor, then place a copy of the file *light.v*, 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 labeled **File name** in Figure 17 to get the pop-up window in Figure 18. Select the *light.v* file and click **Open**. The selected file is now indicated in the **File name** box in Figure 17. Click **Add** then **OK** to include the *light.v* file in the project. We should mention that in many cases the Quartus Prime software is able to automatically find the right files to use for each entity referenced in Verilog 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.



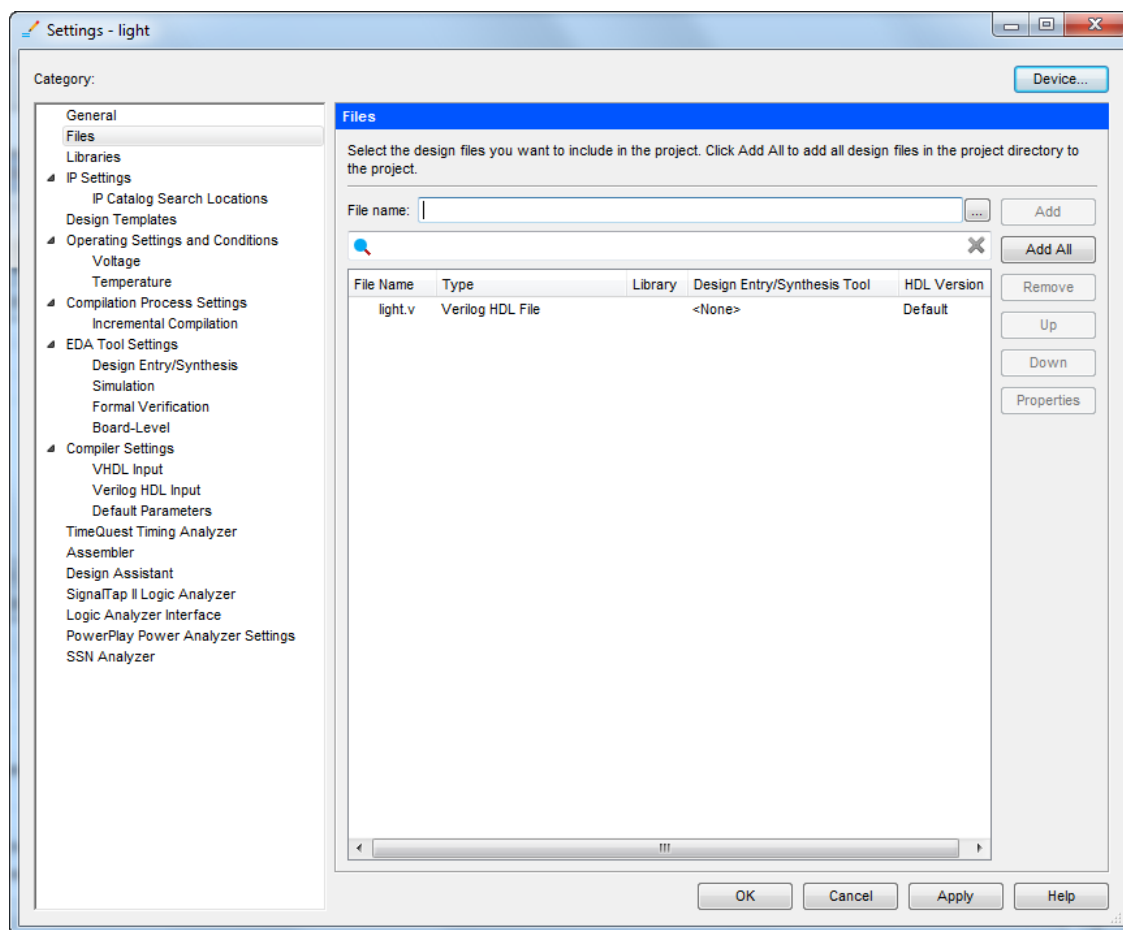


Figure 17. Settings window.

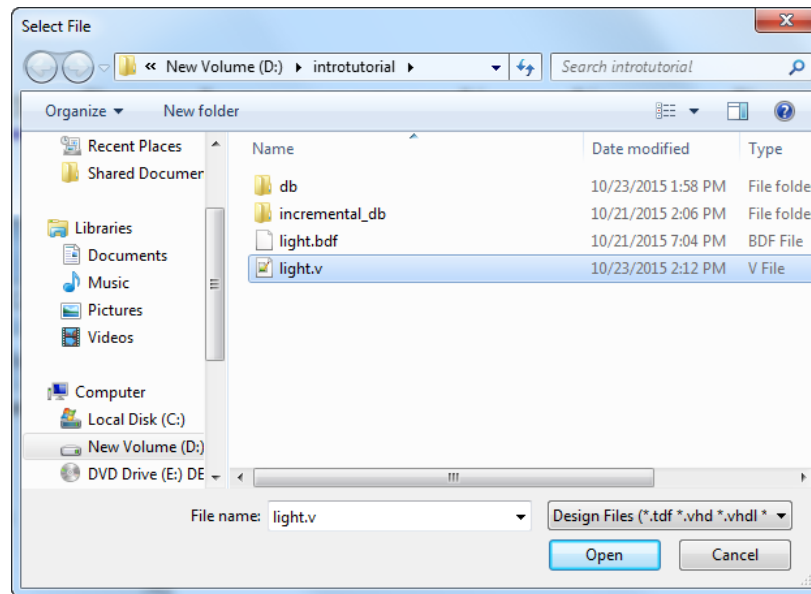




Figure 18. Select the file.

## 6 Compiling the Designed Circuit

The Verilog code in the file *light.v* is processed by several Quartus Prime 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 blue 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 Prime display. In the message window, at the bottom of the figure, various messages are displayed throughout the compilation process. In case of errors, there will be appropriate messages given.

When the compilation is finished, a compilation report is produced. A tab showing this report is opened automatically, as seen in Figure 21. 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 21 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.

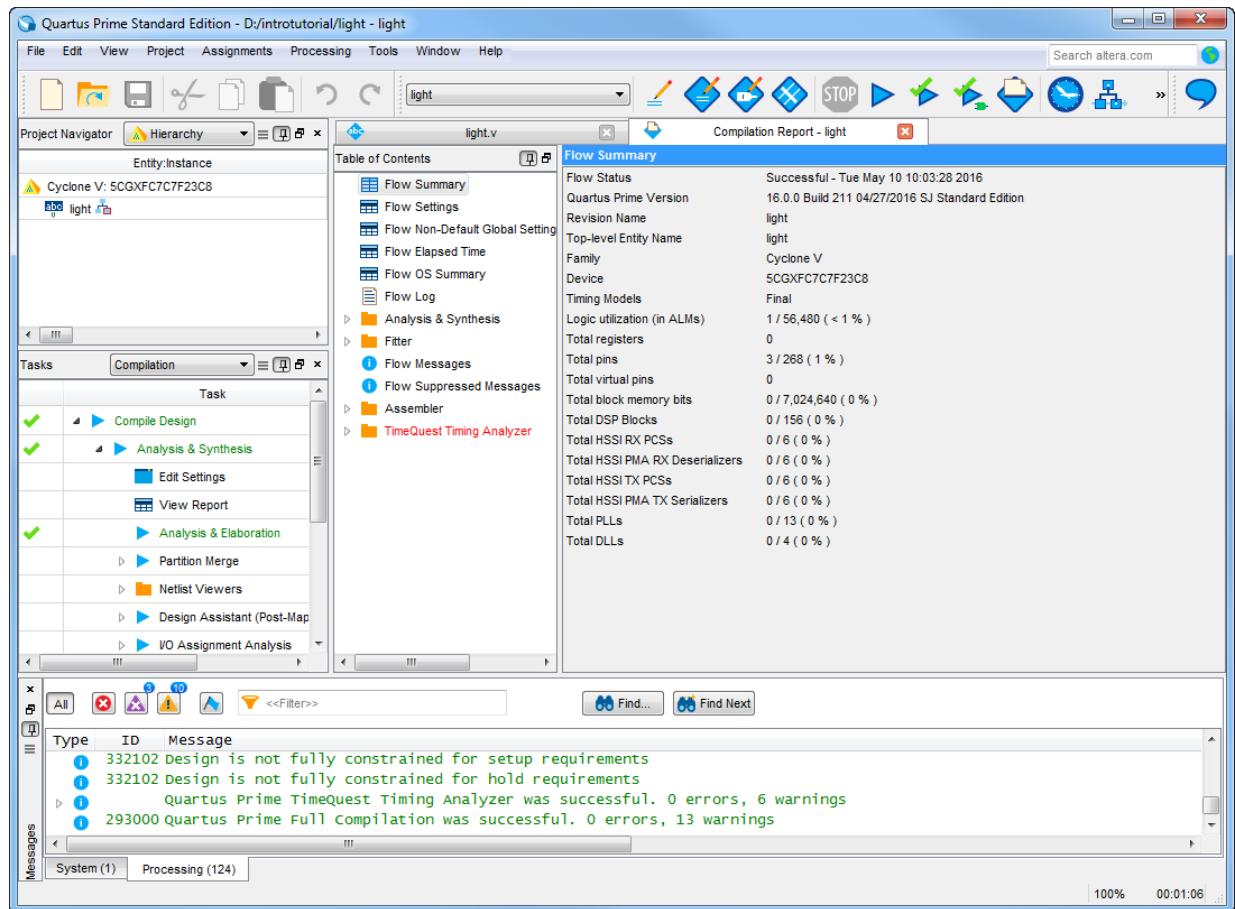



Figure 19. Display after a successful compilation.

## 6.1 Errors

Quartus Prime software displays messages produced during compilation in the Messages window. If the Verilog 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 Verilog 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 Verilog 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.v*. Remove the semicolon in the **assign** statement, 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.v* file should be saved; click **Yes**. After trying to compile the circuit, Quartus Prime software will display error messages in the Messages window, and show that the compilation failed

in the Analysis & Synthesis stage of the compilation process. The compilation report summary, given in Figure 20, 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 21. The Compilation Report can be displayed as a separate window as in Figure 21 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 Prime software responds by opening the *light.v* file and highlighting the statement which is affected by the error, as shown in Figure 22. Correct the error and recompile the design.

