

Appendix B

Tutorial 1 — Using Quartus II CAD Software

Quartus II is a sophisticated CAD system. As most commercial CAD tools are continuously being improved and updated, Quartus II has gone through a number of releases. The version known as Quartus II 4.0 is provided with this book on a CD-ROM. For simplicity, in our discussion we will refer to this software package simply as Quartus II.

In this tutorial we introduce the design of logic circuits using Quartus II. Step-by-step instructions are presented for performing design entry with two methods: using schematic capture and writing Verilog code, as well as with a combination of the two. The tutorial also illustrates the process of simulation.

B.1 Introduction

This tutorial assumes that the reader has access to a computer on which Quartus II is installed. Instructions for installing Quartus II are provided with the software. The Quartus II software will run on several different types of computer systems. For this tutorial a computer running a Microsoft operating system (Windows NT, Windows 2000, or Windows XP) is assumed. Although Quartus II operates similarly on all of the supported types of computers, there are some minor differences. A reader who is not using a Microsoft Windows operating system may experience some slight discrepancies from this tutorial. Examples of potential differences are the locations of files in the computer's file system and the exact appearance of windows displayed by the software. All such discrepancies are minor and will not affect the reader's ability to follow the tutorial.

This tutorial does not describe how to use the operating system provided on the computer. We assume that the reader already knows how to perform actions such as running programs, operating a mouse, moving, resizing, minimizing and maximizing windows, creating directories (folders) and files, and the like. A reader who is not familiar with these procedures will need to learn how to use the computer's operating system before proceeding.

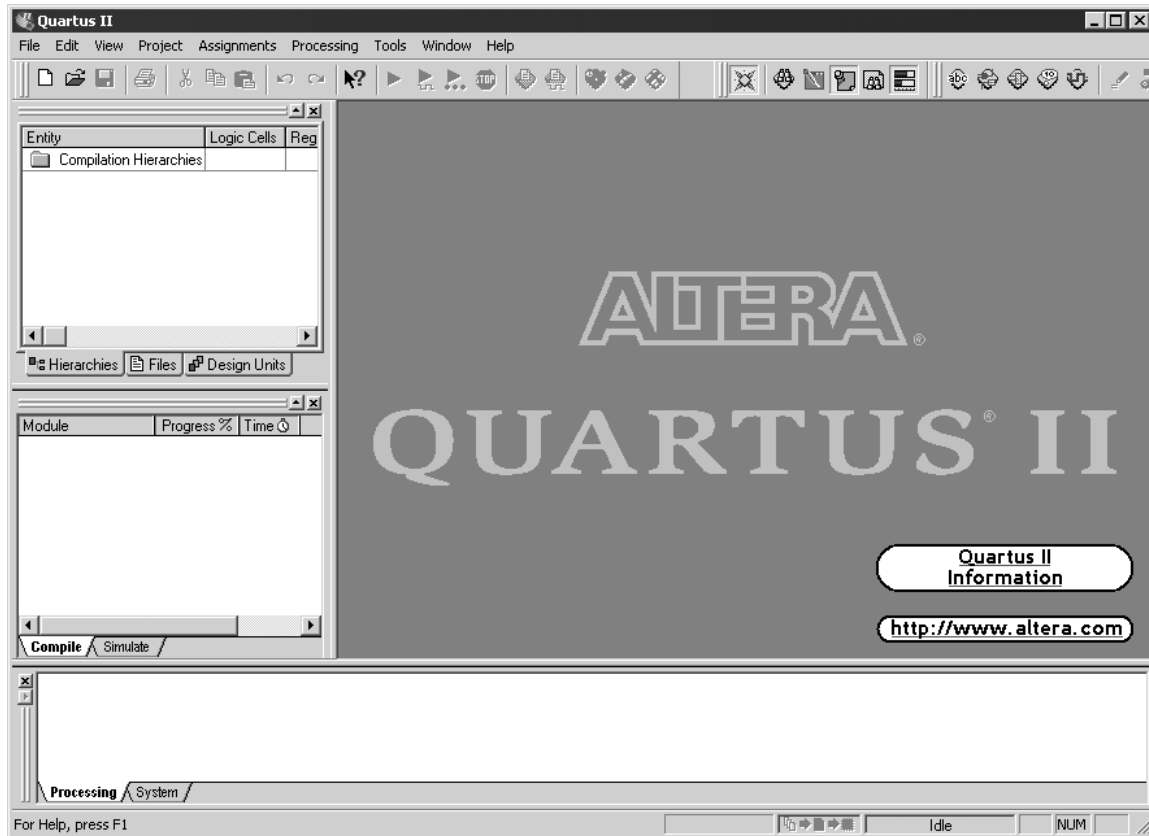


Figure B.1. The main Quartus II display.

B.1.1 Getting Started

Each logic circuit, or subcircuit, being designed in Quartus II is called a *project*. The software works on one project at a time and keeps all information for that project in a single directory in the file system (we use the traditional term *directory* for a location in the file system, but in Microsoft Windows the term *folder* is used). To begin a new logic circuit design, the first step is to create a directory to hold its files. As part of the installation of the Quartus II software, a few sample projects are placed into a directory called *qdesigns*. To hold the design files for this tutorial, we will use a directory *tutorial1*. The location and name of the directory is not important; hence the reader may use any valid directory.

Start the Quartus II software. You should see a display similar to the one in Figure B.1. This display consists of several windows that provide access to all features of Quartus II, which the user selects with the computer mouse.

Most of the commands provided by Quartus II can be accessed by using a set of menus that are located below the title bar. For example, in Figure B.1 clicking the left mouse button on the menu named **File** opens the menu shown in Figure B.2. Clicking the left mouse button on the item **Exit** exits from Quartus II. In general, whenever the mouse is employed 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. 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 Quartus II system. Many Quartus II commands have an associated icon displayed in one of the toolbars. To see the list of available toolbars, select Tools | Customize | Toolbars. Once a toolbar is opened, it can be moved with the mouse, and icons can be dragged from one toolbar to another. To see the Quartus II command associated with an icon, position the mouse cursor on top of the icon and a tooltip will appear that displays the command name.

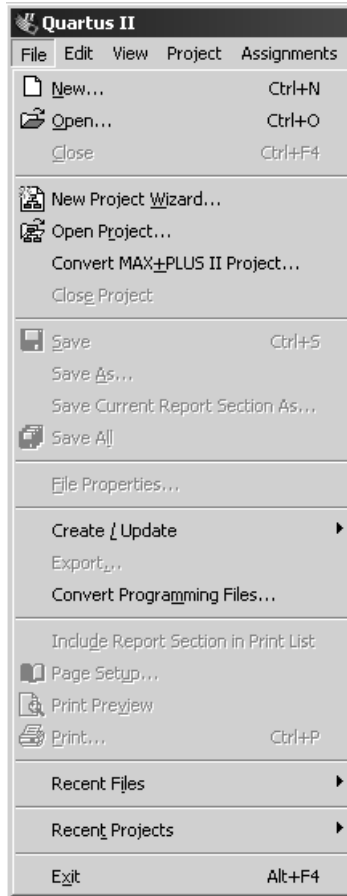


Figure B.2. An example of the File menu.

It is possible to modify the appearance of the Quartus II display in Figure B.1 in many ways. Section B.6 shows how to move, resize, close, and open windows within the main Quartus II display.

Quartus II On-Line Help

Quartus II provides comprehensive on-line documentation that answers many of the questions that may arise when using the software. The documentation is accessed from the menu in the Help window. To get some idea of the extent of documentation provided, it is worthwhile for the reader to browse through the Help topics. For instance, selecting Help | How to Use Help gives an indication of what type of help is provided.

