Contents

Getting Started with Altera's DE2-70 Board

This document describes the scope of Altera's DE2-70 Development and Education Board and the supporting materials provided by the Altera Corporation. It also explains the installation process needed to use a DE2-70 board connected to a computer that has the Quartus II CAD system installed on it.

1. Purpose of the DE2-70 Board

University and college courses on the design of logic circuits and computer organization usually include a laboratory component. In a modern curriculum, the laboratory equipment should ideally exemplify state-of-the-art technology and design tools, but be suitable for exercises that range from the simple tasks that illustrate the most basic concepts to challenging designs that require knowledge of advanced topics. From the logistic point of view, it is ideal if the same equipment can be used in all cases. The DE2 board has been designed to provide the desired platform.

2. Scope of the DE2-70 Board and Supporting Material

The DE2-70 board features a powerful Cyclone RII FPGA chip. All important components on the board are connected to the pins of this chip, allowing the user to configure the connection between the various components as desired. For simple experiments, the DE2 board includes a sufficient number of switches (of both toggle and pushbutton variety), LEDs, and 7-segment displays. For more advanced experiments, there are SRAM, SDRAM, and Flash memory chips. For experiments that require a processor and simple I/O interfaces, it is easy to instantiate Altera's Nios II processor and use interface standards such as RS-232 and PS/2. For experiments that involve sound or video signals, there are standard connectors provided on the board. For large design projects, it is possible to use the SD memory card.

Finally, it is possible to connect other user-designed boards to the DE2-70 board by means of two expansion headers. