



Quartus II Introduction Using VHDL Designs

Appendix

Tutorial — Using Quartus II CAD Software

1 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.

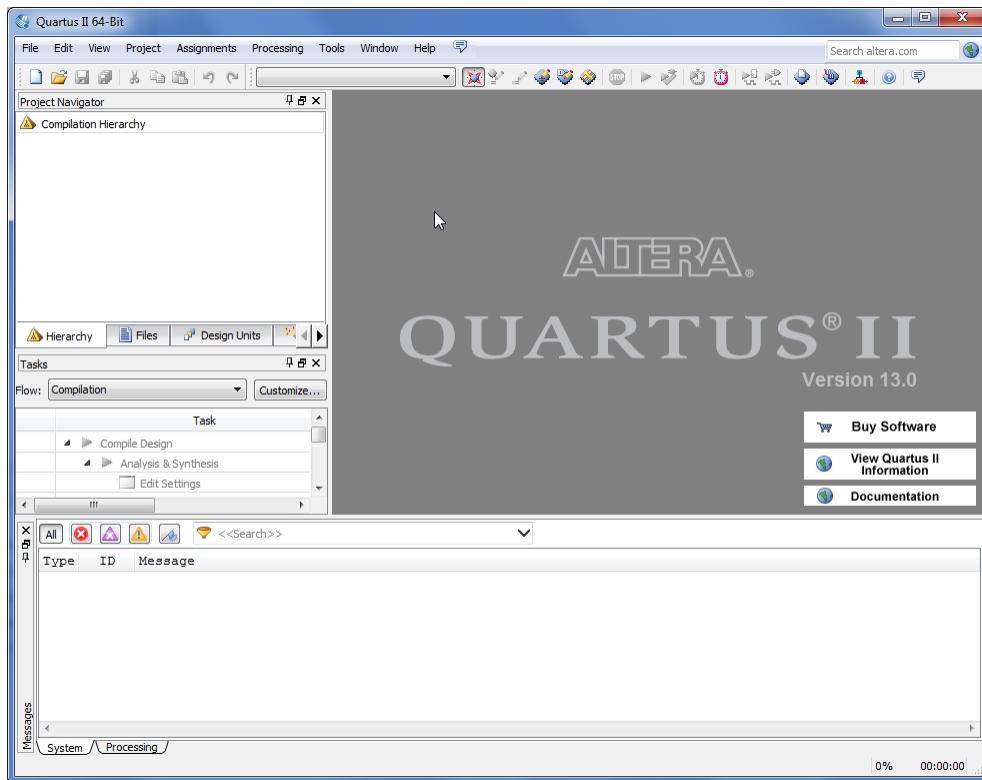


Figure 2. The main Quartus II display.

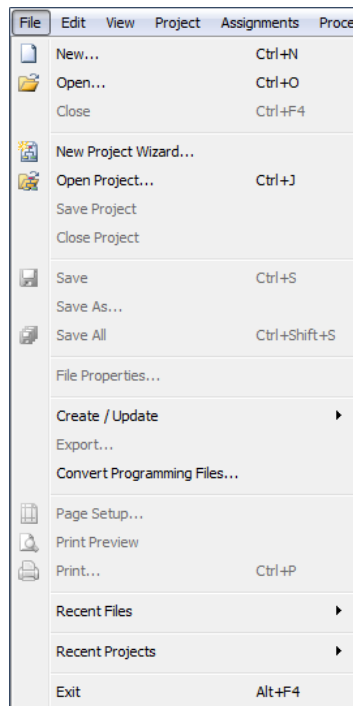


Figure 3. An example of the File menu.

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.

If no web browser is specified, Quartus will complain with an error message. To specify a web browser, go to **Tools > Options... > General > Internet Connectivity**. Specify a path to a web browser in the web browser field.

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.

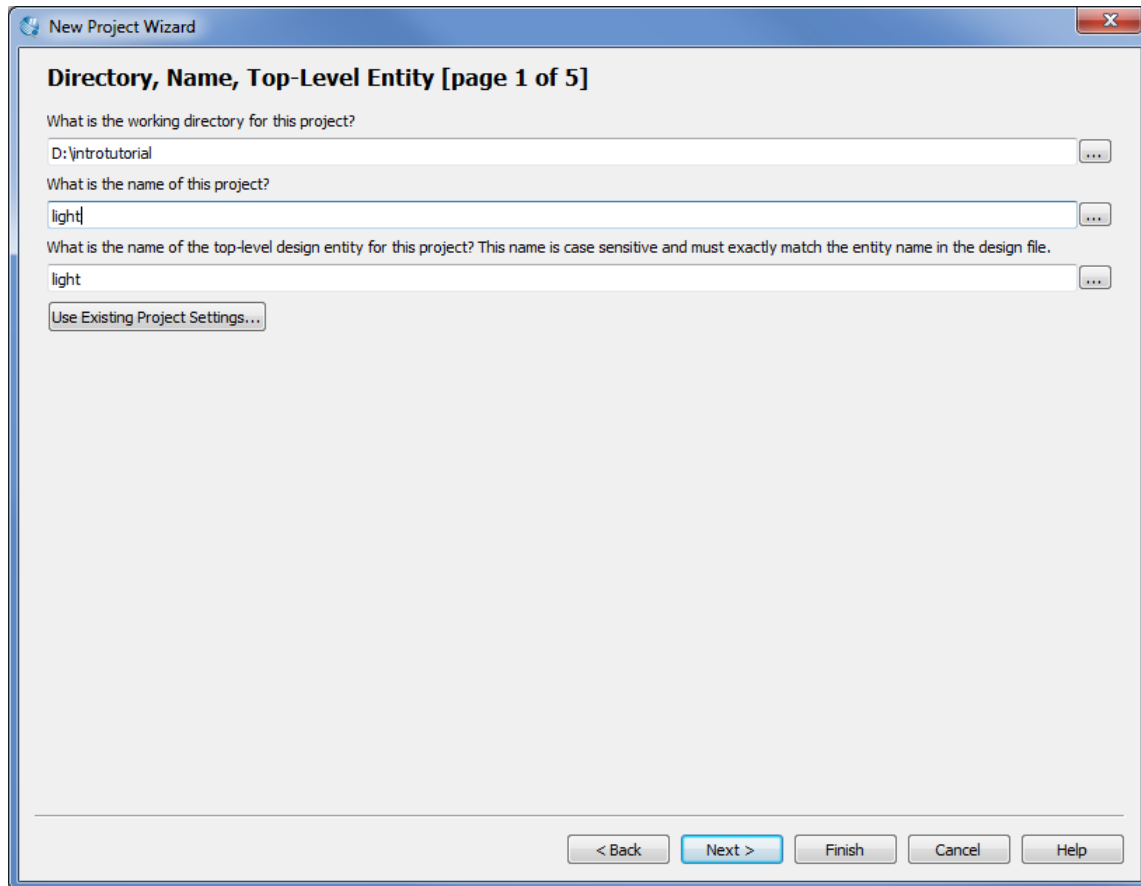


Figure 4. Creation of a new project.

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.

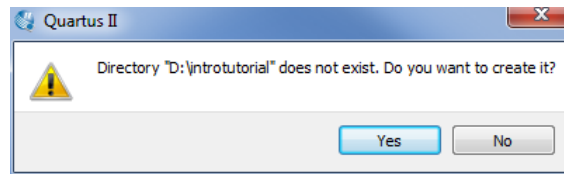


Figure 5. Quartus II software can create a new directory for the project.

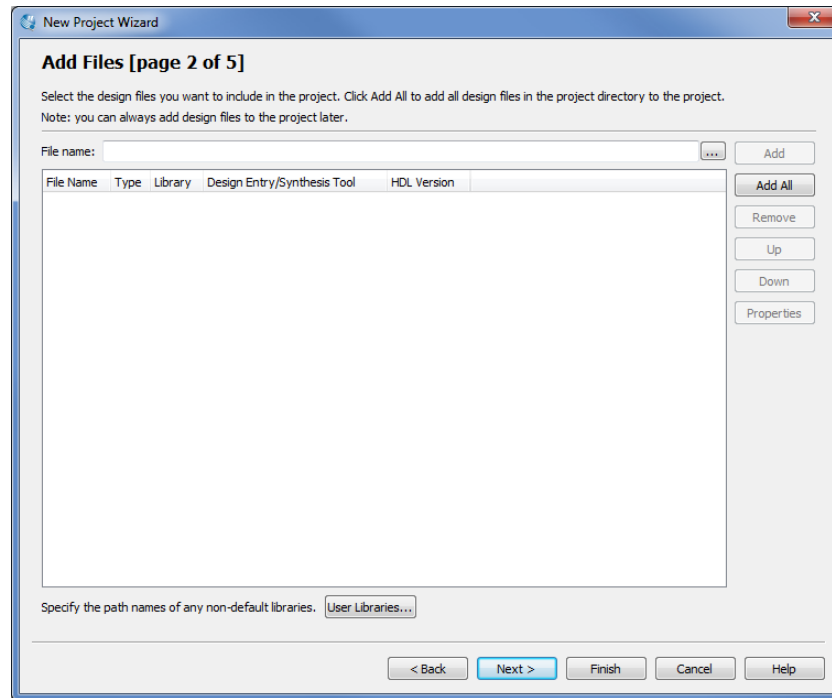


Figure 6. The wizard can include user-specified design files.

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.

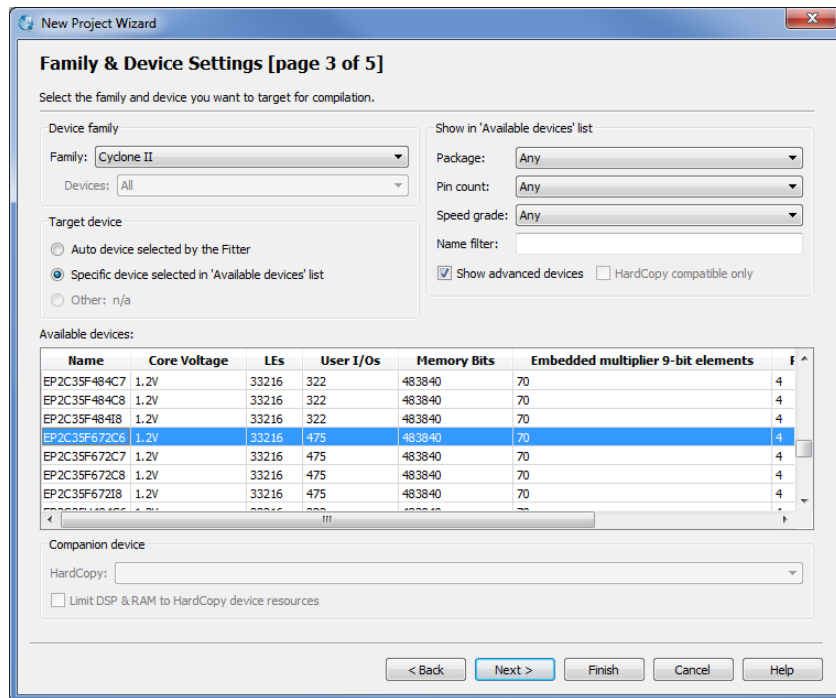


Figure 7. Choose the device family and a specific device.