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

B.2 Starting a New Project

To start working on a new design we first have to define a new *design project*. Quartus II makes the designer's task easier by providing support in the form of a *wizard*. Select **File | New Project Wizard** to reach a window that indicates the capability of this wizard. Press **Next** to get the window shown in Figure B.3. Set the working directory to be *tutorial1\designstyle1*. The project must have a name, which may optionally be the same as the name of the directory. We have chosen the name *example_schematic* because our first example involves design entry by means of schematic capture. Observe that Quartus II automatically suggests that the name *example_schematic* be also the name of the top-level design entity in the project. This is a reasonable suggestion, but it can be ignored if the user wants to use a different name. Press **Next**. Since we have not yet created the directory *tutorial1\designstyle1*, Quartus II displays the pop-up box in Figure B.4 asking if it should create the desired directory. Click **Yes**, which leads to the window in Figure B.5. In this window the designer can specify which existing files (if any) should be included in the project. We have no existing files, so click **Next**.

Now, the window in Figure B.6 appears, which allows the designer to specify third-party CAD tools (i.e. those that are not a part of Quartus II software) that should be used. In this book, we have used the term CAD tools to refer to software packages developed for use in computer aided design tasks. Another term for software of this type 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, we will not choose any other tools.

Press **Next** to go to the window shown in Figure B.7. Here, we can specify the type of device in which the designed circuit will be implemented. For the purpose of this tutorial the choice of device is unimportant. Choose the device family called Cyclone, which is a type of FPGA that we will use in Appendix C. We do not need to choose a specific device, so click on the selection **No, I want to allow the Compiler to choose a device**. Press **Finish**, which returns to the main Quartus II display in Figure B.1, but with *example_schematic* specified as the new project.

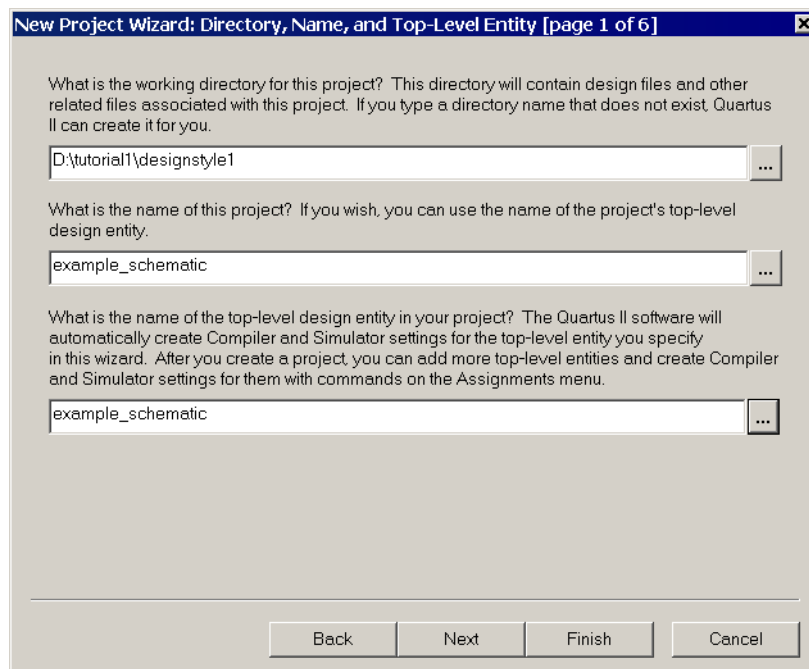


Figure B.3. Specifying the project directory and name.



Figure B.4. Quartus II can create the desired directory.

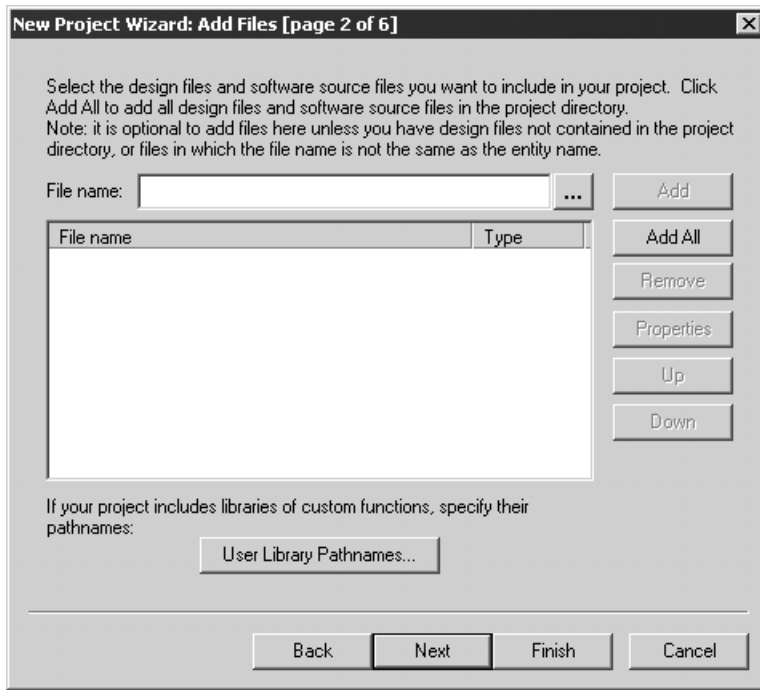


Figure B.5. A window for inclusion of design files.

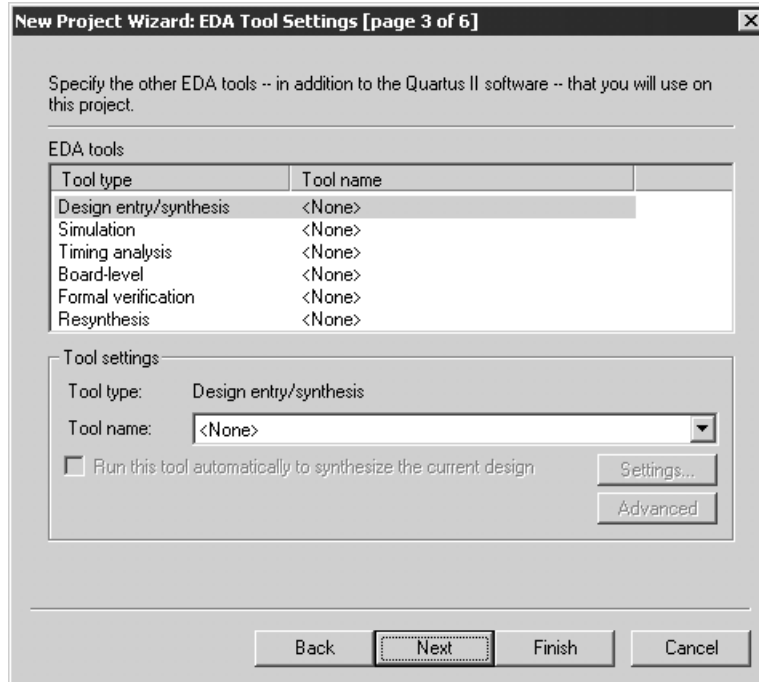


Figure B.6. Inclusion of other EDA tools.

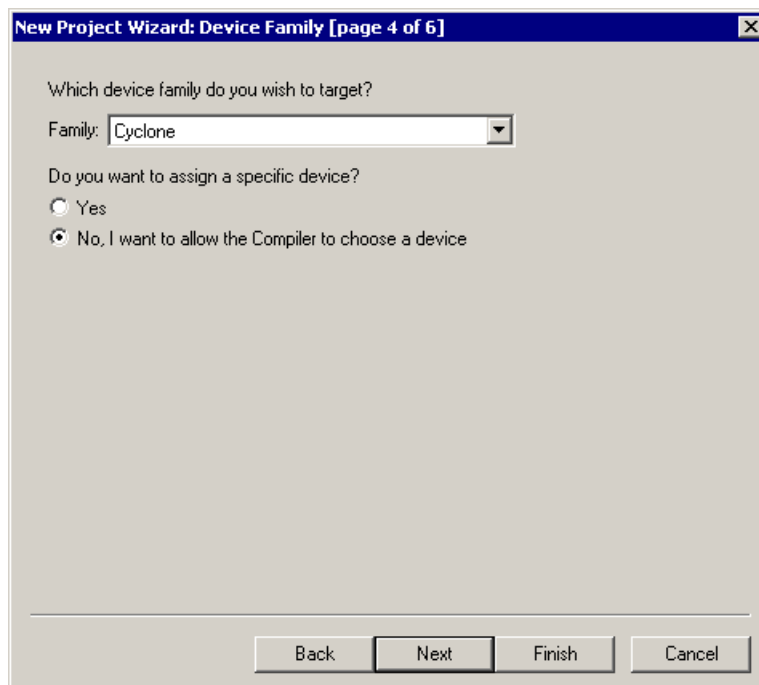


Figure B.7. Specification of the device family.