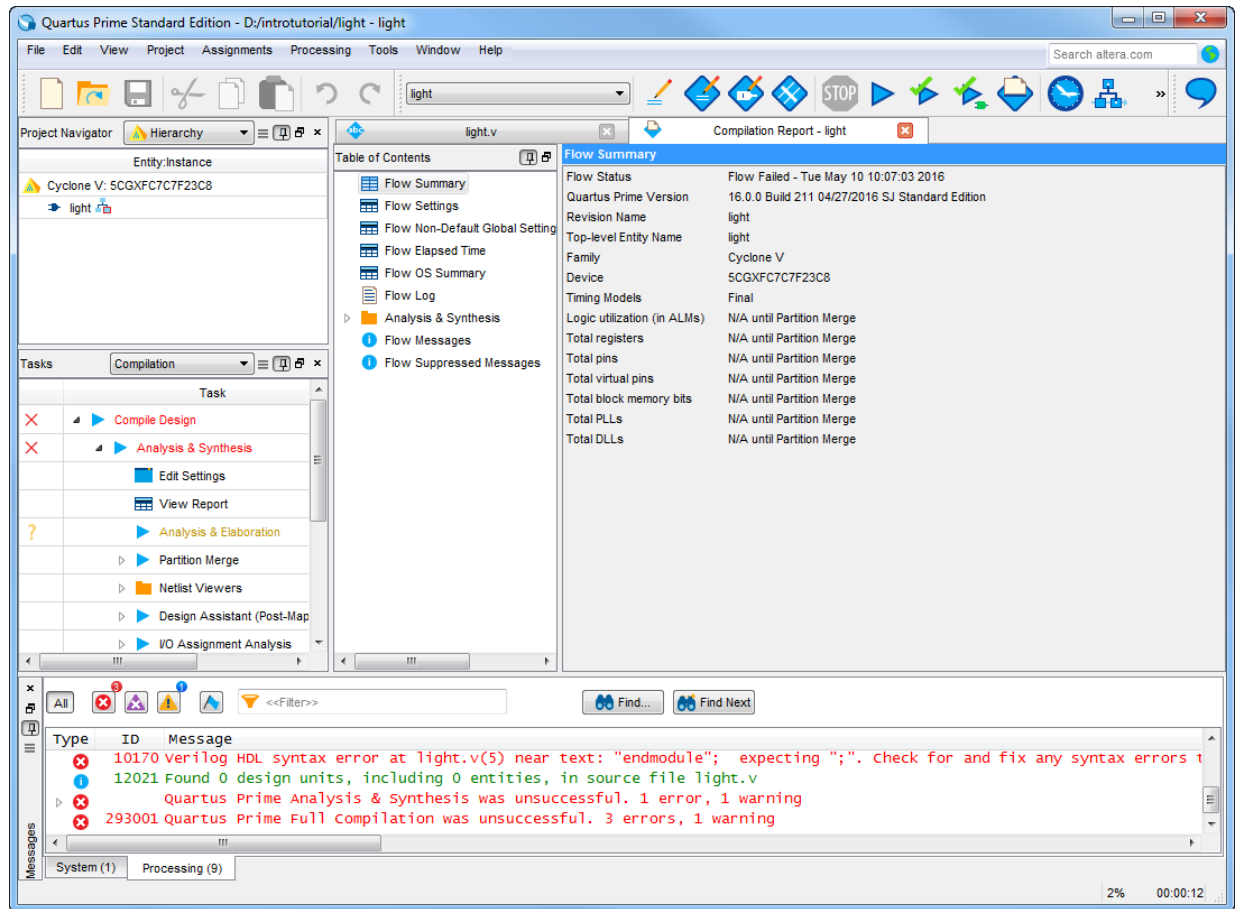


Figure 20. Compilation report for the failed design.

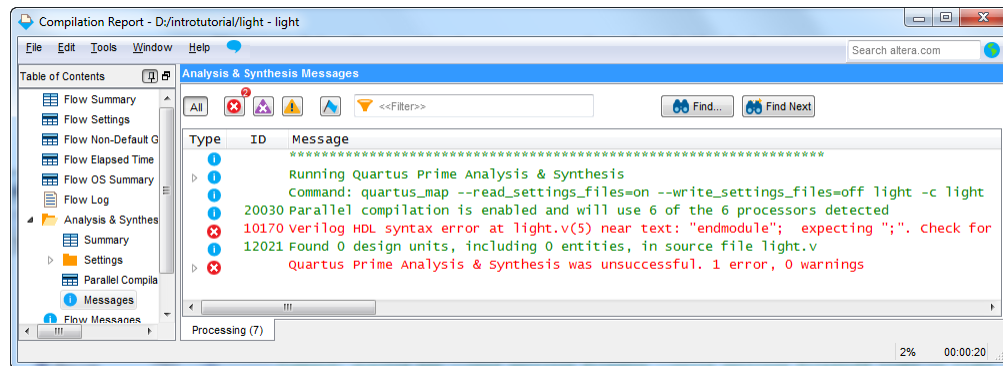


Figure 21. Error messages.

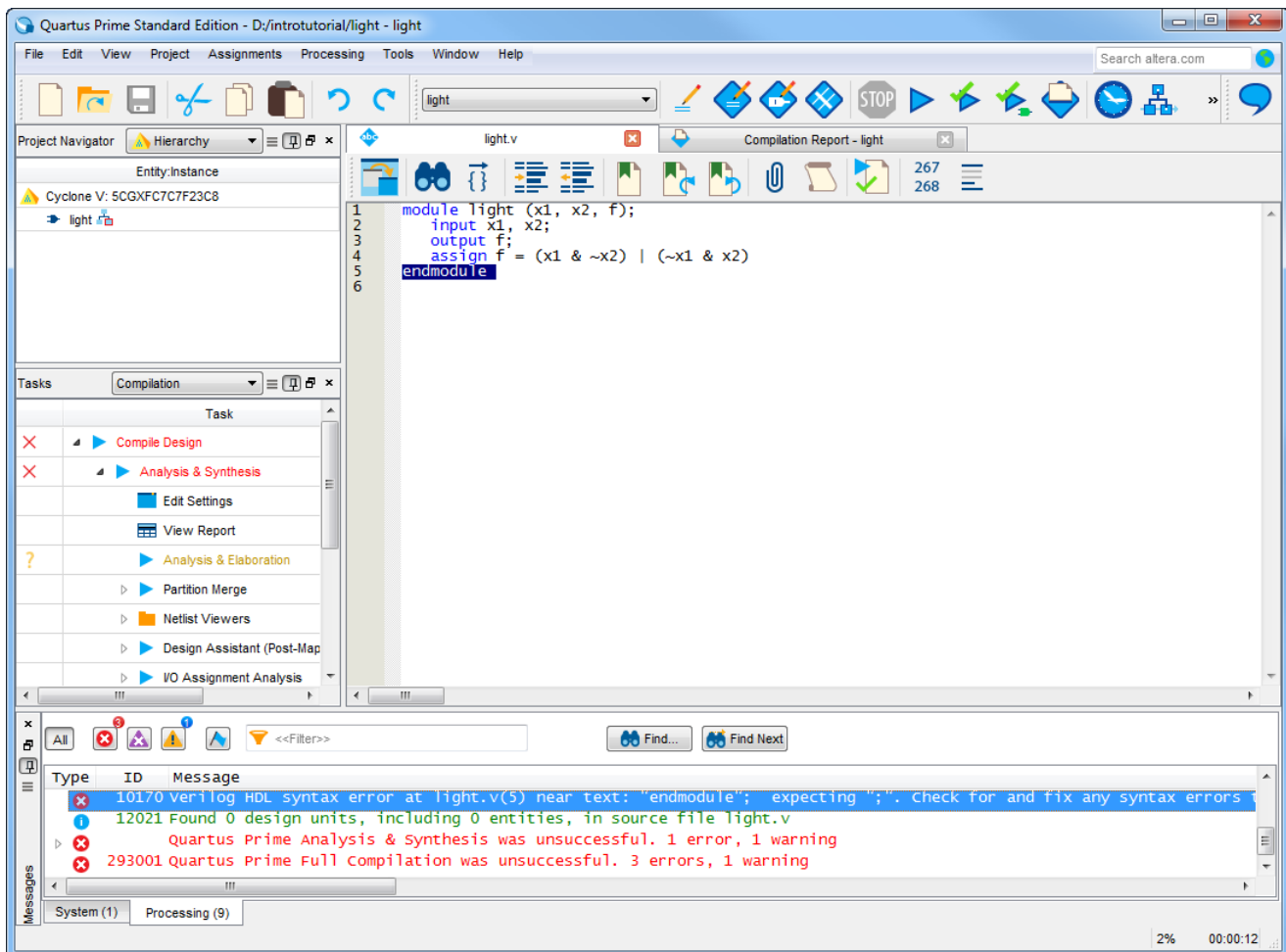


Figure 22. Identifying the location of the error.