- 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 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	Cyclone III EP3C16F484C6
DE0-Nano	Cyclone IVE EP4CE22F17C6
DE1	Cyclone II EP2C20F484C7
DE2	Cyclone II EP2C35F672C6
DE2-70	Cyclone II EP2C70F896C6
DE2-115	Cyclone IVE EP4CE115F29C7

Table 1. DE-series FPGA device names

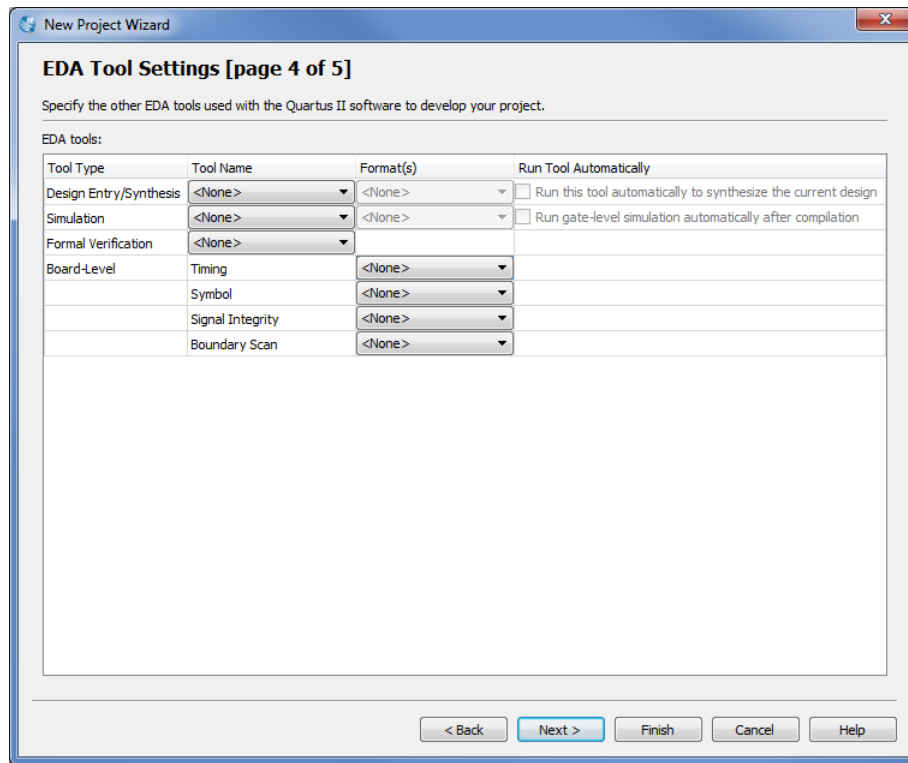


Figure 8. Other EDA tools can be specified.

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.

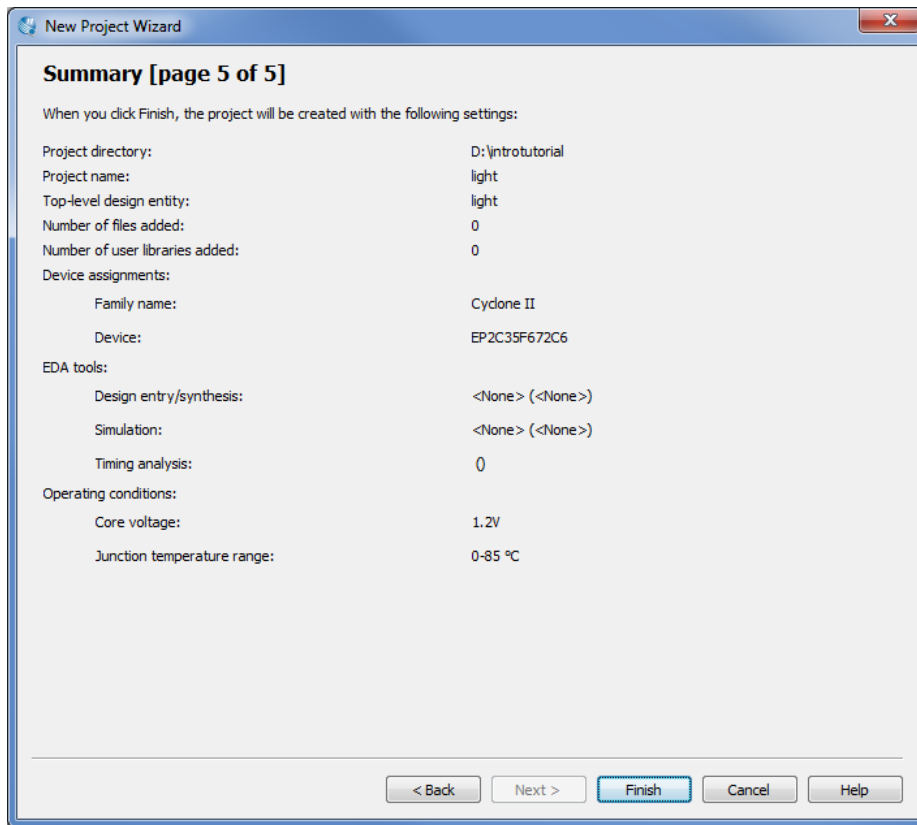


Figure 9. Example summary of a DE2 board project settings.

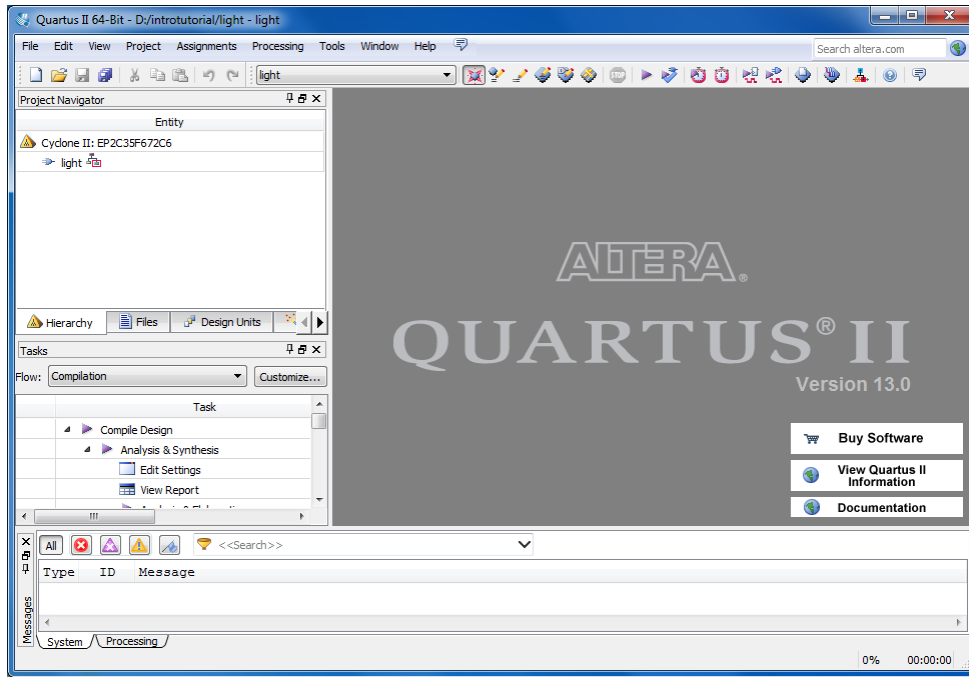


Figure 10. The Quartus II display for created project on a DE2 board.

5 Design Entry Using Schematic Capture

As explained in Chapter 2, commonly used design entry methods include schematic capture and Verilog code. This section illustrates the process of using the schematic capture tool provided in Quartus II, which is called the Block Editor. As a simple example, we will draw a schematic for the logic function $f = x_1x_2 + x_2x_3$. A circuit diagram for f was shown in Figure 2.30 and is reproduced as Figure B.8a. The truth table for f is given in Figure B.8b. Chapter 2 also introduced functional simulation. After creating the schematic, we show how to use the simulator in Quartus II to verify the correctness of the designed circuit.

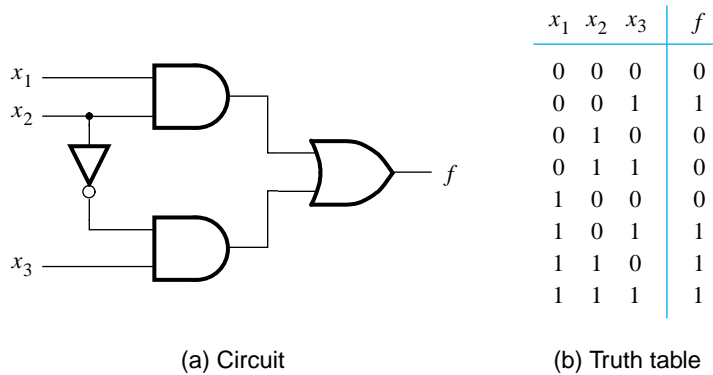


Figure B.8. The logic function of Figure 2.30.

B.3.1 Using the Block Editor

The first step is to draw the schematic. In the Quartus II display select **File | New**. A window that appears, shown in Figure B.9, allows the designer to choose the type of file that should be created. The possible file types include schematics, Verilog code, and other hardware description language files such as VHDL and AHDL (Altera’s proprietary HDL). It is also possible to use a third-party synthesis tool to generate a file that represents the circuit in a standard format called EDIF (Electronic Design Interface Format). The EDIF standard provides a convenient mechanism for exchanging information between EDA tools. Since we want to illustrate the schematic-entry approach in this section, choose **Block Diagram/Schematic File** and click **OK**. This selection opens the Block Editor window shown on the right side of Figure B.10. Drawing a circuit in this window will produce the desired block diagram file.

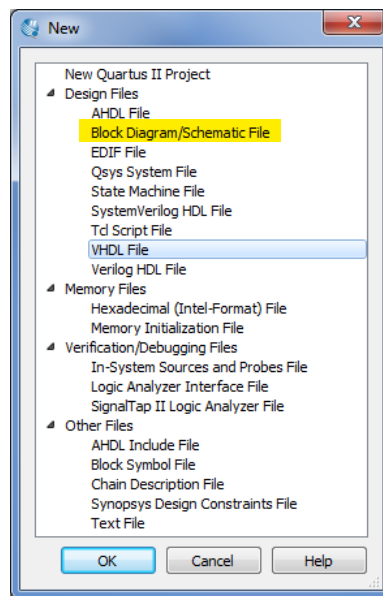


Figure B.9. Choosing the type of design file.

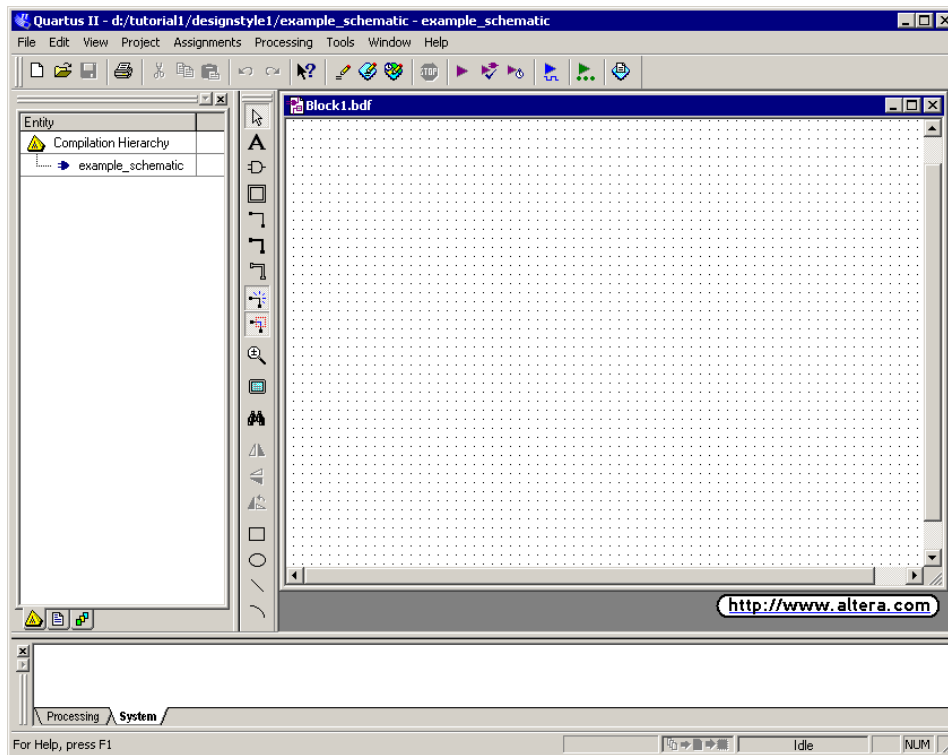


Figure B.10. Block Editor window.