B.3 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 $f = x_1x_2 + \bar{x}_2x_3$. 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.

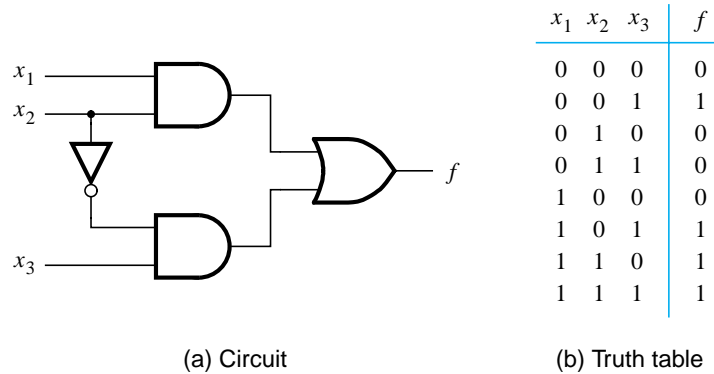


Figure B.8. The logic function of Figure 2.30.

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.

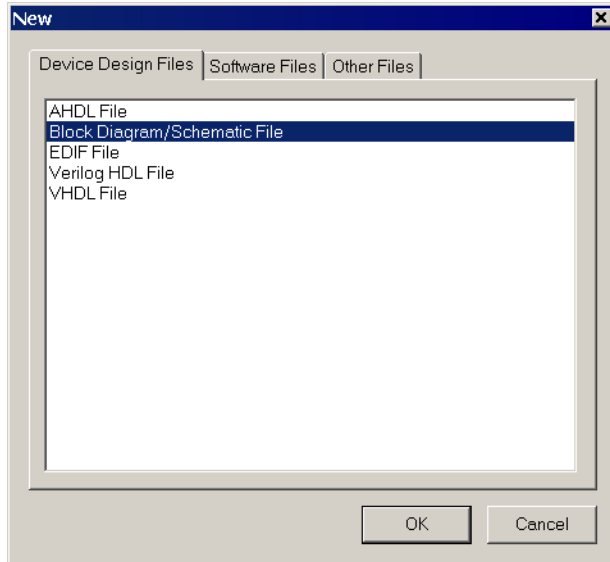


Figure B.9. Choosing the type of design file.

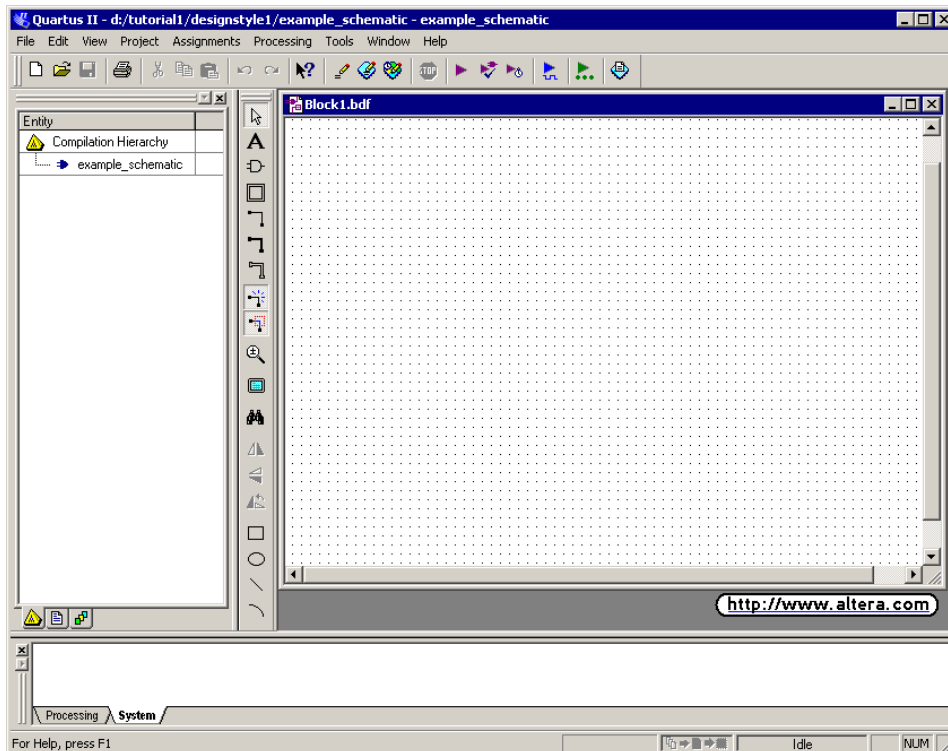


Figure B.10. Block Editor window.

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:**

quartus

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**.

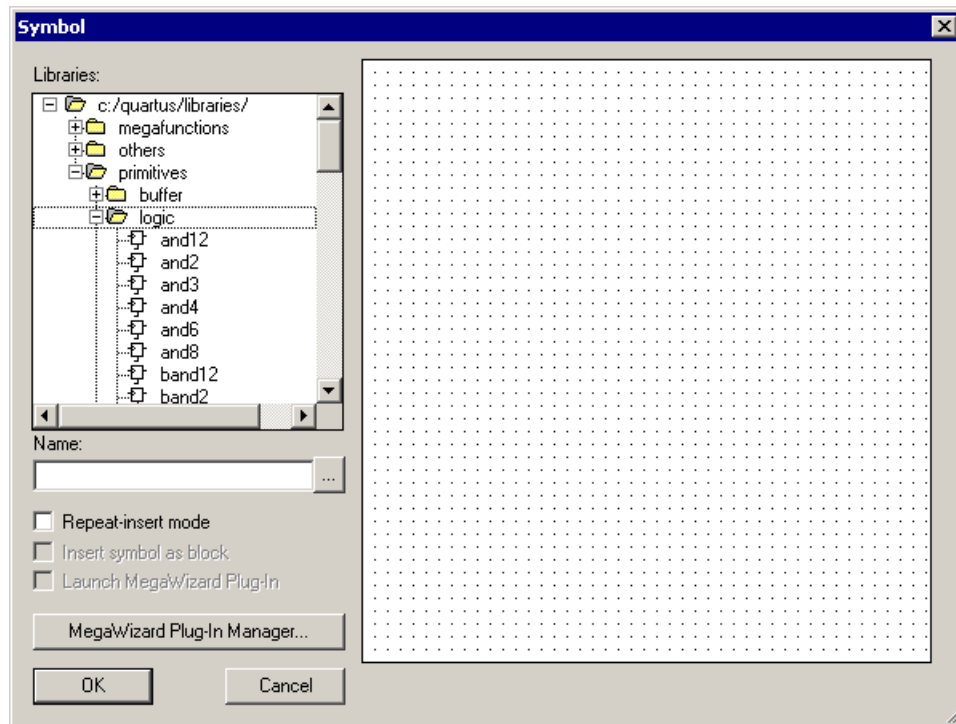


Figure B.11. Selection of logic symbols.

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.

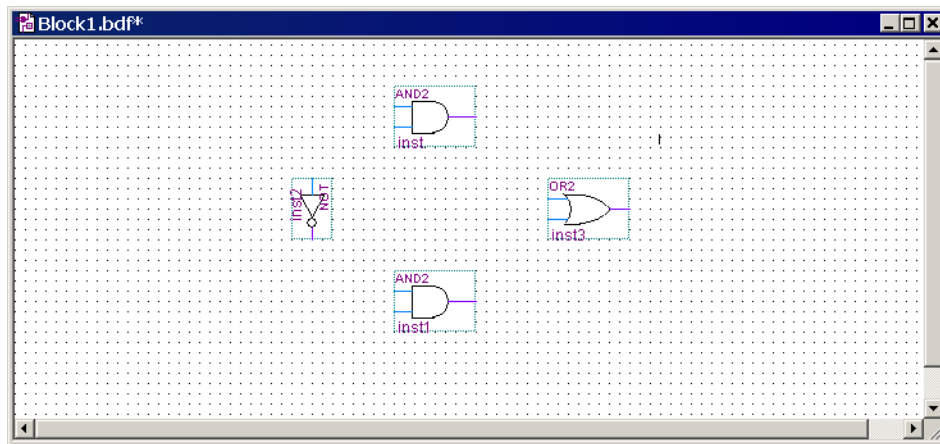


Figure B.12. Imported gate symbols.

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.

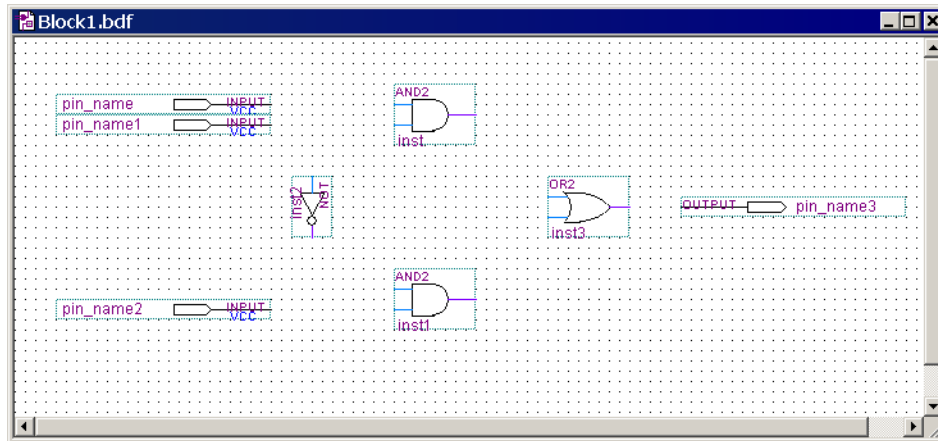


Figure B.13. The desired arrangement of gates and pins.

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.

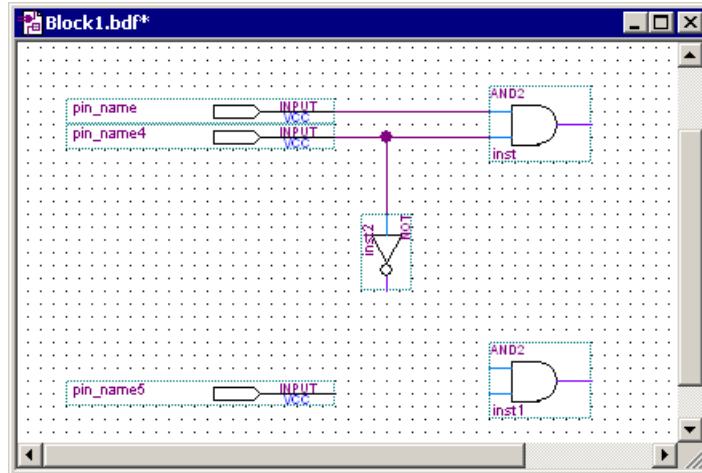


Figure B.14. Expanded view of the circuit.

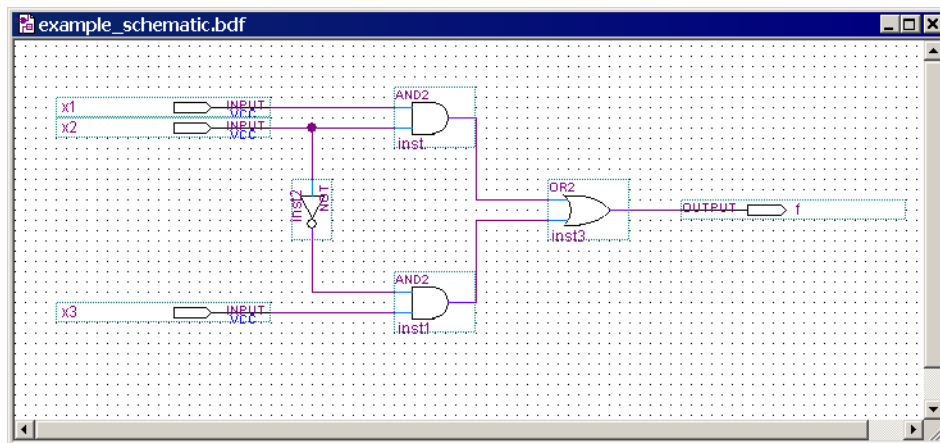


Figure B.15. The completed schematic.

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.

Using the Compiler

The CAD tools available in Quartus II are divided into a number of modules. Select **Tools | Compiler Tool** to open the window in Figure B.16, which lists five of the main modules. The **Analysis & Synthesis** module performs the synthesis step in Quartus II. It produces a circuit of logic elements, where each element can be directly implemented in the target chip. The **Fitter** module determines the exact location on the chip where each of these elements produced by synthesis will be implemented. A detailed discussion of CAD modules is provided in Chapter 12.

These Quartus II modules are controlled by an application program called the Compiler. The Compiler can be used to run a single module at a time, or it can invoke multiple modules in sequence. There are several ways to access the Compiler in the Quartus II user interface. In Figure B.16 clicking on the leftmost button under **Analysis & Synthesis** will run this module. Similarly, the **Fitter** module can be executed by clicking its leftmost button in the figure. Pressing the **Start Compilation** button runs the modules in Figure B.16 in sequence.

Another convenient way of accessing the Compiler is to use the **Processing | Start** menu. The command for running the synthesis module is **Processing | Start | Start Analysis & Synthesis**. Part of the synthesis module can also be invoked by using the command **Processing | Start | Start Analysis & Elaboration**. This command runs only the early part of synthesis, which checks the design project for syntax errors, and identifies the major subdesign names that are present in the project. The command **Processing | Start Compilation** is equivalent to pressing the **Start Compilation** button in Figure B.16. There is also a toolbar icon for this command, which looks like a purple triangle.

An efficient way of using the CAD tools is to run only the modules that are needed at any particular phase of the design process. This approach is pragmatic because some of the CAD tools may require a long time, on the order of hours, to complete when processing a large design project. For the purpose of this tutorial, we wish to perform functional simulation of our schematic. Since only the output of synthesis is needed to perform this task, we will run only the synthesis module.

Select **Processing | Start | Start Analysis & Synthesis**, use the corresponding icon in the toolbar, or use the shortcut **Ctrl-k**. As the compilation proceeds, its progress is reported in the lower-right corner of the Quartus II display, and also in the Status utility window on the left side (if this window is not open it can be accessed by selecting **View | Utility Windows | Status**). Successful (or unsuccessful) compilation is indicated in a pop-up box. Acknowledge it by clicking **OK** and examine the compilation report depicted in Figure B.17 (if the report is not already opened, it can be accessed by clicking on the **Report** icon in the Compiler Tool window, using the appropriate icon in the toolbar, or by selecting **Processing | Compilation Report**). The report summary shows that our small design would use only four pins and one logic element in a Cyclone FPGA.

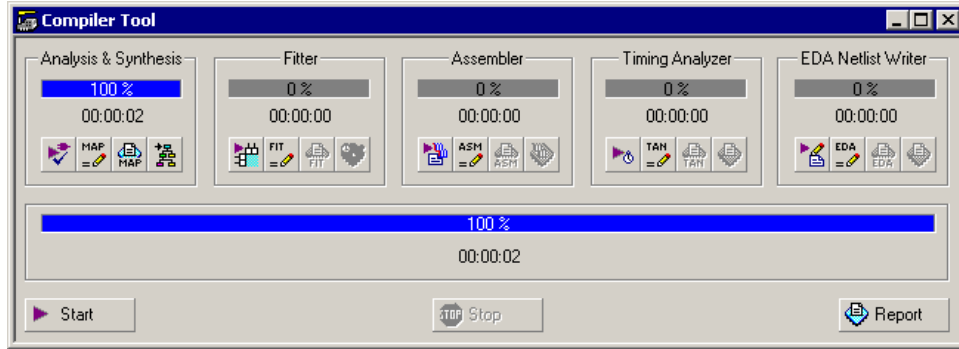


Figure B.16. The Compiler Tool window.

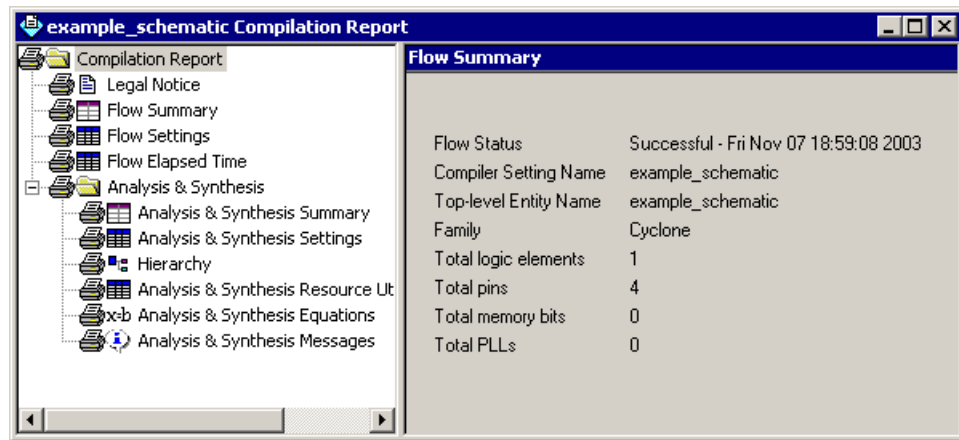


Figure B.17. The compilation report summary.

The compilation report provides a lot of information that may be of interest to the designer. For example, the detailed implementation in the form of synthesized logic expressions can be seen by clicking on the small + symbol next to Analysis & Synthesis in the compilation report, and selecting Analysis & Synthesis Equations. The equation that Quartus II used to implement our circuit is

$$f = x_1(x_3 + x_2) + \bar{x}_1x_3\bar{x}_2$$

The report denotes AND as &, OR as #, and NOT as !. This is not the simplest expression that one may expect, namely

$$f = x_1x_2 + \bar{x}_2x_3$$

But both expressions represent the same function and the CAD tools do not always display the simplest form of the equations in the compilation report. The compilation report can be opened at any time by selecting Processing | Compilation Report or by clicking on the corresponding toolbar icon which looks like a white sheet on top of a blue chip.

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.

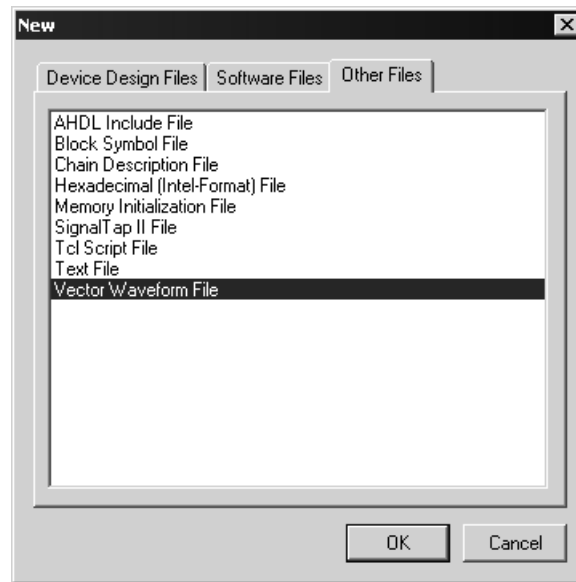


Figure B.18. Choose to prepare a test-vector file.

B.3.3 Simulating the Designed Circuit

Quartus II includes a simulation tool that can be used to simulate the behavior of the designed circuit. Before the circuit can be simulated, it is necessary to create the desired waveforms, called *test vectors*, to represent the input signals. We will use the Quartus II Waveform Editor to draw test vectors.

Using the Waveform Editor

Open the Waveform Editor window by selecting **File | New**, which gives the window in Figure B.9. Click on the **Other Files** tab to reach the window displayed in Figure B.18. Choose **Vector Waveform File** and click **OK**.

The Waveform Editor window is depicted in Figure B.19. Save the file under the name *example_schematic.vwf*, and note that this changes the name in the displayed window. Set the desired simulation to run from 0 to 160 ns by selecting **Edit | End Time** and entering 160 ns in the dialog box that pops up. Select **View | Fit in Window** to display the entire simulation range of 0 to 160 ns in the window. You may want to resize the window to its maximum size.

Next, we want to include the input and output nodes of the circuit to be simulated. This is done by using the Node Finder utility. Click **Edit | Insert Node or Bus** to open the window in Figure B.20. It is possible to type the name of a signal (pin) into the **Name** box, but it is more convenient to click on the button labeled **Node Finder** to open the window in Figure B.21. 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.

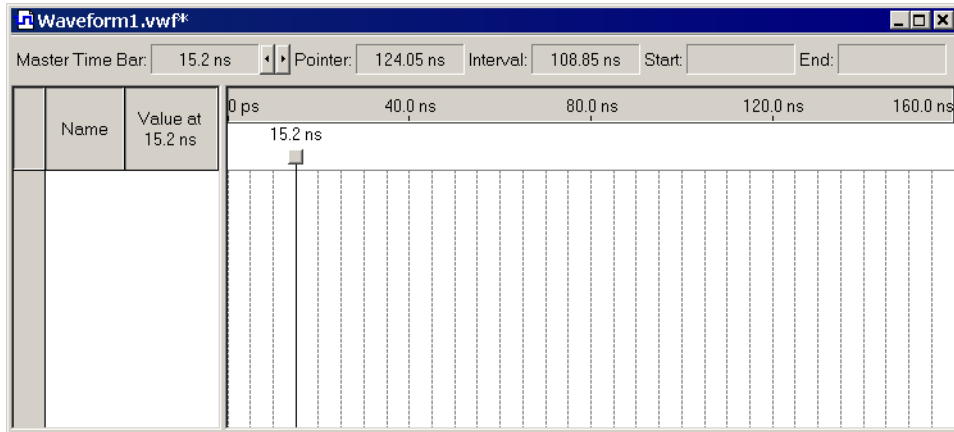


Figure B.19. The Waveform Editor window.

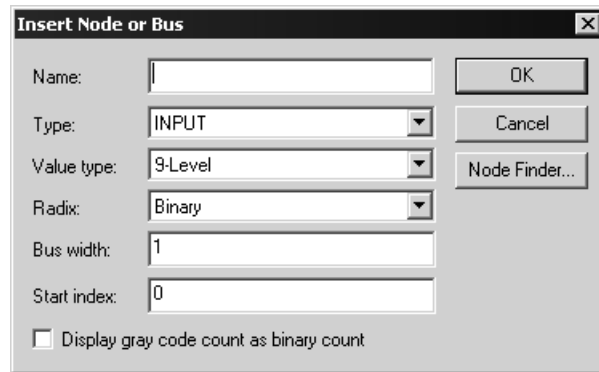


Figure B.20. The Insert Node or Bus dialogue.

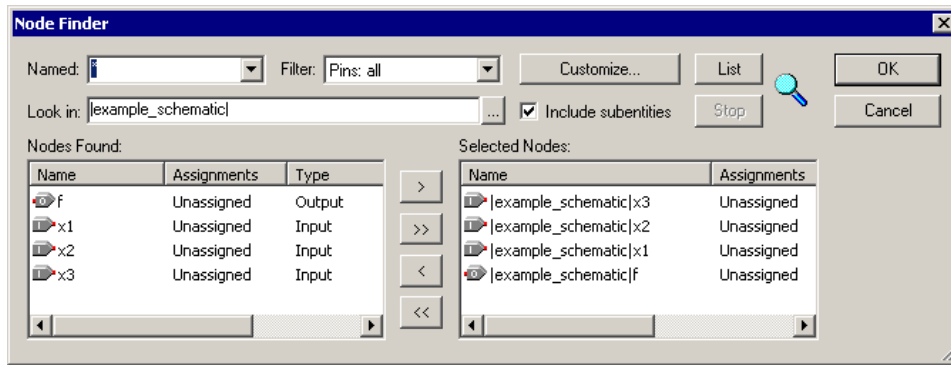


Figure B.21. The Node Finder window.

The Node Finder displays on the left side of the window the nodes f , $x1$, $x2$, and $x3$. Click on $x3$ and then click the $>$ sign to add it to the Selected Nodes box on the right side of the figure. Do the same for $x2$, $x1$, and f . Click OK to close the Node Finder window, and then click OK in the window of Figure B.20. This leaves a fully displayed Waveform Editor window, as shown in Figure B.22. If you did not select the nodes in the same order as displayed in Figure B.22, it is possible to rearrange them. To move a waveform up or down in the Waveform Editor window, click on the node name (in the Name column) and release the mouse button. The waveform is now highlighted to show the selection. Click again on the waveform and drag it up or down in the Waveform Editor.

We will now specify the logic values to be used for the input signals 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, Quartus II displays (by default) the vertical guidelines and provides a drawing feature that snaps on these lines (which can otherwise be invoked by choosing View | Snap to Grid). Observe also a solid vertical line, which can be moved by pointing to its top and dragging it horizontally. We will use this “reference line” in Tutorial 2. The waveforms can be drawn using the Selection tool, which is activated by selecting the icon in the vertical toolbar that looks like a big arrowhead.

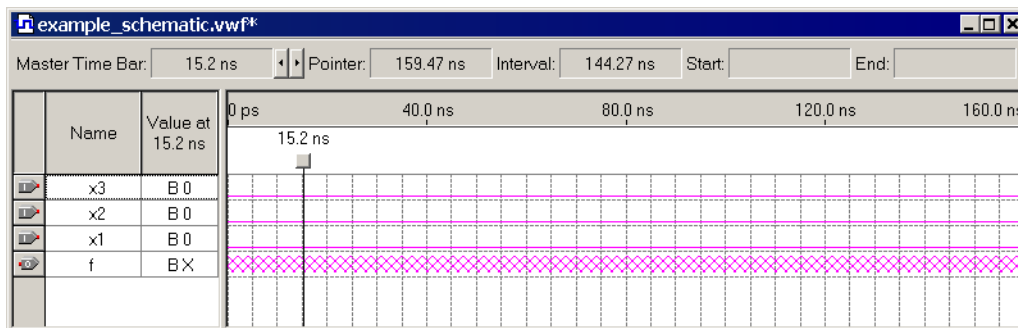


Figure B.22. The nodes needed for simulation.

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. The number of possible input valuations may be huge, so it is necessary to choose a relatively small (but representative) sample of these input valuations. (The topic of circuit testing is explored in Chapter 11.) Our circuit is very small, so it can be simulated fully by applying all eight possible valuations of inputs $x1$, $x2$, and $x3$. Let us apply a new valuation every 20 ns. To start,

all inputs are zero. At the 20-ns point we want x_3 to go to 1. Click on x_3 ; this highlights the signal and activates the vertical toolbar that allows us to shape the selected waveform. The toolbar provides options such as setting the signal to 0, 1, unknown (X), high impedance (Z), don't care (DC), and inverting its existing value (INV). Observe that the output f is displayed as having an unknown value at this time, which is indicated by a hashed pattern. A specific time interval is selected by pressing the mouse on a waveform at the start of the interval and dragging it to its end; the selected interval is highlighted. Select the interval from 20 to 40 ns for x_3 and set the signal to 1. Similarly, set x_3 to 1 from 60 to 80 ns, 100 to 120 ns, and 140 to 160 ns. Next, set x_2 to 1 from 40 to 80 ns, and from 120 to 160 ns. Finally, set x_1 to 1 from 80 to 160 ns. Complete the remaining assignments to obtain the image in Figure B.23 and save the file.

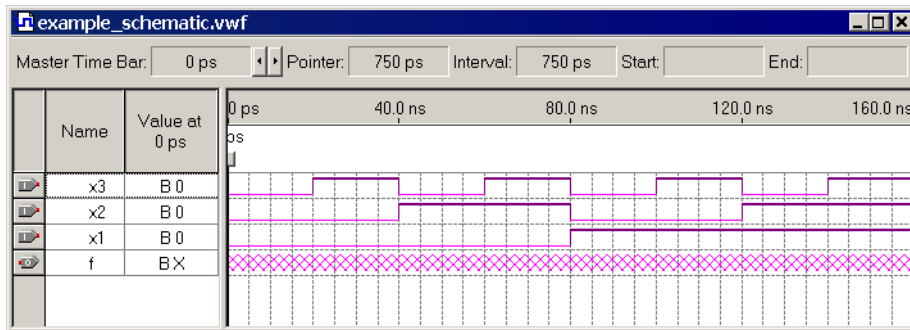


Figure B.23. The complete test vectors.

A convenient mechanism for changing the input waveforms is provided by the Waveform Editing tool. The icon for the tool is in the vertical toolbar; it looks like two arrows pointing left and right. When the mouse is dragged over some time interval in which the waveform is 0 (1), the waveform will be changed to 1 (0). Experiment with this feature on signal x_3 .

Performing the Simulation

As explained in Section 2.9.3, a circuit can be simulated in two ways. The simplest way is to assume that logic elements and interconnection wires 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. This takes much less time, because the simulation can be performed simply by using the logic expressions that define the circuit. In this tutorial we will use only the functional simulation. We will deal with the timing simulation in Appendix C.

To perform the functional simulation, select **Assignments | Settings** to open the Settings window. On the left side of this window click on **Simulator** to display the window in Figure B.24 and choose **Functional** as the simulation mode. To complete the set up of the simulator select the command **Processing | Generate Functional Simulation Netlist**. The Quartus II simulator takes the test inputs and generates the outputs defined in the *example_schematic.vwf* file. A simulation run is started by selecting **Processing | Start Simulation**, or by using the shortcut icon in the toolbar that looks like a blue triangle with a square wave below it. At the end of the simulation, Quartus II indicates its successful completion and displays a simulation report shown in Figure B.25. As seen in the figure, the Simulator creates a waveform for the output f . The reader should verify that the generated waveform corresponds to the truth table for f given in Figure B.8b.

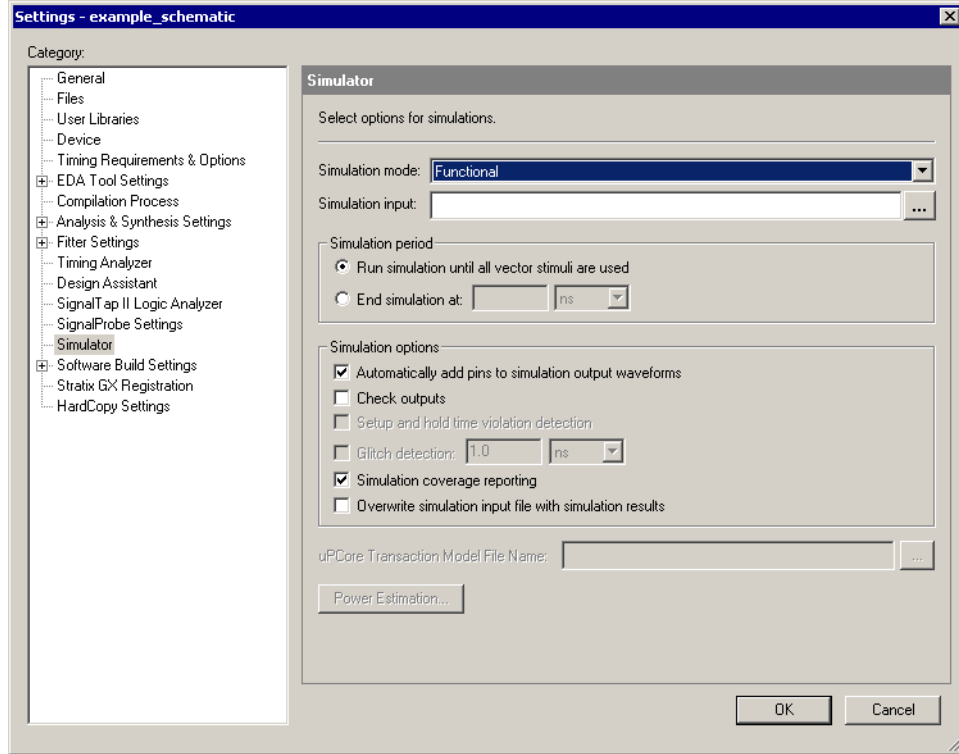


Figure B.24. Specifying the simulation mode.