## 7 Pin Assignment

During the compilation above, the Quartus Prime 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 a light-emitting diode on your DE-series board. For the DE2-115 we will use a green LED:  $LEDG_0$ . On the DE0-CV, DE1-SoC, DE-10 Lite and DE10-Standard we will use  $LEDR_0$ . On the DE0-Nano and DE0-Nano-SoC, we will use  $LED_0$ . The FPGA pin assignment for the LEDs can also be found in Table 2.

| Component     | $SW_0$   | $SW_1$   | $LEDG_0$ , $LED_0$ , or $LEDR_0$ |
|---------------|----------|----------|----------------------------------|
| DE0-CV        | PIN_U13  | PIN_V13  | PIN_AA2                          |
| DE0-Nano      | PIN_M1   | PIN_T8   | PIN_A1                           |
| DE-Nano-SoC   | PIN_L10  | PIN_L9   | PIN_W15                          |
| DE2-115       | PIN_AB28 | PIN_AC28 | PIN_E21                          |
| DE1-SoC       | PIN_AB12 | PIN_AC12 | PIN_V16                          |
| DE10-Lite     | PIN_C10  | PIN_C11  | PIN_A8                           |
| DE10-Standard | PIN_AB30 | PIN_AB28 | PIN_AA24                         |
| DE10-Nano     | PIN_Y24  | PIN_W24  | PIN_W15                          |

Table 2. DE-Series Pin Assignments

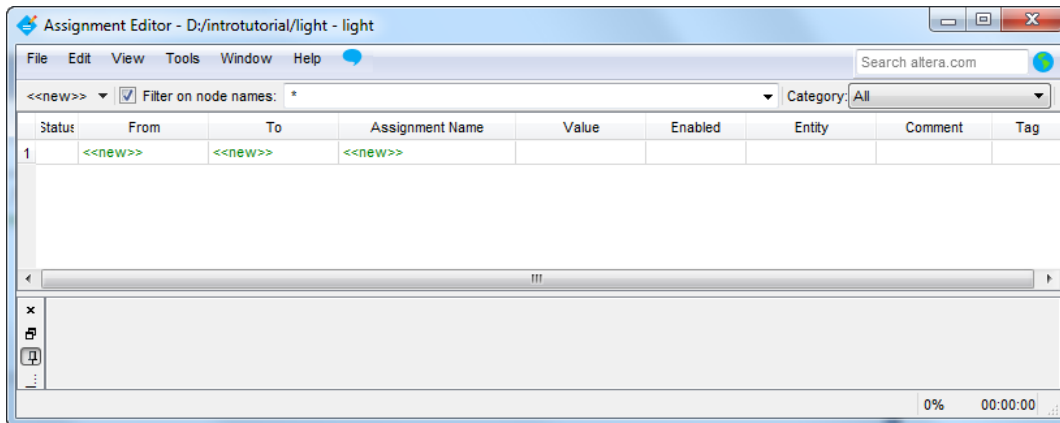





Figure 23. The Assignment Editor window.

Pin assignments are made by using the Assignment Editor. Select **Assignments > Assignment Editor** to reach the window in Figure 23 (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 24. Click on  and  to show or hide 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 25 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  $x2$  and output  $f$  to the appropriate pins listed in Table 2. An example using a DE1-SoC board is shown in Figure 26. 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.

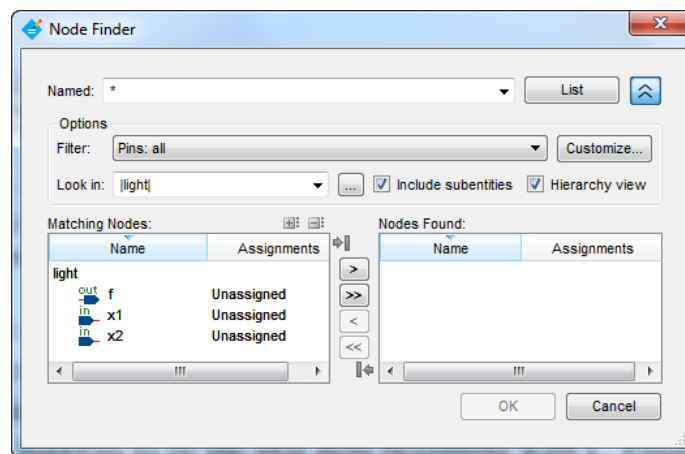


Figure 24. The Node Finder displays the input and output names.

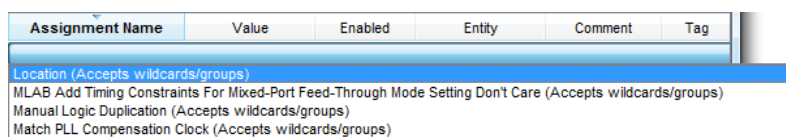


Figure 25. The available assignment names for a DE-series board.

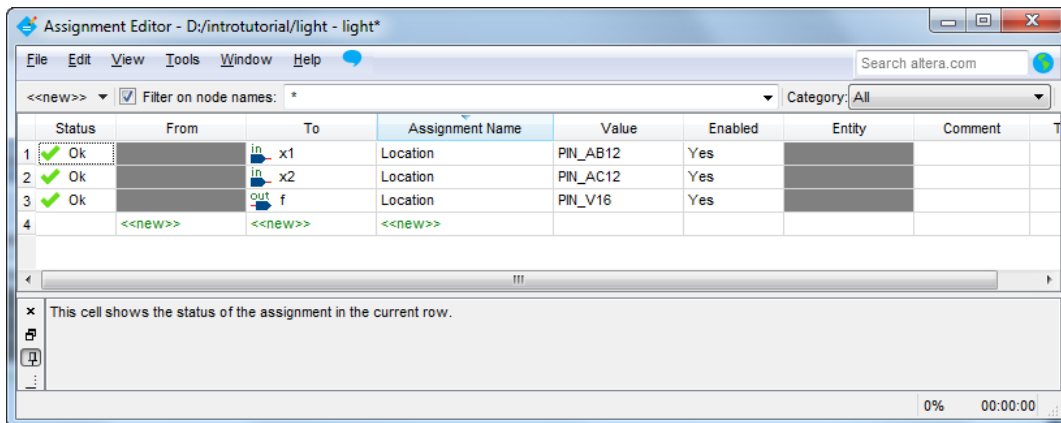


Figure 26. The complete assignment.