Importing Logic Gate Symbols

The Block Editor provides several libraries that contain circuit elements which can be imported into a schematic. For our simple example we will use a library called *primitives*, which contains basic logic gates. To access the library, double-click on the blank space inside the Block Editor display to open the window in Figure B.11 (another way to open this window is to select **Edit | Insert Symbol** or by clicking on the AND gate symbol in the toolbar). In the figure, the box labeled **Libraries** lists several libraries that are provided with Quartus II. To expand the list, click on the small **+** symbol next to **c:/altera/libraries**, then click on the **+** next to **primitives**, and finally click on the **+** next to **logic**. Now, double-click on the *and2* symbol to import it into the schematic (you can alternatively click on *and2* and then click **OK**). A two-input AND-gate symbol now appears in the Block Editor window. Using the mouse, move the symbol to the position where it should appear in the diagram and place it there by clicking the mouse.

Any symbol in a schematic can be selected by using the mouse. Position the mouse pointer on top of the AND-gate symbol in the schematic and click the mouse to select it. The symbol is highlighted in color. To move a symbol, select it and, while continuing to press the mouse button, drag the mouse to move the symbol. To make it easier to position the graphical symbols, a grid of guidelines can be displayed in the Block Editor window by selecting **View Show Guidelines**.

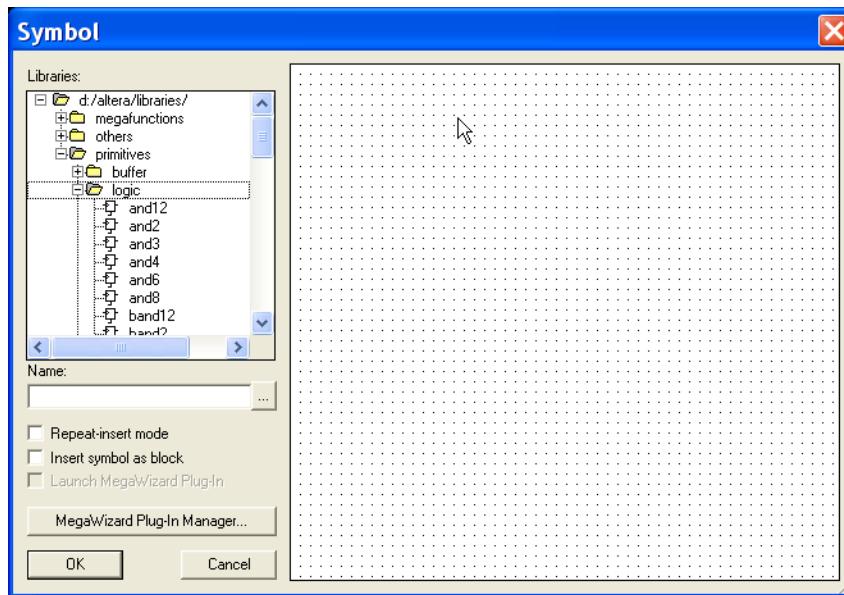


Figure B.11. Selection of logic symbols.

The logic function f requires a second two-input AND gate, a two-input OR gate, and a NOT gate. Use the following steps to import them into the schematic.

Position the mouse pointer over the AND-gate symbol that has already been imported. Press and hold down the Ctrl keyboard key and click and drag the mouse on the AND-gate symbol. The Block Editor automatically imports a second instance of the AND-gate symbol. This shortcut procedure for making a copy of a circuit element is convenient when you need many instances of the same element in a schematic. Of course, an alternative approach is to import each instance of the symbol by opening the primitives library as described above.

To import the OR-gate symbol, again double-click on a blank space in the Block Editor to get to the primitives library. Use the scroll bar to scroll down through the list of gates to find the symbol named *or2*. Import this symbol into the schematic. Next import the NOT gate using the same procedure. To orient the NOT gate so that it points downward, as depicted in Figure B.8a, select the NOT-gate symbol and then use the command `Edit | Rotate by Degrees | 270` to rotate the symbol 270 degrees counterclockwise. The symbols in the schematic can be moved by selecting them and dragging the mouse, as explained above. More than one symbol can be selected at the same time by clicking the mouse and dragging an outline around the symbols. The selected symbols are moved together by clicking on any one of them and moving it. Experiment with this procedure. Arrange the symbols so that the schematic appears similar to the one in Figure B.12.

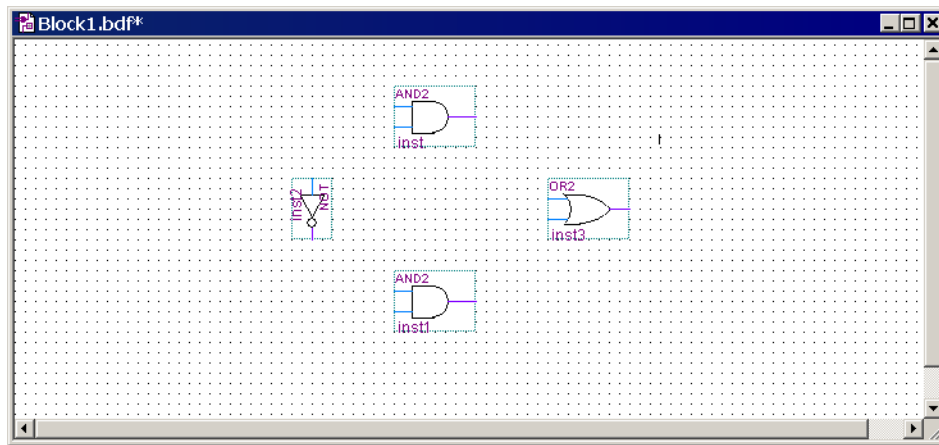


Figure B.12. Imported gate symbols.

Importing Input and Output Symbols

Now that the logic-gate symbols have been entered, it is necessary to import symbols to represent the input and output ports of the circuit. Open the primitives library again. Scroll down past the gates until you reach *pins*. Import the symbol named *input* into the schematic. Import two additional instances of the input symbol. To represent the output of the circuit, open the primitives library and import the symbol named *output*. Arrange the symbols to appear as illustrated in Figure B.13.

Assigning Names to Input and Output Symbols

Point to the word *pin name* on the input pin symbol in the upper-left corner of the schematic and double-click the mouse. The pin name is selected, allowing a new pin name to be typed. Type *x1* as the pin name. Hitting carriage return immediately after typing the pin name causes the mouse focus to move to the pin directly below the one currently being named. This method can be used to name any number of pins. Assign the names *x2* and *x3* to the middle and bottom input pins, respectively. Finally, assign the name *f* to the output pin.

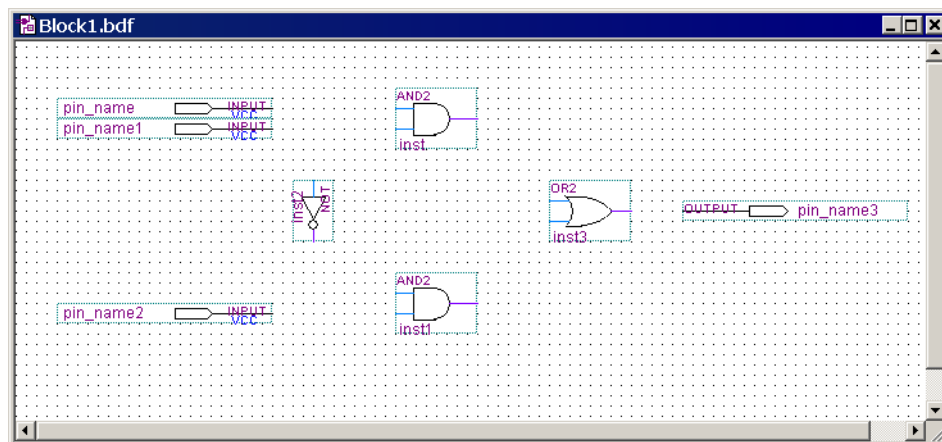


Figure B.13. The desired arrangement of gates and pins.

Connecting Nodes with Wires

The next step is to draw lines (wires) to connect the symbols in the schematic together. Click on the icon that looks like a big arrowhead in the vertical toolbar. This icon is called the **Selection and Smart Drawing** tool, and it allows the Block Editor to change automatically between the modes of selecting a symbol on the screen or drawing wires to interconnect symbols. The appropriate mode is chosen depending on where the mouse is pointing.

Move the mouse pointer on top of the $x1$ input symbol. When pointing anywhere on the symbol except at the right edge, the mouse pointer appears as crossed arrowheads. This indicates that the symbol will be selected if the mouse button is pressed. Move the mouse to point to the small line, called a *pinstub*, on the right edge of the $x1$ input symbol. The mouse pointer changes to a crosshair, which allows a wire to be drawn to connect the pinstub to another location in the schematic. A connection between two or more pinstubs in a schematic is called a *node*. The name derives from electrical terminology, where the term *node* refers to any number of points in a circuit that are connected together by wires.

Connect the input symbol for $x1$ to the AND gate at the top of the schematic as follows. While the mouse is pointing at the pinstub on the $x1$ symbol, click and hold the mouse button. Drag the mouse to the right until the line (wire) that is drawn reaches the pinstub on the top input of the AND gate; then release the button. The two pinstubs are now connected and represent a single node in the circuit.

Use the same procedure to draw a wire from the pinstub on the $x2$ input symbol to the other input on the AND gate. Then draw a wire from the pinstub on the input of the NOT gate upward until it reaches the wire connecting $x2$ to the AND gate. Release the mouse button and observe that a connecting dot is drawn automatically. The three pinstubs corresponding to the $x2$ input symbol, the AND-gate input, and the NOT-gate input now represent a single node in the circuit. Figure B.14 shows a magnified view of the part of the schematic that contains the connections drawn so far. To increase or decrease the portion of the schematic displayed on the screen, use the icon that looks like a magnifying glass in the toolbar.

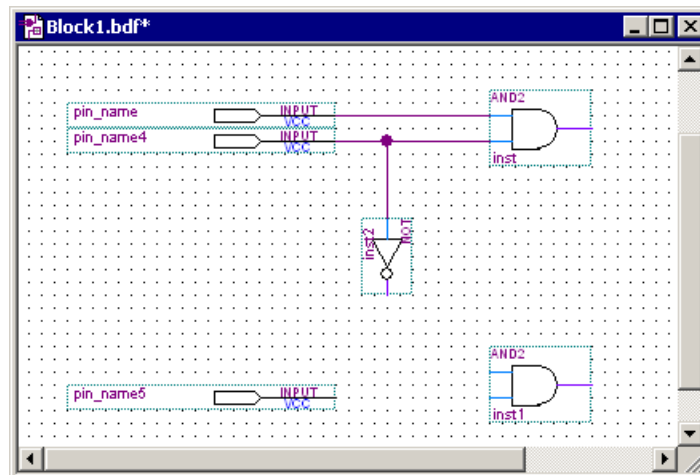


Figure B.14. Expanded view of the circuit.

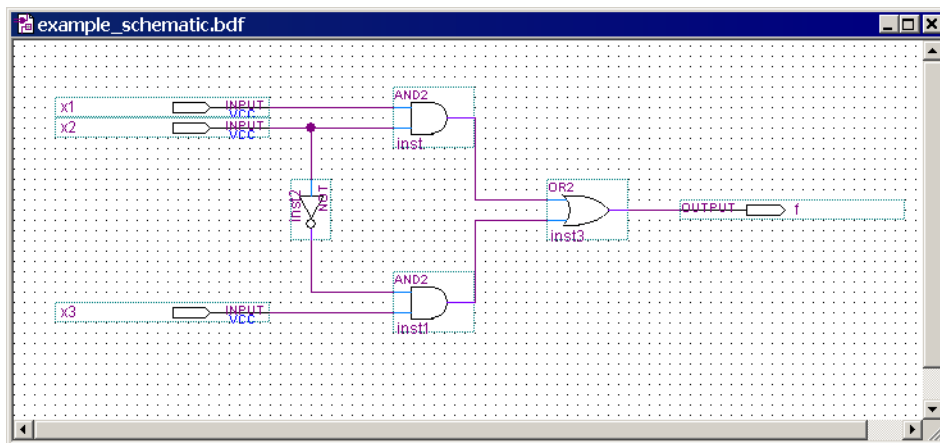


Figure B.15. The completed schematic.

To complete the schematic, connect the output of the NOT gate to the lower AND gate and connect the input symbol for x_3 to that AND gate as well. Connect the outputs of the two AND gates to the OR gate and connect the OR gate to the f output symbol. If any mistakes are made while connecting the symbols, erroneous wires can be selected with the mouse and then removed by pressing the Delete key or by selecting Edit | Delete. The finished schematic is depicted in Figure B.15. Save the schematic using File | Save As and choose the name *example_schematic*. Note that the saved file is called *example_schematic.bdf*.

Try to rearrange the layout of the circuit by selecting one of the gates and moving it. Observe that as you move the gate symbol all connecting wires are adjusted automatically. This takes place because Quartus II has a feature called *rubberbanding* which was activated by default when you chose to use the Selection and Smart Drawing tool. There is a rubberbanding icon, which is the icon in the toolbar that looks like an L-shaped wire with small tick marks on the corner. Observe that this icon is highlighted to indicate the use of rubberbanding. Turn the icon off and move one of the gates to see the effect of this feature.


Since our example schematic is quite simple, it is easy to draw all the wires in the circuit without producing a messy diagram. However, in larger schematics some nodes that have to be connected may be far apart, in which case it is awkward to draw wires between them. In such cases the nodes are connected by assigning labels to them, instead of drawing wires. See Help for a more detailed description.

B.3.2 Synthesizing a Circuit from the Schematic

After a schematic is entered into a CAD system, it is processed by a number of CAD tools. We showed in Chapter 2 that the first step in the CAD flow uses the synthesis tool to translate the schematic into logic expressions. Then, the next step in the synthesis process, called technology mapping, determines how each logic expression should be implemented in the logic elements available in the target chip.

6 Compiling the Designed Circuit

The block schematic in the .BDF file 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.

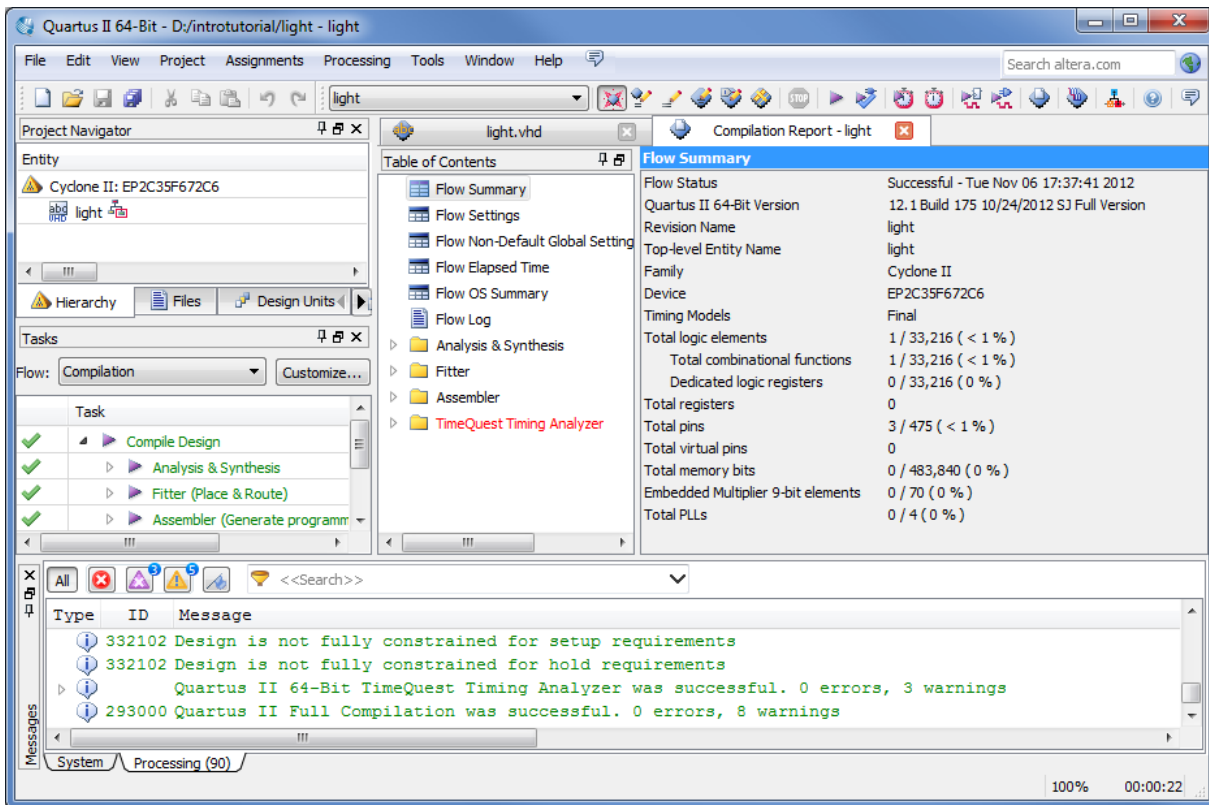



Figure 18. Display after a successful compilation.

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.

Component	DE0	DE0-Nano	DE1	DE2	DE2-70	DE2-115
SW_0	PIN_J6	PIN_M1	PIN_L22	PIN_N25	PIN_AA23	PIN_AB28
SW_1	PIN_H5	PIN_T8	PIN_L21	PIN_N26	PIN_AB26	PIN_AC28
$LEDG_0$	PIN_J1	PIN_A15	PIN_U22	PIN_AE22	PIN_W27	PIN_E21

Table 2. DE-Series Pin Assignments

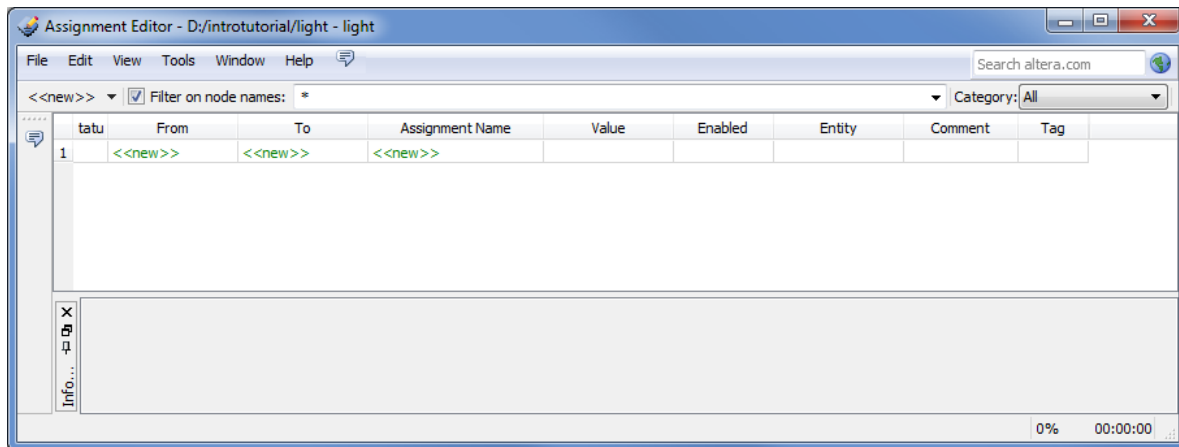


Figure 22. The Assignment Editor window.

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.

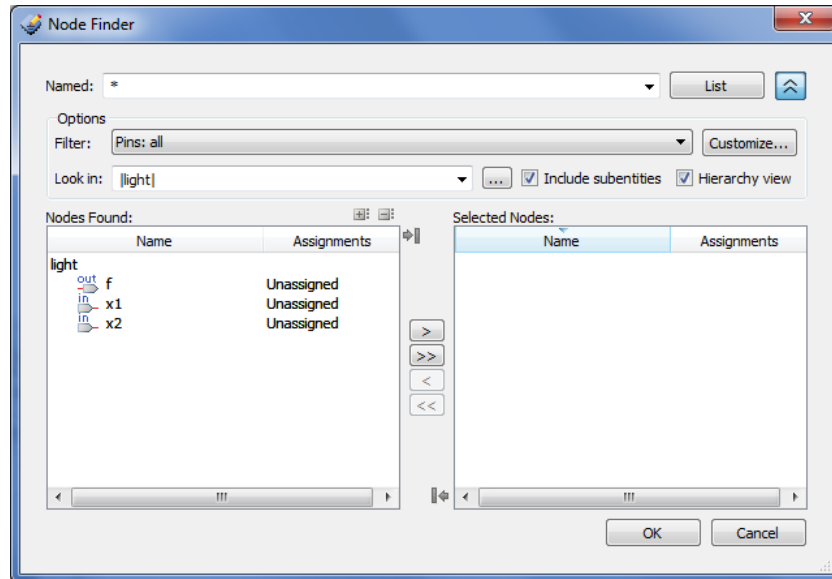


Figure 23. The Node Finder displays the input and output names.

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 *SW₀* 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 DE2 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.

Assignment Name	Value	Enabled	Entity	Comment	Tag
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 board.

tatu	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1 ✓	in	x1	Location	PIN_N25	Yes			
2 ✓	in	x2	Location	PIN_N26	Yes			
3 ✓	out	f	Location	PIN_AE22	Yes			
4	<<new>>	<<new>>	<<new>>					

Assigns a location on the device for the current node(s) and/or pin(s).

Figure 25. The complete assignment on a DE2 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 board) is

```
set_location_assignment PIN_N25 -to x1
set_location_assignment PIN_N26 -to x2
set_location_assignment PIN_AE22 -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.

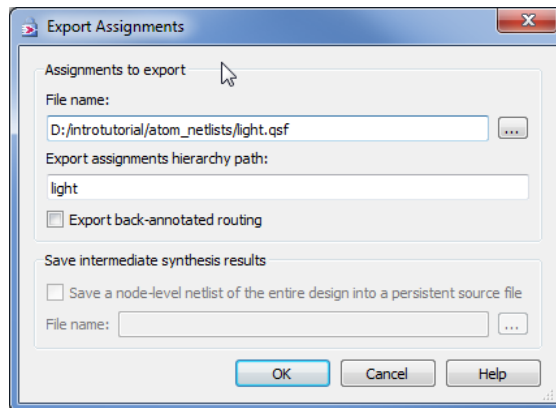


Figure 26. Exporting the pin assignment.

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.

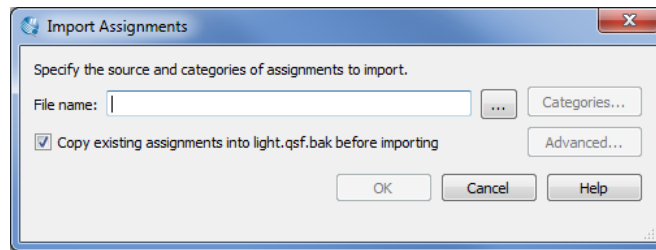


Figure 27. Importing the pin assignment.

For convenience when using large designs, all relevant pin assignments for the DE-series board are given in individual files. For example, the DE2 pin assignments can be found in the *DE2_pin_assignments.qsf* file, in the directory *tutorials\design_files*, which is included on the CD-ROM that accompanies the DE-series board and can also be found on Altera's DE-series web pages. This file uses the names found in the *DE2 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 VHDL design file; namely, *SW[0]*, *SW[1]* and *LEDG[0]* for *x1*, *x2* and *f*, respectively. Since these signals are specified in the *qsf* file as elements of vectors *SW* and *LEDG*, we must refer to them in the same way in the VHDL design file. For example, in the *qsf* file the 18 toggle switches are called *SW[17]* to *SW[0]*; since VHDL uses parentheses rather than square brackets, these switches are referred to as *SW(17)* to *SW(0)*. They can also be referred to as an array *SW(17 downto 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 *lab1.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.

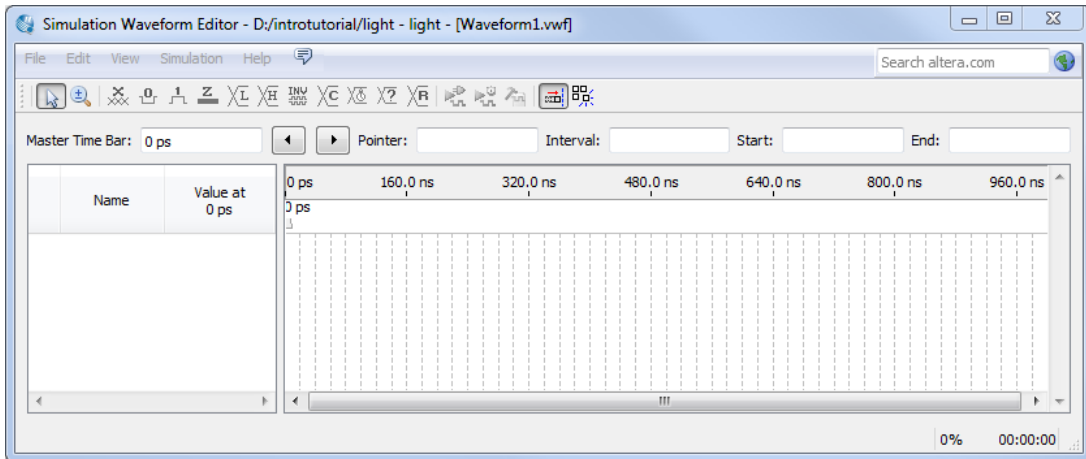


Figure 28. The Waveform Editor window.

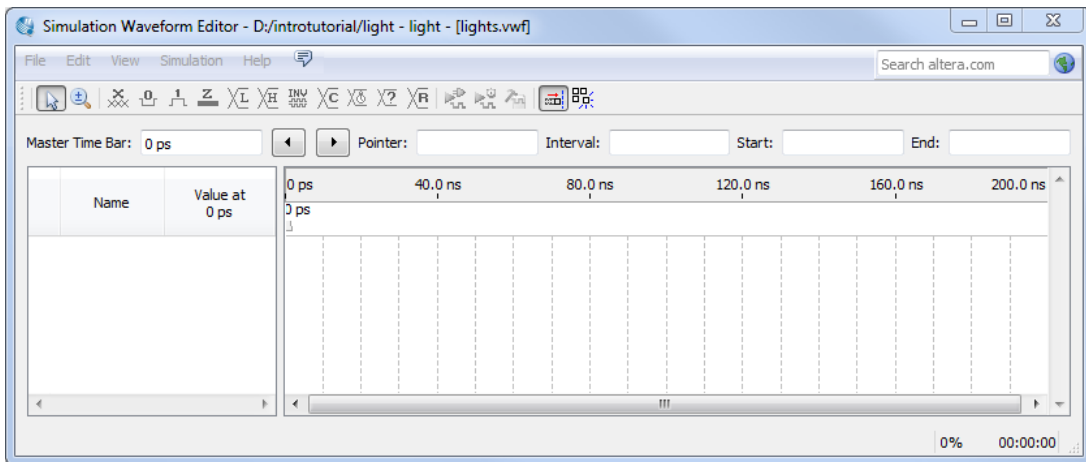


Figure 29. The augmented Waveform Editor window.

- 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 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.

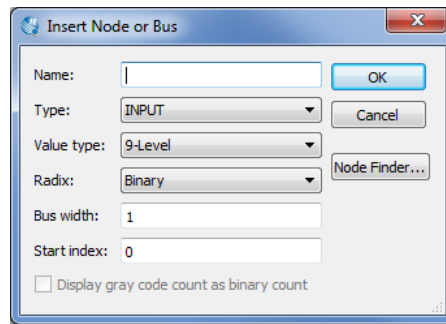


Figure 30. The Insert Node or Bus dialogue.

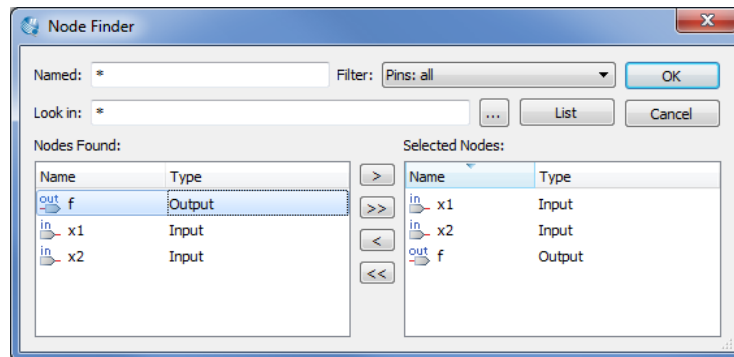


Figure 31. Selecting nodes to insert into the Waveform Editor.

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.

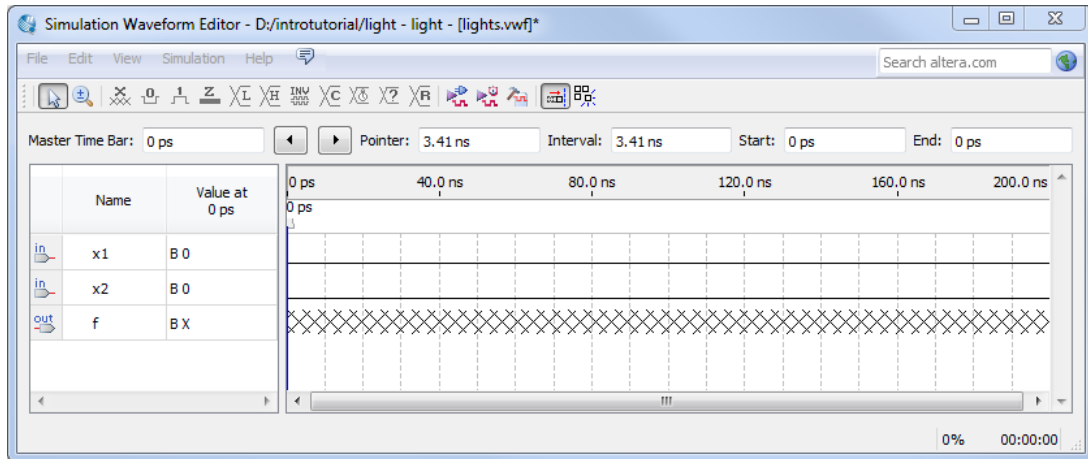



Figure 32. The nodes needed for simulation.

- 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.

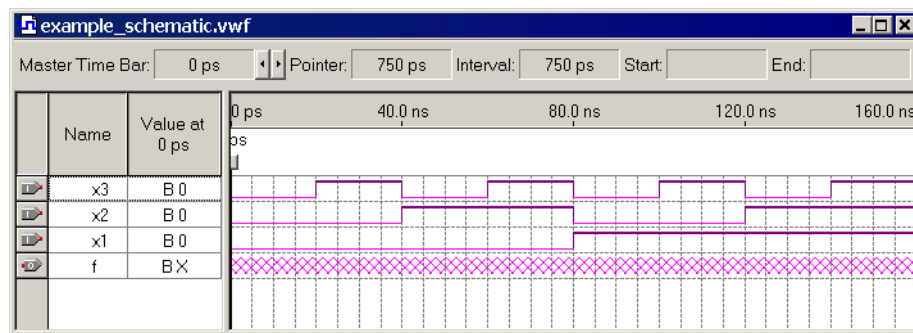




Figure 33. Setting of test values.

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 B.8a](#).

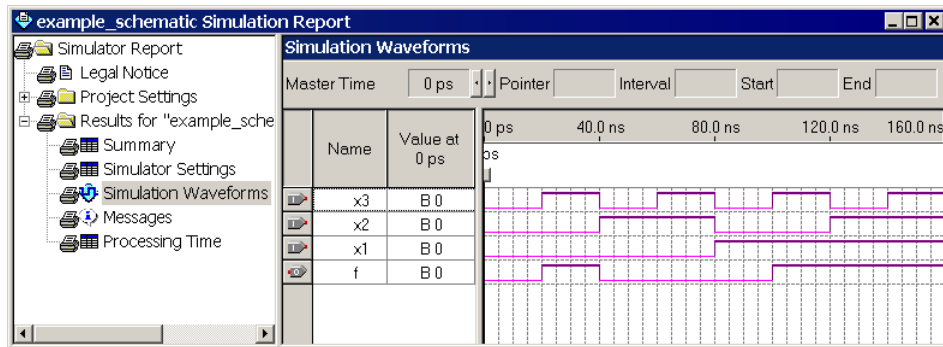


Figure 34. The result of functional simulation.

B.4 Design Entry Using Verilog

This section illustrates the process of using Quartus II to implement logic functions by writing Verilog code. We will implement the function f from section B.3, where we used schematic capture. After entering the Verilog code, we will simulate it using functional simulation.

B.4.1 Create Another Project

Create a new project for the Verilog design in the directory *tutorial1\designstyle2*. Use the New Project Wizard to create the project as explained in section B.2. Call the project *example verilog* and choose the same FPGA chip family for implementation. Note that we are creating this project in a new directory, *designstyle2*, which is a subdirectory of the directory *tutorial1*. While we could have created a new project, *example verilog*, in the previous directory *designstyle1*, it is a good practice to create different projects in separate directories.

B.4.2 Using the Text Editor

Quartus II provides a text editor that can be used for typing Verilog code. Select **File** **New** to get the window in Figure B.9, 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 B.26. In the box labeled **Save as type** choose **Verilog HDL File**. In the box labeled **File name type** *example verilog*. (Quartus II will add the filename extension *v*, which must be used for all files that contain Verilog code.) Leave the box checked at the bottom of the figure, which specifies **Add file to current project**. This setting informs Quartus II that the new file is part of the currently open project. Save the file. We should mention that it is not necessary to use the Text Editor provided in Quartus II. Any text editor can be used to create the file named *example verilog.v*, as long as the text editor can generate a plain text (ASCII) file. A file created using another text editor can be placed in the directory *tutorial1\designstyle2* and included in the project by specifying it in the New Project Wizard screen shown in Figure B.5 or by identifying it in the Settings window of Figure B.24 under the category **Files**.

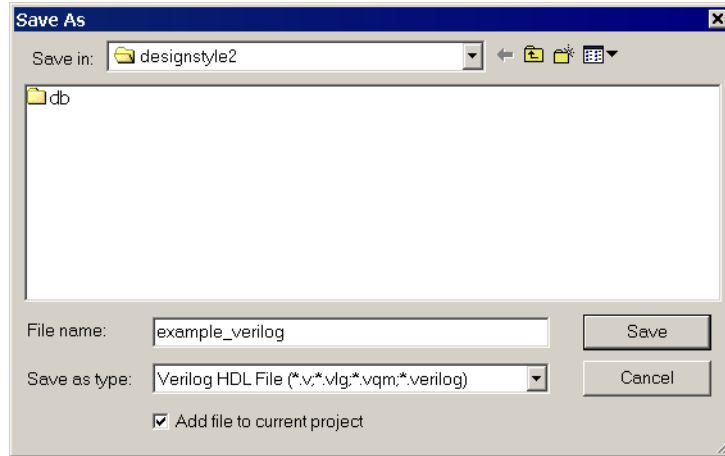


Figure B.26. Opening a new Verilog file.

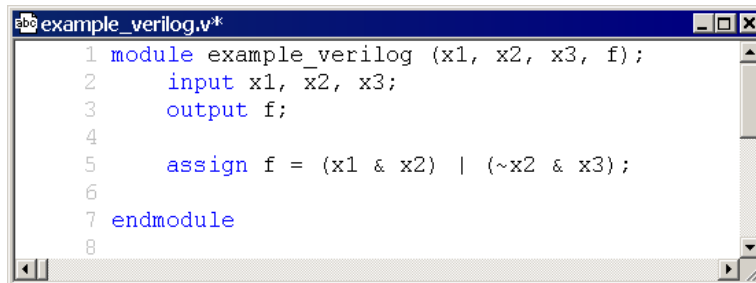


Figure B.27. The Verilog code entered in the Text Editor.

The Verilog code for this example is shown in Figure 2.34. Enter this code into the Text Editor window, with one small modification. In Figure 2.34, the name of the module is *example3*. When creating the new project, we chose the name *example_verilog* for the top-level design entity. Hence, the Verilog module must match this name. The typed code should appear as shown in Figure B.27. Save the file, by using File | Save or 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 by using either the keyboard arrow keys or the mouse. Two features of the Text Editor are especially convenient for typing Verilog code. First, the editor displays different types of Verilog statements in different colors, and, 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.

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.

B.4.3 Synthesizing a Circuit from the Verilog Code

As described for the design created with schematic capture in section B.3.2, select **Processing | Start | Start Analysis and Synthesis** (shortcut Ctrl-k) so that the Compiler will synthesize a circuit that implements the given Verilog code. If the Verilog code has been typed correctly, the Compiler will display a message that says that no errors or warnings were generated. A summary of the compilation report will be essentially the same as in Figure B.17.

If the Compiler does not report zero errors, then at least one mistake was made when typing 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 particular error or warning message by selecting the message and pressing the F1 key.

B.4.4 Performing Functional Simulation

Functional simulation of the Verilog code is done in exactly the same way as the simulation described earlier for the design created with schematic capture. Create a new Waveform Editor file and select **File | Save As** to save the file with the name *example_verilog.vwf*. Following the procedure given in section B.3.3, import the nodes in the project into the Waveform Editor. Draw the waveforms for inputs x_1 , x_2 , and x_3 shown in Figure B.23. It is also possible to open the previously drawn waveform file *example_schematic.vwf* and then “copy and paste” the waveforms for x_1 , x_2 , and x_3 . The procedure for copying waveforms is described in **Help**; it follows the standard Windows procedure for copying and pasting. We should also note that since the contents of the two files are identical, we can simply make a copy of the *example_schematic.vwf* file and save it under the name *example_verilog.vwf*.

Select the Functional Simulation option in Figure B.24 and select **Processing | Generate Functional Simulation Netlist**. Start the simulation. The waveform generated by the Simulator for the output f should be the same as the waveform in Figure B.25.

B.4.5 Using Quartus II to Debug Verilog Code

In section B.3.2 we showed that the displayed messages can be used to quickly locate and fix errors in a schematic. A similar procedure is available for finding errors in Verilog code. To illustrate this feature, open the *example_verilog.v* file with the Text Editor. In the fifth line, which is the **assign** statement, delete the semicolon at the end of the line. Save the *example_verilog.v* file and then run the Compiler again. The Compiler detects one error and displays the messages shown in Figure B.28. The error message specifies that the problem was identified when processing line 7 in the Verilog source code file. Double-click on this message to locate the corresponding part of the Verilog code. The Text Editor window is automatically displayed with line 7 highlighted.

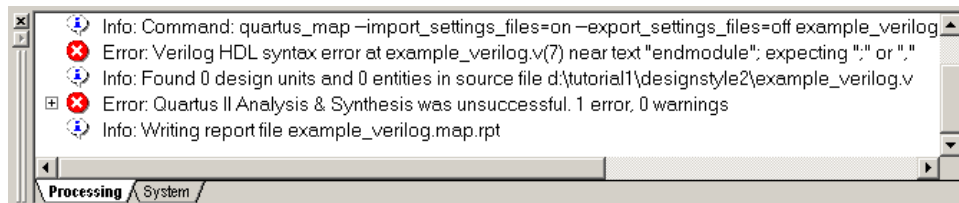


Figure B.28. The Message window.

Fix the error by reinserting the missing semicolon; then save the file and run the Compiler again to confirm that the error is fixed. We have now completed the introduction to design using Verilog code. Close this project.

B.5 Mixing Design-Entry Methods

It is possible to design a logic circuit using a mixture of design-entry methods. As an example, we will design a circuit that implements the function

$$f = x_1x_2 + \overline{x_2}x_3$$

where

$$x_1 = w_1w_2 + w_3w_4$$

$$x_3 = w_1w_3 + w_2w_4$$

Hence, the circuit has five inputs, x_2 and w_1 through w_4 , and an output f . We already designed a circuit for

$$f = x_1x_2 + \overline{x_2}x_3$$

in section B.3 by using the schematic entry approach. To show how schematic capture and Verilog can be mixed, we will create Verilog code for expressions x_1 and x_3 , and then make a top-level schematic that connects this Verilog subcircuit to the schematic created in section B.3.

B.5.1 Using Schematic Entry at the Top Level

Using the approach explained in section B.2, create a new project in a directory named *tutorial1\designstyle3*. Use the name *example_mixed1* for both the project and the top-level entity. For the New Project Wizard's screens in Figures B.5 to B.7, use the same settings as we did in section B.2. With the *example_mixed1* project open, select File | New to open the window in Figure B.9, and select Verilog HDL as the type of file to create. Type the code in Figure B.29 and then save the file with the name *verfunctions.v*.