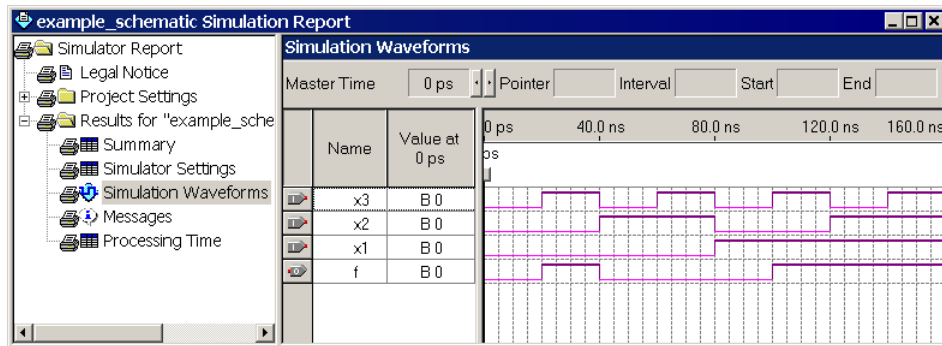


Figure B.25. The result of functional simulation.

We have now completed our introduction to design using schematic capture. Select **File | Close Project** to close the current project. Next, we will show how to use Quartus II to implement circuits specified in Verilog.

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.

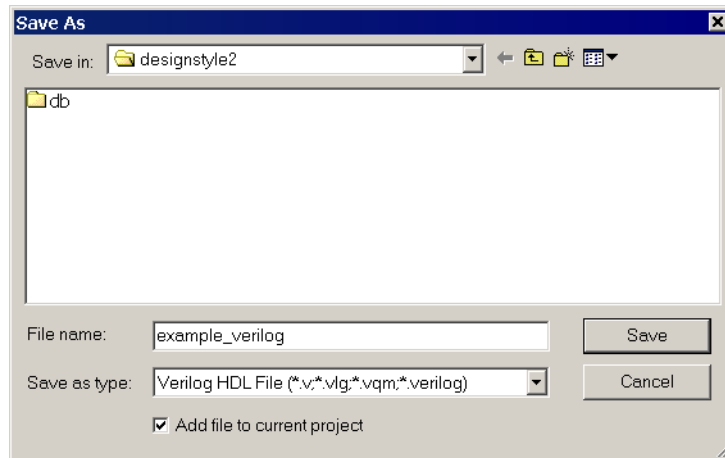
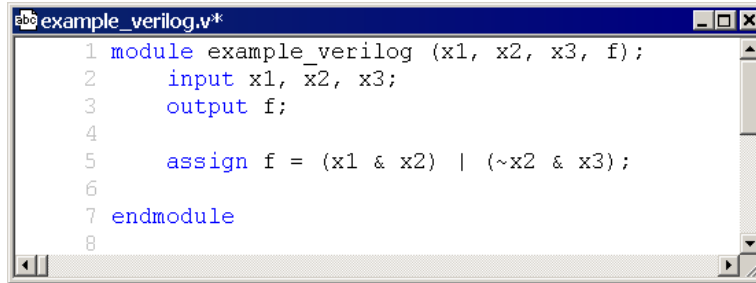


Figure B.26. Opening a new Verilog file.



```
1 module example_verilog (x1, x2, x3, f);
2     input x1, x2, x3;
3     output f;
4
5     assign f = (x1 & x2) | (~x2 & x3);
6
7 endmodule
8
```

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.

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.

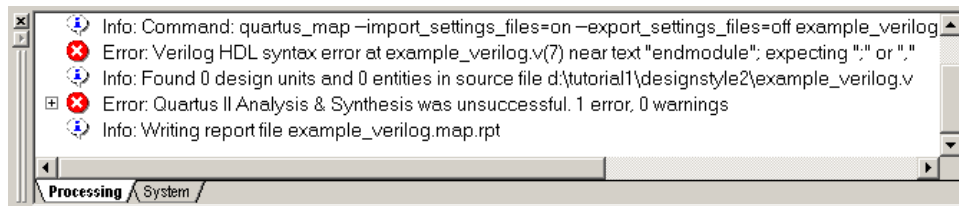


Figure B.28. The Message window.

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

$$f = x_1x_2 + \bar{x}_2x_3$$

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

$$f = x_1x_2 + \bar{x}_2x_3$$

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.

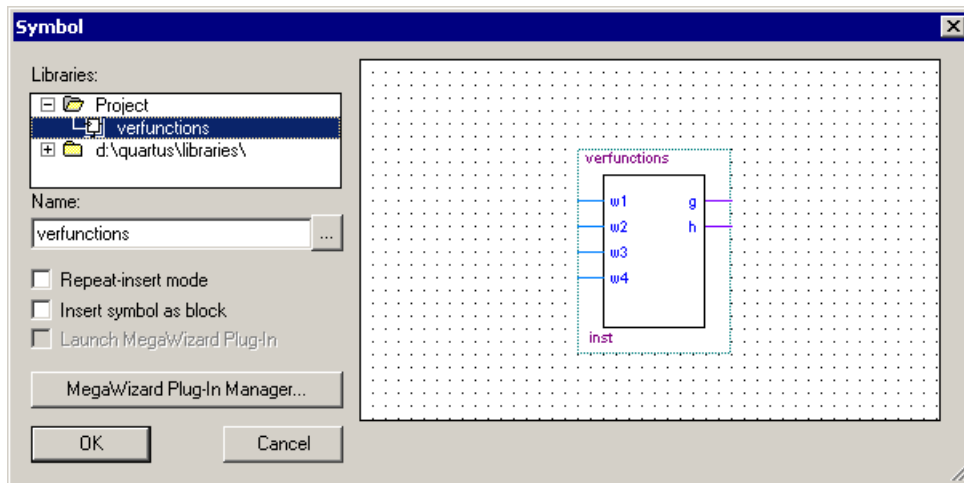


Figure B.30. Importing the symbol for the *verfunctions* subcircuit.

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.)

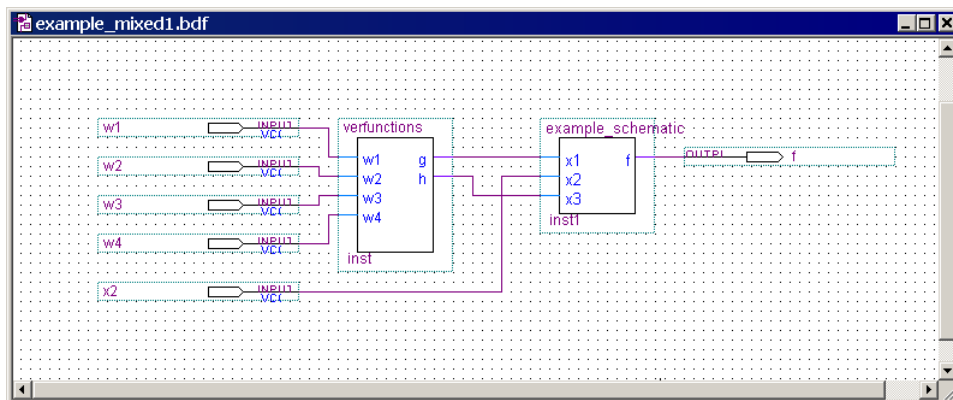


Figure B.31. The complete circuit.