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 Prime 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 Settings File (QSF)* format. The format for the file for our simple project (on a DE1-SoC board) is

```
set_location_assignment PIN_AB12 -to x1
set_location_assignment PIN_AC12 -to x2
set_location_assignment PIN_V16 -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 26. Select **Assignments > Export Assignment** which leads to the window in Figure 27. 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.



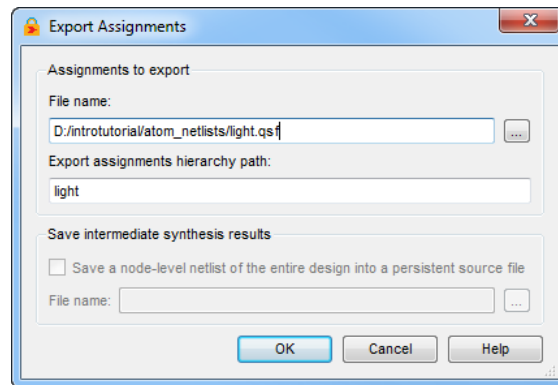


Figure 27. Exporting the pin assignment.

You can import a pin assignment by choosing **Assignments > Import Assignments**. This opens the dialogue in Figure 28 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.

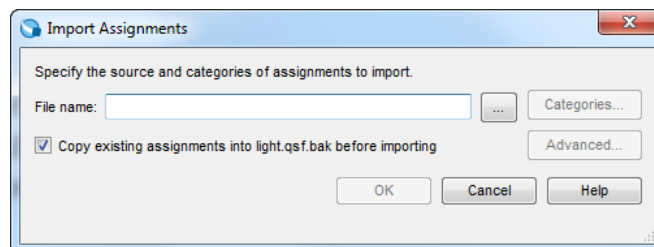


Figure 28. Importing the pin assignment.

For convenience when using large designs, all relevant pin assignments for the DE-series board are given in individual files. For example, the DE1-SoC pin assignments can be found in the *DE1\_SoC.qsf* file, which is available from Intel's FPGA University Program website. This file uses the names found in the *DE1-SoC 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 *DE1\_SoC.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 *DE1\_SoC.qsf* file the 10 toggle switches are called *SW[9]* to *SW[0]*. In a design file they can also be referred to as a vector *SW[9..0]*.

## 8 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 Prime Compiler's Assembler module. Intel'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 Prime software) to the board by means of a cable that connects a USB port on the host computer to the USB-Blaster 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 Intel'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 Prime 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 Prime software if