```
module verfunctions (w1, w2, w3, w4, g, h);  
  input w1, w2, w3, w4;  
  output g, h;  
  
  assign g = (w1 & w2) | (w3 & w4);  
  assign h = (w1 & w3) | (w2 & w4);  
  
endmodule
```

Figure B.29. Verilog code for the *verfunctions* subcircuit.

To include the subcircuit represented by *verfunctions.v* in a schematic we need to create a symbol for this file that can be imported into the Block Editor. To do this, select File | Create/Update | Create Symbol Files for Current File. In response, Quartus II generates a Block Symbol File, *verfunctions.bsf*, in the *tutorial1\designstyle3* directory.

We also wish to use the *example_schematic* circuit created in section B.2 as a subcircuit in the *example_mixed1* project. In the same way that we needed to make a symbol for *verfunctions*, a Block Editor

symbol is required for *example_schematic*. Select **File | Open** and browse to open the file *tutorial1\designstyle1\example_schematic.bdf*. Now, select **File | Create/Update | Create Symbol Files for Current File**. Quartus II will generate the file *example_schematic.bsf* in the *designstyle1* directory. Close the *example_schematic.bdf* file.

We will now create the top-level schematic for our mixed-design project. Select **File | New** and specify **Block Diagram/Schematic File** as the type of file to create. To save the file, select **File | Save As** and browse to the directory *tutorial1\designstyle3*. It is necessary to browse back to our *designstyle3* directory because Quartus II always remembers the last directory that has been accessed; in the preceding step we had created the *example_schematic.bsf* symbol file in the *designstyle1* directory. Use the name *example_mixed1.bdf* when saving the top-level file.

To import the *verfunctions* and *example_schematic* symbols, double-click on the Block Editor screen, or select **Edit | Insert Symbol**. This command opens the window in Figure B.30. Click on the + next to the label **Project** on the top-left of the figure, and then click on the item *verfunctions* to select this symbol. Click **OK** to import the symbol into the schematic. Next, we need to import the *example_schematic* subcircuit. Since this symbol is stored in the *designstyle1* project directory, it is not listed under the **Project** label in Figure B.30. To find the symbol, browse on the **Name:** box in the figure. Locate *example_schematic.bsf* in the *tutorial1\designstyle1* directory and perform the import operation. Finally, import the input and output symbols from the primitives library and make the wiring connections, as explained in section B.3, to obtain the final circuit depicted in Figure B.31.

Compile the schematic. If Quartus II produces an error saying that it cannot find the schematic file *example_schematic.bdf*, then you need to tell Quartus II where to look for this file. Select **Assignments | Settings** to open the Settings window, which was displayed in Figure B.24. On the left side of this window, click on **User Libraries**, and then in the **Library name** box browse to find the directory *tutorial1\designstyle1*. Click **Open** to add this directory into the **Libraries** box of the Settings window. Finally, click **OK** to close the Settings window and then try again to compile the project.

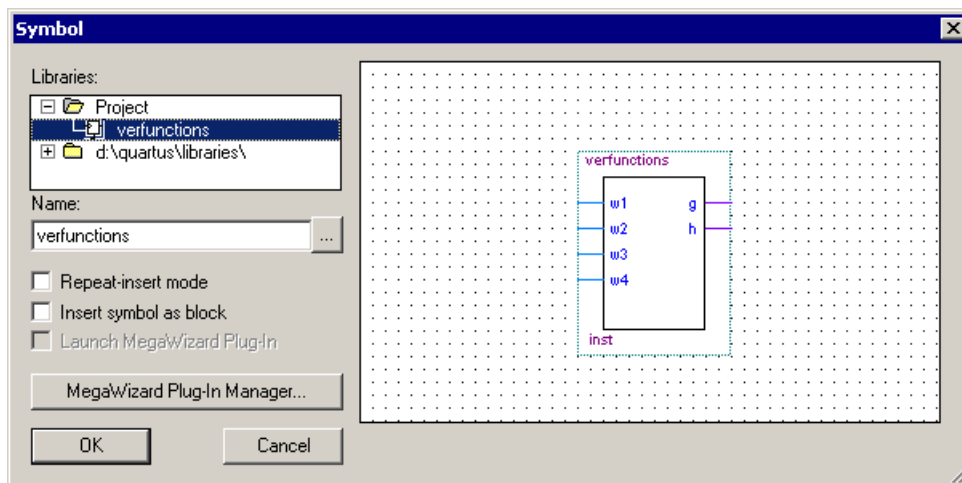


Figure B.30. Importing the symbol for the *verfunctions* subcircuit.

To verify its correctness, the circuit has to be simulated. This circuit has five inputs, so there are 32 possible input valuations that could be tested. Instead, we will randomly choose just six valuations, as shown in Figure B.32, and perform the simulation. The correct values of f which are produced by the simulator are shown in the figure. (Chapter 11 deals with the testing issues in detail and explains that using

a relatively small number of randomly-chosen input test vectors is a reasonable approach.)

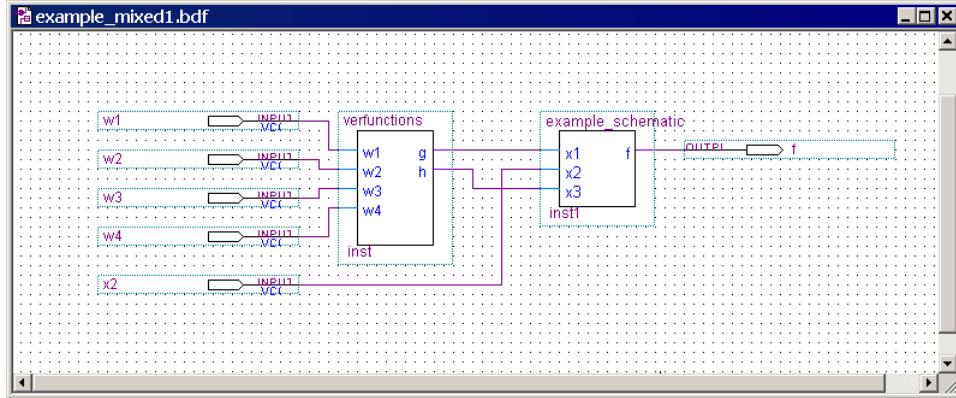


Figure B.31. The complete circuit.

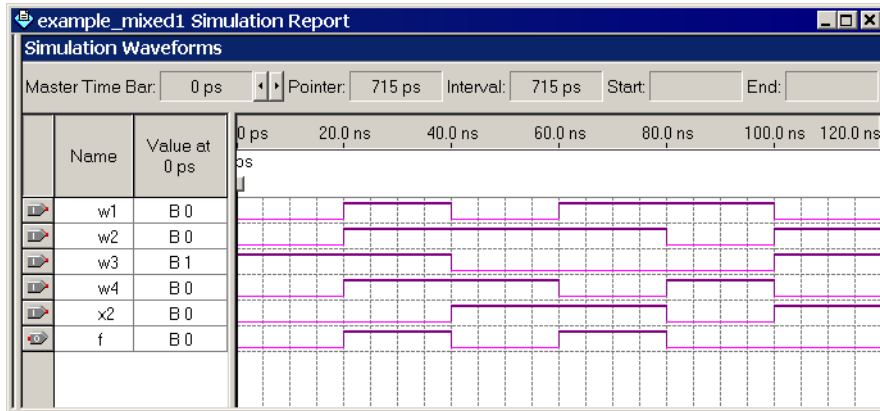


Figure B.32. Simulation results for the *example_mixed1* circuit.

About Errors

Quartus II displays messages produced during compilation in the Messages window. This window is at the bottom of the Quartus II display in Figure B.1. If the schematic is drawn correctly, one of the messages will state that the compilation was successful and that there are no errors or warnings.

To see what happens if an error is made, remove the wire that connects input $x3$ to the bottom AND gate and compile the modified schematic. Now, the compilation is not successful and two error messages are displayed. The first tells the designer that the affected AND gate is missing a source. The second states that there is one error and one warning. In a large circuit it may be difficult to find the location of an error. Quartus II provides help whereby if the user double-clicks on the error message, the corresponding location (AND gate in our case) will be highlighted. Reconnect the removed wire and recompile the corrected circuit.

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.

9.1 JTAG Programming

The programming and configuration task is performed as follows. Flip the RUN/PROG switch into 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.

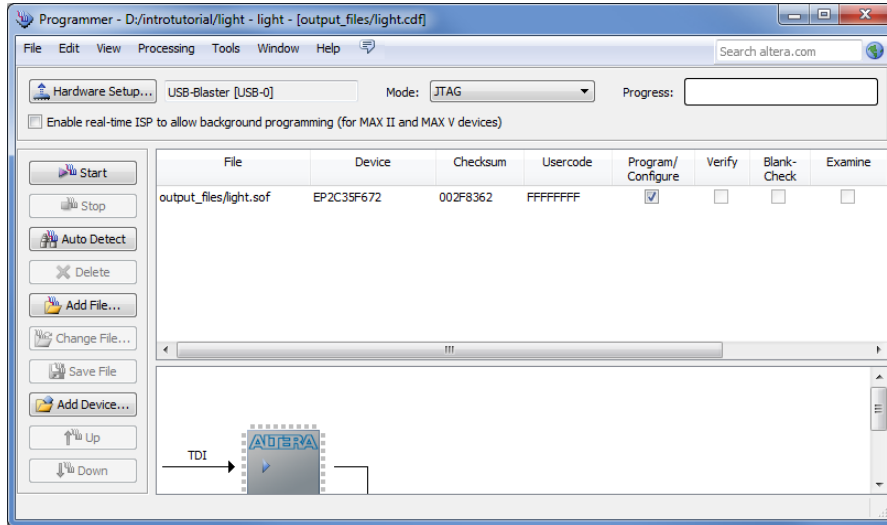


Figure 36. The Programmer window.

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 EP2C35F672, which is the FPGA device used on the DE2 board. Click on the **Program/Configure** check box, as shown in Figure 38.

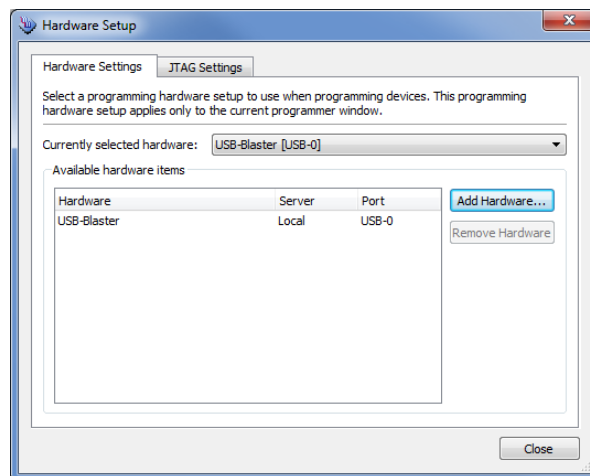


Figure 37. The Hardware Setup window.

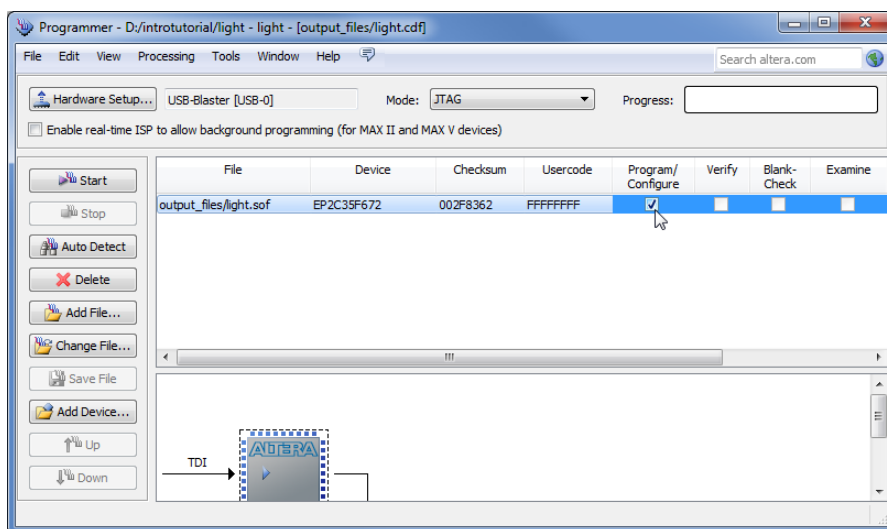


Figure 38. The updated Programmer window.

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 Active Serial Mode Programming

In this case, the configuration data has to be loaded into the configuration device on the DE-series board. Refer to Table 3 for a list of configuration devices on DE-series boards. To specify the required configuration device select Assignments > Device, which leads to the window in Figure 39.

Click on the Device and Pin Options button to reach the window in Figure 40. Now, click on the Configuration tab to obtain the window in Figure 41. In the Configuration device box (which may be set to Auto) choose the correct configuration device name and click OK. Upon returning to the window in Figure 39, click OK. Recompile the designed circuit.

Board	Configuration Device
DE0	EPCS4
DE0-Nano	EPCS64
DE1	EPCS4
DE2	EPCS16
DE2-70	EPCS64
DE2-115	EPCS64

Table 3. DE-series Configuration Device Names

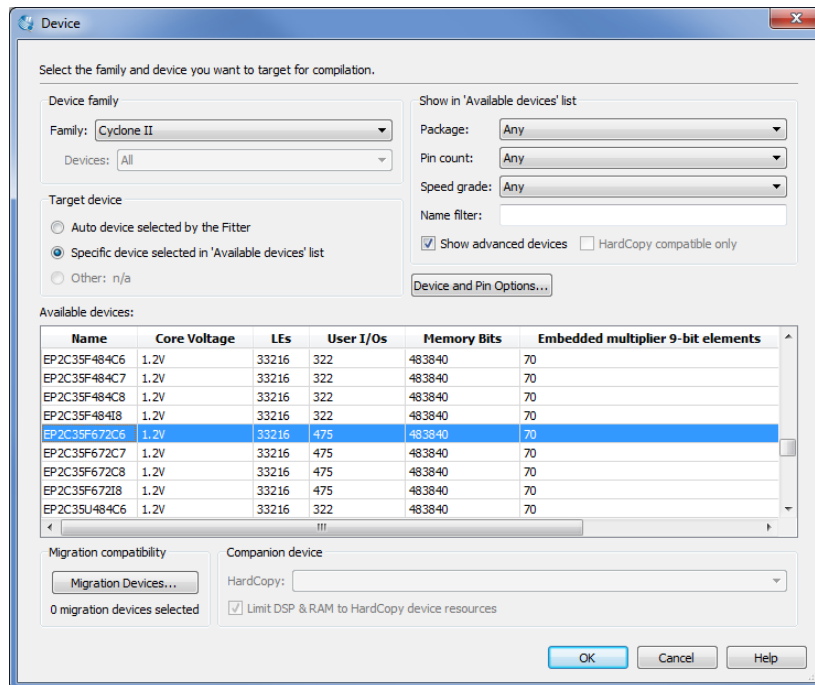


Figure 39. The Device Settings window.

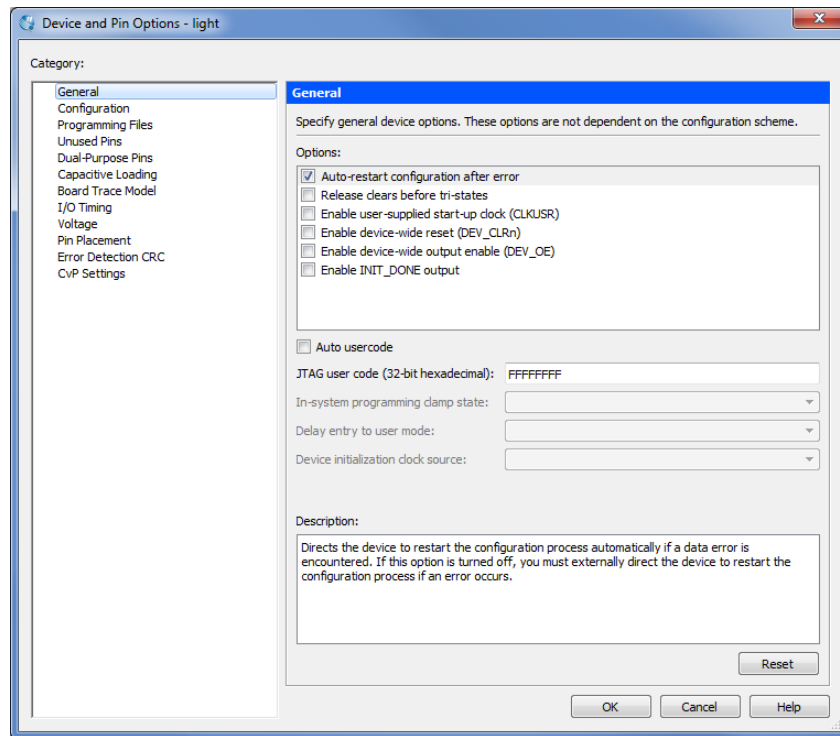


Figure 40. The Options window.

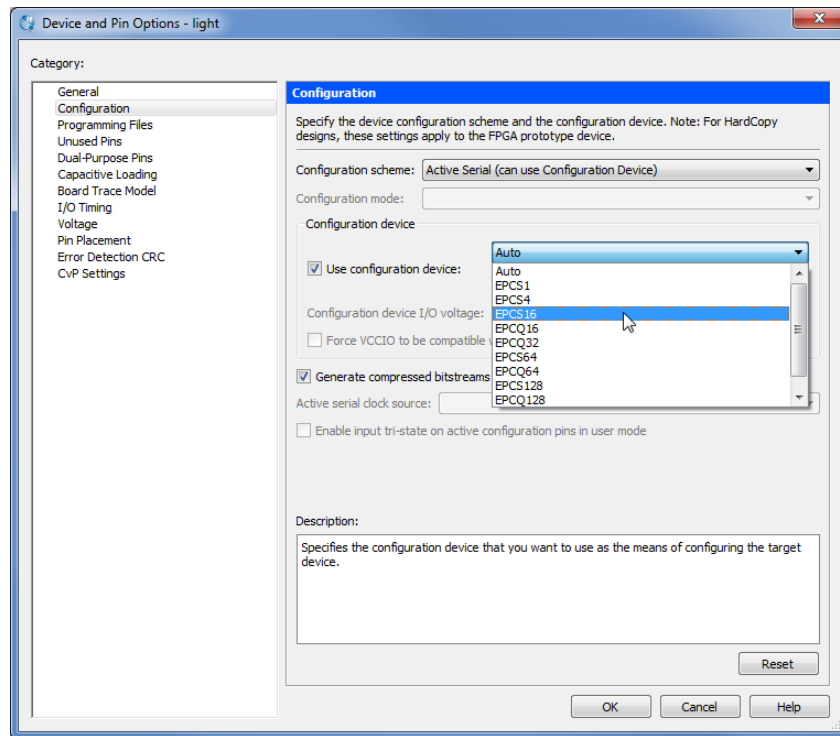


Figure 41. Specifying the configuration device.

The rest of the procedure is similar to the one described above for the JTAG mode. Select **Tools > Programmer** to reach the window in Figure 36. In the Mode box select **Active Serial Programming**. If you are changing the mode from the previously used JTAG mode, the pop-up box in Figure 42 will appear, asking if you want to clear all devices. Click **Yes**. Now, the Programmer window shown in Figure 43 will appear. Make sure that the Hardware Setup indicates the USB-Blaster. If the configuration file is not already listed in the window, press **Add File**. The pop-up box in Figure 44 will appear. Select the file *lab1.pof* in the directory *introtutorial/output_files* and click **Open**. As a result, the configuration file *light.pof* will be listed in the window. This is a binary file produced by the Compiler's Assembler module, which contains the data to be loaded into the configuration device on the DE-series board. The extension *.pof* stands for Programmer Object File. Upon returning to the Programmer window, click on the **Program/Configure** check box, as shown in Figure 45.

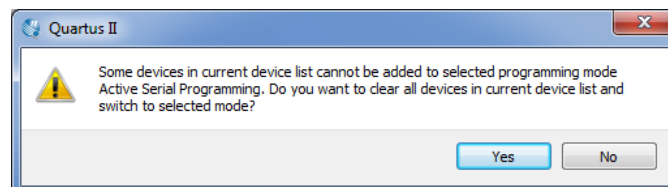


Figure 42. Clear the previously selected devices.

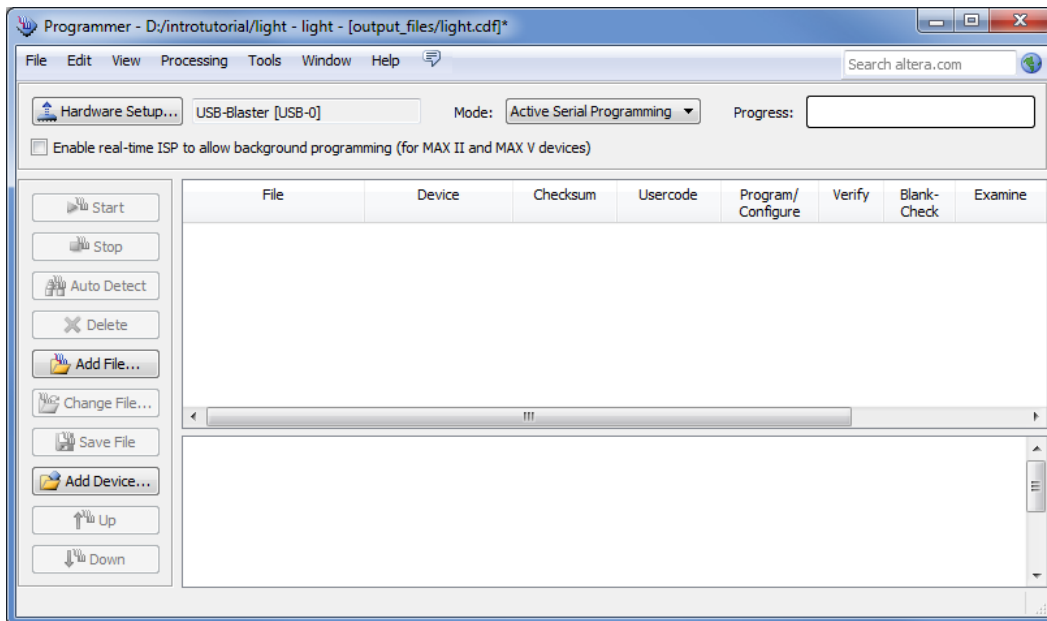


Figure 43. The Programmer window with Active Serial Programming selected.

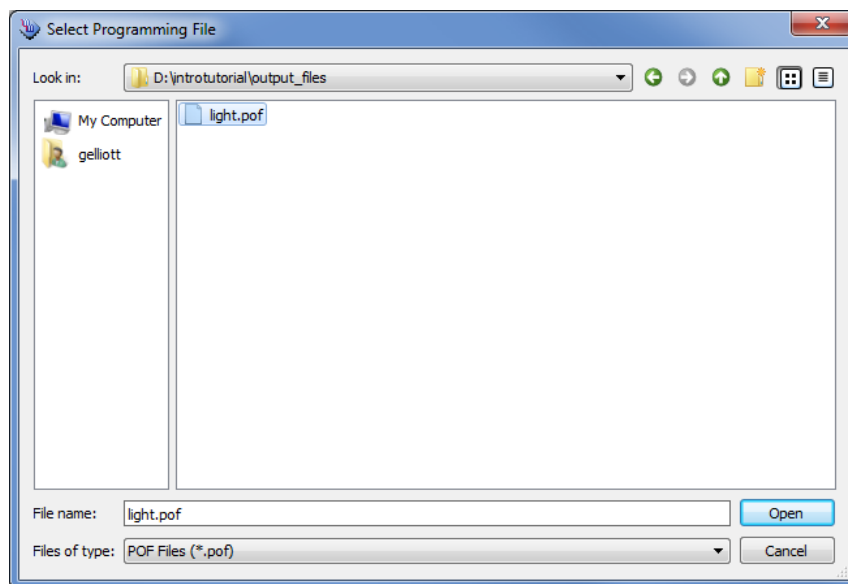


Figure 44. Choose the configuration file.

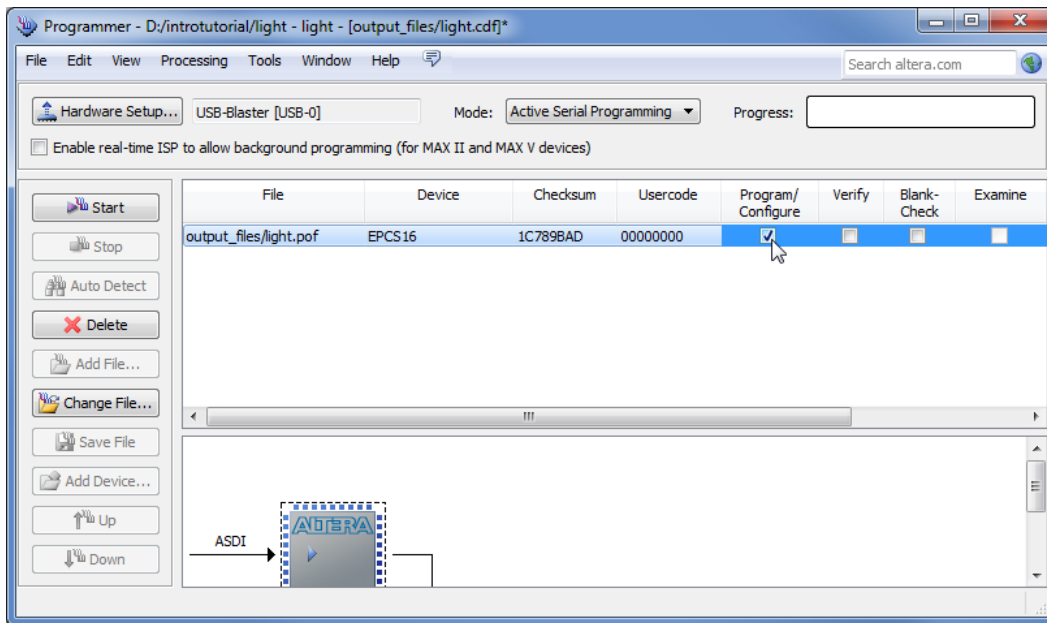


Figure 45. The updated Programmer window.

Flip the RUN/PROG switch on the DE-series board to the PROG position. Press Start in the window in Figure 45. An LED on the board will light up when the configuration data has been downloaded successfully. Also, the Progress box in Figure 45 will indicate when the configuration and programming process is completed, as shown in Figure 46.

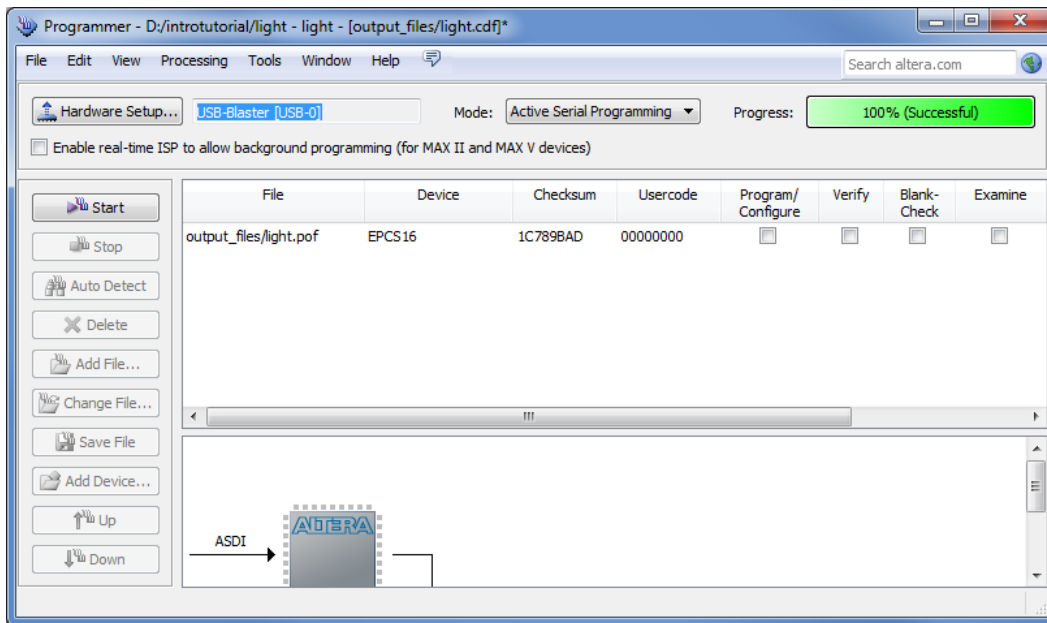


Figure 46. The Programmer window upon completion of programming.

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.

Copyright ©1991-2013 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera 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 Altera Corporation. Altera 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.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.