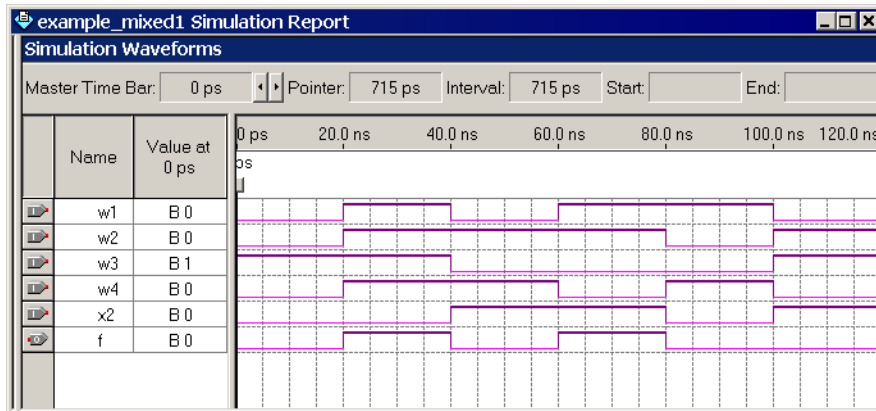


Figure B.32. Simulation results for the *example_mixed1* circuit.

B.5.2 Using Verilog at the Top Level

The previous example shows that a schematic can include a symbol which represents a Verilog module. In the alternative situation where Verilog is used for the top-level design file in a project, the user may wish to include a subcircuit that has been previously designed as a schematic. One way to do this is to use a software program that can translate the schematic into a Verilog file. Quartus II includes such a program, which is accessed under the File menu. To experiment with this feature, open the *example_schematic.bdf* file in the *designstyle1* project directory, and then select File | Create/Update | Create HDL Design for Current File. In the window that pops up, shown in Figure B.33, choose Verilog HDL as the type of source file to create, and click OK. Quartus II will generate the file *example_schematic.v*. Figure B.34 shows the contents of this file (in a slightly edited form to make it more compact). Note that Quartus II retained the original names of inputs x_1 , x_2 , and x_3 , and output f . It also chose some arbitrary names for the internal wires in the circuit.

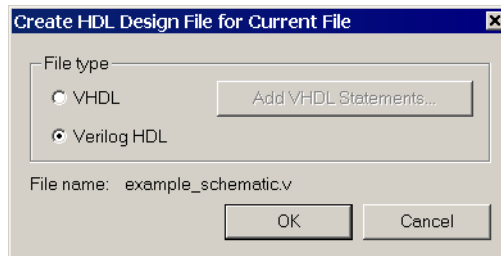


Figure B.33. Create a Verilog file for the schematic designed in Section B.3.

```

module example_schematic (x1, x2, x3, f);
  input x1;
  input x2;
  input x3;
  output f;
  wire SYNTHESIZED_WIRE_0;
  wire SYNTHESIZED_WIRE_1;
  wire SYNTHESIZED_WIRE_2;

  assign SYNTHESIZED_WIRE_2 = x1 & x2;
  assign SYNTHESIZED_WIRE_1 = SYNTHESIZED_WIRE_0 & x3;
  assign SYNTHESIZED_WIRE_0 = ~x2;
  assign f = SYNTHESIZED_WIRE_1 | SYNTHESIZED_WIRE_2;

endmodule

```

Figure B.34. Verilog code for the circuit designed in Section B.3.

The Verilog code we wrote in section B.4, presented in Figure B.27, is equivalent to the automatically-generated code in Figure B.34. It can be instantiated in a top-level Verilog module in the normal way, as illustrated in Figure B.35. This module, named *example_mixed2*, implements the same function that we designed by using schematic capture in Figure B.31. The reader may wish to create a new Quartus II project for this code, which can then be compiled and simulated using the test vectors from Figure B.32.

```

module example_mixed2 (w1, w2, w3, w4, x2, f);
  input w1, w2, w3, w4, x2;
  output f;

  verfunctions gandh (w1, w2, w3, w4, g, h);
  example_schematic inst1 (g, x2, h, f);

endmodule

```

Figure B.35. The top-level Verilog module for the *example_mixed2* example.

B.6 Quartus II Windows

The Quartus II display contains a number of utility windows, which can be positioned in various places on the screen, changed in size, or closed. In Figure B.36, five Quartus II windows are displayed.

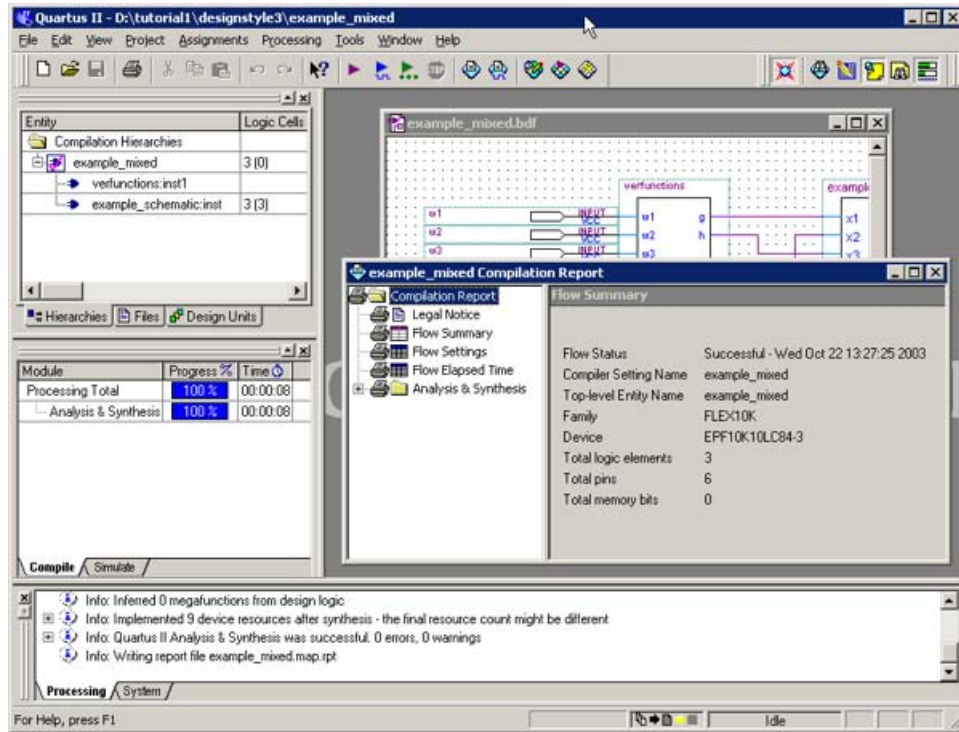


Figure B.36. The main Quartus II display.

The Project Navigator window is shown near the top left of the figure. Under the heading *Compilation Hierarchy*, it depicts a tree-like structure of the designed circuit using the names of the modules in the schematic of Figure B.31. To see the usefulness of this window, open the previously compiled project *example_mixed1* to get to the display that corresponds to Figure B.36. Now, double-click on the name *verfunctions* in the Project Navigator. Quartus II will automatically open the file *verfunctions.v*. Similarly, you can double-click on the name *example_schematic* and the corresponding schematic will be opened.

The Status window is located below the Project Navigator window. As you have already observed, this window displays the compilation progress as a project is being compiled by Quartus II. At the bottom of Figure B.36 there is the Message window, which displays user messages produced during the compilation process.

The large area on the right side of the Quartus II display is used for various purposes. As we have seen, it is used by the Block Editor, Text Editor, and Waveform Editor. It is also used to display various results of compilation and simulation.

A utility window can be moved by dragging its title bar, resized by dragging the window border, or closed by clicking on the X in the top-right corner. A specific utility window can be opened by using the **View | Utility Windows** command.

The commands available in Quartus II are *context sensitive*, depending on which Quartus II tool is currently being used. For example, when the Text Editor is in use, the Edit menu contains a different set of commands than when another tool, such as the Waveform Editor, is in use.

B.6.1 Concluding Remarks

This tutorial has introduced the basic use of the Quartus II CAD system. We have shown how to perform design entry by drawing a schematic and/or writing Verilog code. We have also illustrated how these design-entry methods can be mixed in a hierarchical design. Each design was compiled and then simulated using functional simulation.

In the next tutorial we will describe additional modules of Quartus II that are used to implement circuits in PLDs.