the power is turned off and on. The choice between the two modes is made by switches on the DE-series board. Consult your manual for the location of this switch on your DE-series board. The boards should be set to JTAG mode by default. This tutorial discusses only the JTAG programming mode.

### 8.1 JTAG\* Programming for the DE0-CV, DE0-Nano, DE10-Lite, and DE2-115 Boards

For the DE0-CV, DE0-Nano, DE10-Lite, 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 29. 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 30.

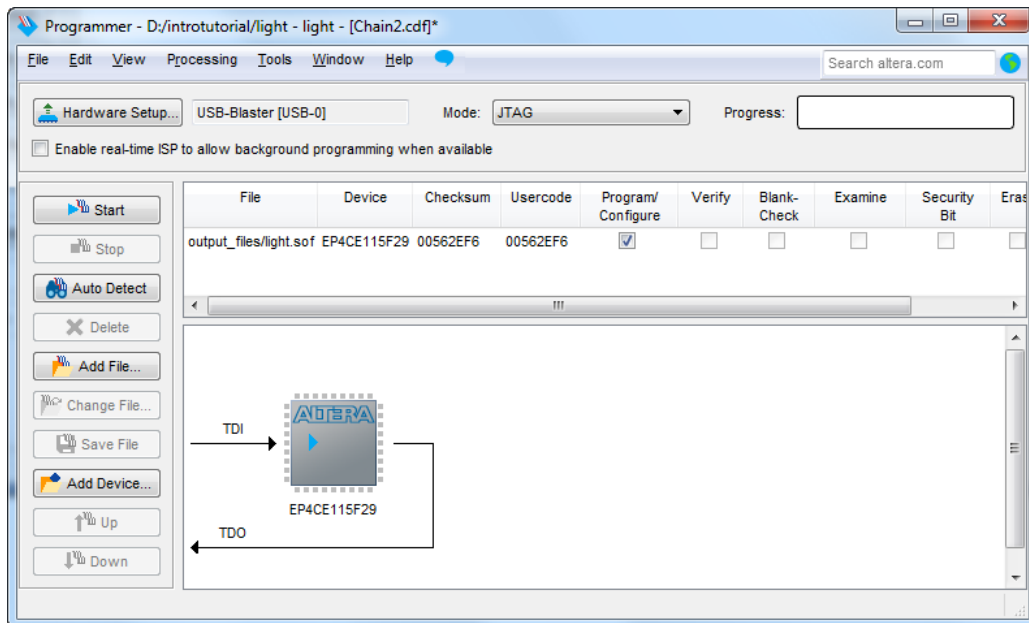


Figure 29. The Programmer window.

Observe that the configuration file *light.sof* is listed in the window in Figure 29. 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. Ensure the **Program/Configure** check box is ticked, as shown in Figure 29.

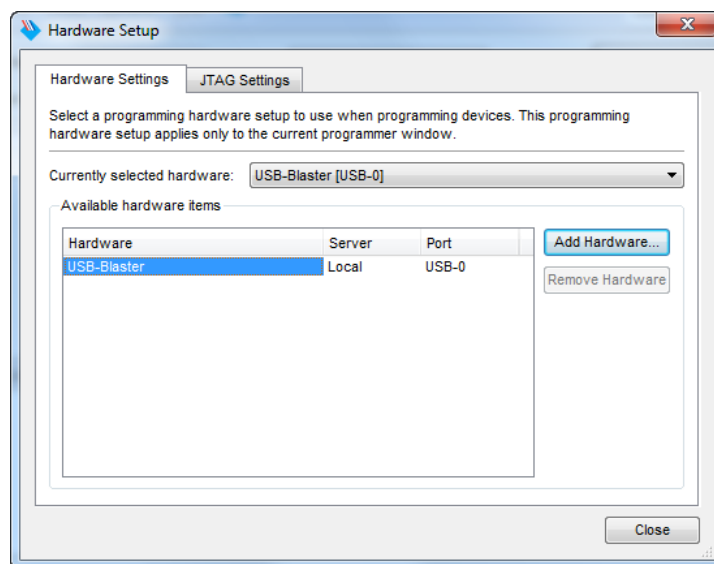


Figure 30. The Hardware Setup window.

Now, press **Start** in the window in Figure 29. An LED on the board will light up corresponding to the programming operation. If you see an error reported by Quartus Prime software indicating that programming failed, then check to ensure that the board is properly powered on.

## 8.2 JTAG\* Programming for the DE0-Nano-SoC, DE1-SoC Board, DE10-Nano, and DE10-Standard

For the DE0-Nano-SoC, DE1-SoC Board, DE10-Nano, and DE10-Standard boards, the following steps should be used for programming. Select **Tools > Programmer** to reach the window in Figure 31. 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 32.

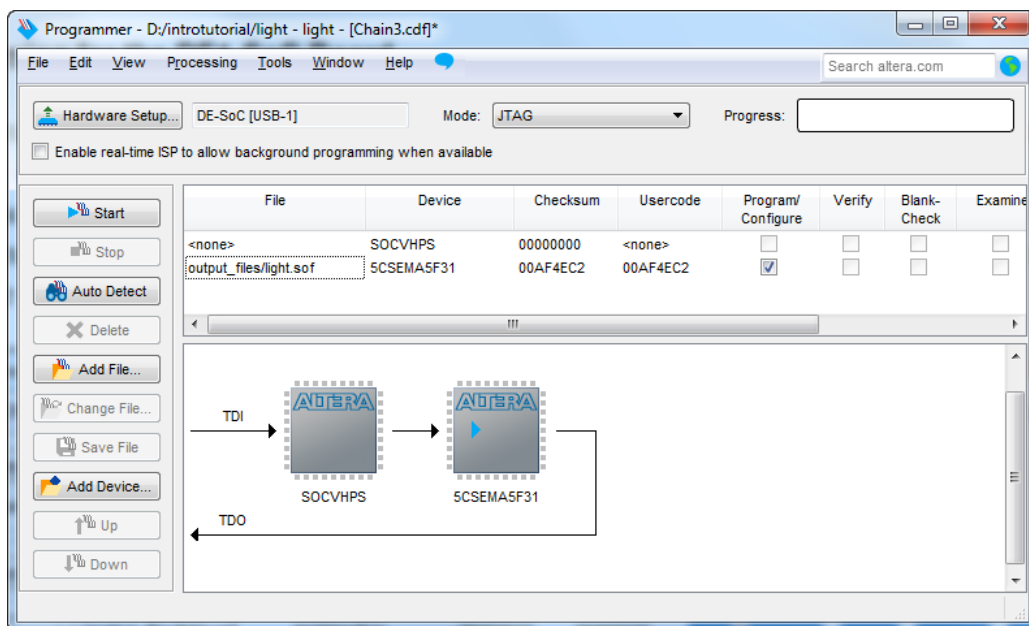


Figure 31. The Programmer window.

Observe that the configuration file *light.sof* in directory *output\_files* is listed in the window in Figure 31. 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. Ensure the **Program/Configure** box is checked. This setting is used to select the FPGA in the Cyclone V SoC chip for programming. If the *SOCVHPS* device is not shown as in Figure 31, click **Add Device > SoC Series V > SOCVHPS** then click **OK**. Ensure that your device order is consistent with Figure 31 by clicking on a device and then clicking **Up** or **Down**.

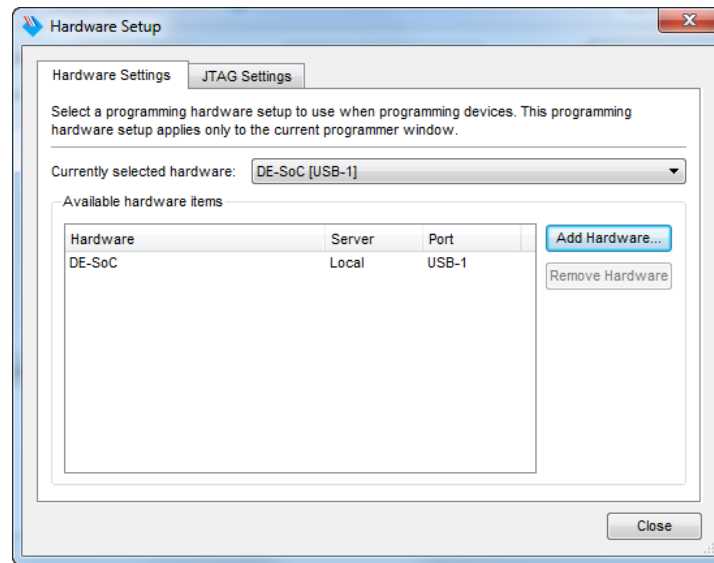


Figure 32. The Hardware Setup window.

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

## 9 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 Prime'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 Prime 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 33. 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 34. You may wish to resize the window to its maximum size.

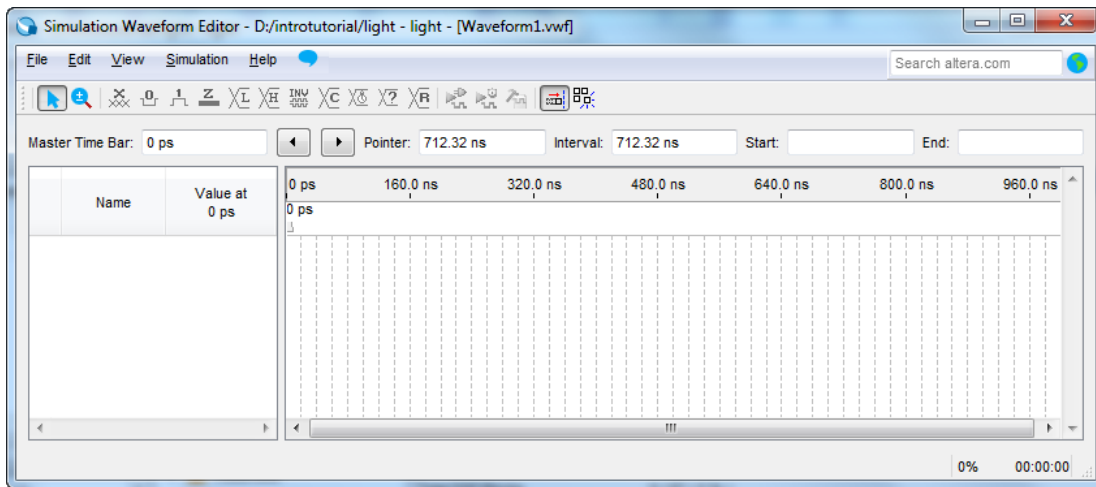


Figure 33. The Waveform Editor window.

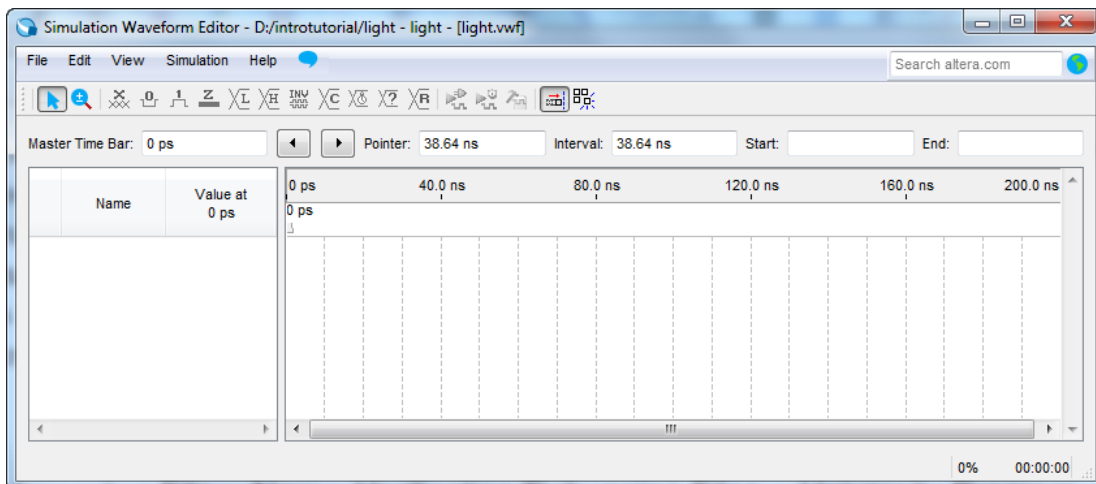


Figure 34. The augmented Waveform Editor window.

- 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 35. 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 36. 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.

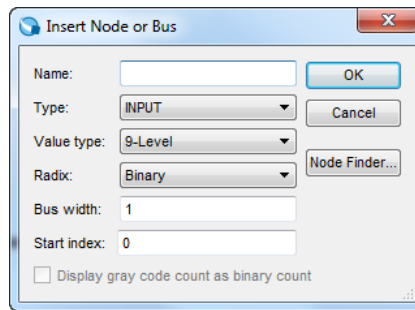


Figure 35. The Insert Node or Bus dialogue.

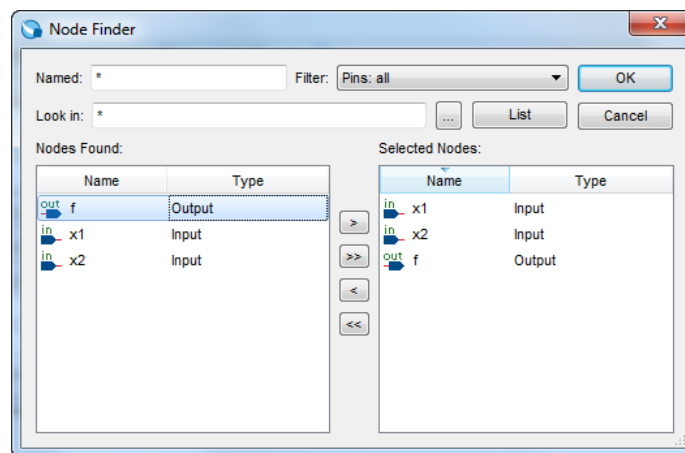


Figure 36. Selecting nodes to insert into the Waveform Editor.

Click on the *x1* signal in the Nodes Found box in Figure 36, 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 35. This leaves a fully displayed Waveform Editor window, as shown in Figure 37. If you did not select the nodes in the same order as displayed in Figure 37, 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.

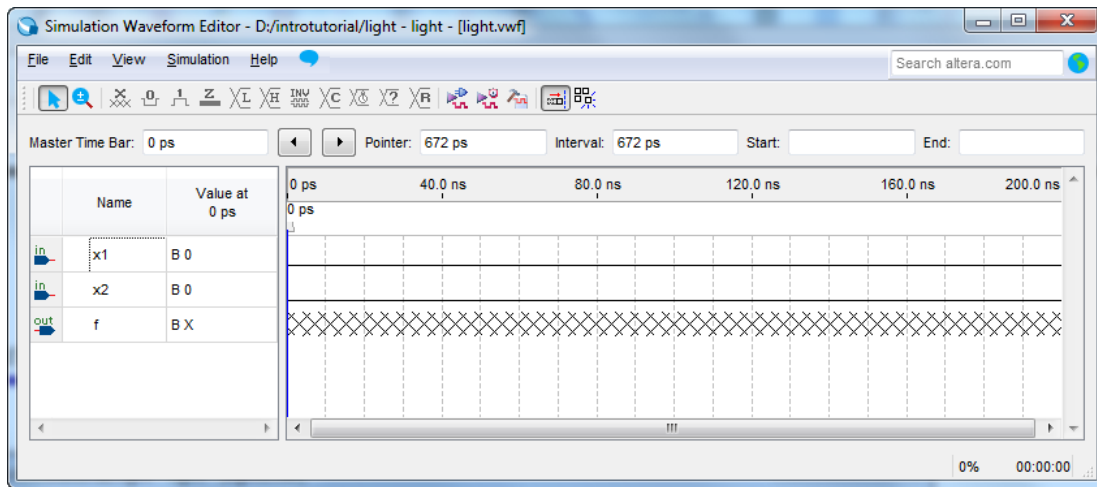



Figure 37. The nodes needed for simulation.

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 12. 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 12. This should produce the image in Figure 38. 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.



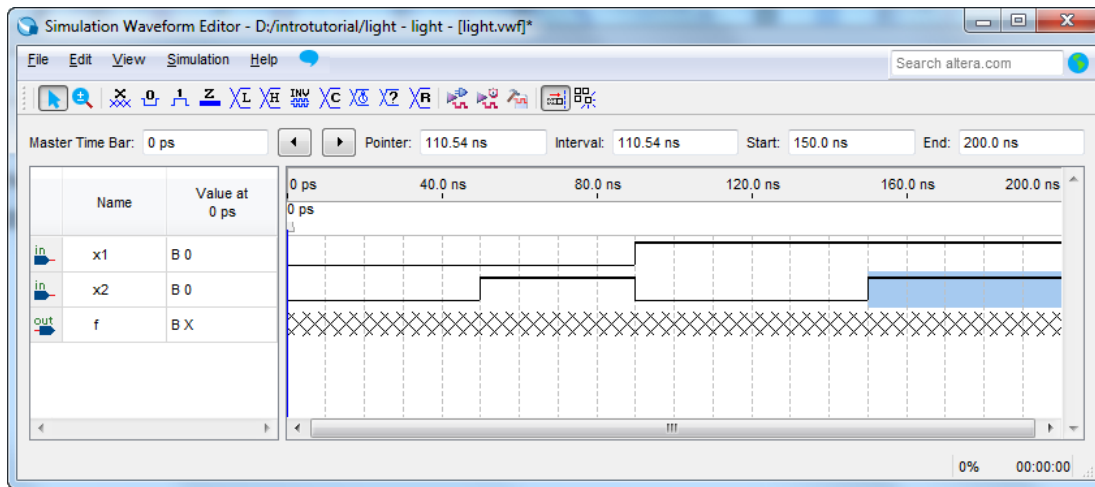




Figure 38. Setting of test values.

## 9.1 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.

### 9.1.1 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 Prime 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 in the Simulation Waveform Editor window. 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 39. Observe that the output *f* is as specified in the truth table of Figure 12.

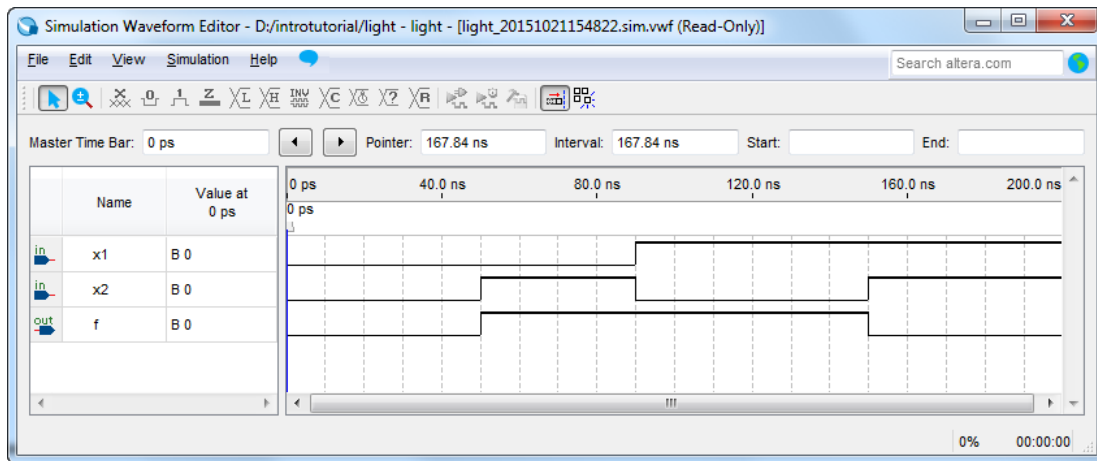




Figure 39. The result of functional simulation.

### 9.1.2 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 Prime 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 in the Simulation Waveform Editor window. The simulation should produce the waveforms in Figure 40. 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.

**Note:** timing simulations are only supported by Cyclone IV and Stratix® IV FPGAs. If your DE-series board does not have a Cyclone IV or Stratix IV FPGA, the result of a timing simulation will be identical to the functional simulation.

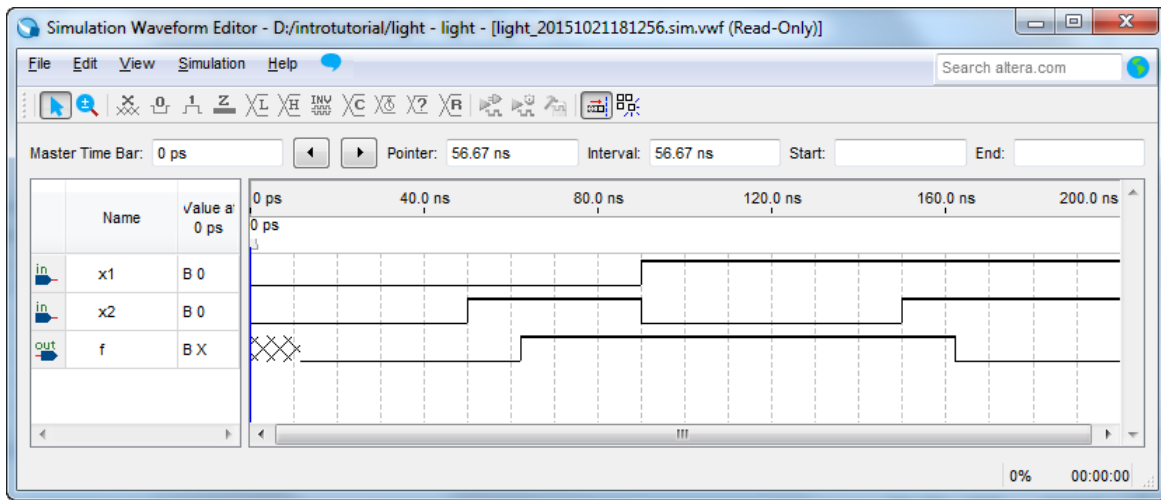


Figure 40. The result of timing simulation.

## 10 Testing the Designed Circuit

Having downloaded the configuration data into the FPGA device, you can now test the implemented circuit. 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 12.

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

Copyright © Intel Corporation. All rights reserved. Intel, the Intel logo, Altera, Arria, Cyclone, Enpirion, MAX, Nios, Quartus and Stratix words and logos are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries. Intel warrants performance of its FPGA and semiconductor products to current specifications in accordance with Intel's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Intel assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Intel. Intel customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

\*Other names and brands may be claimed as the property of others.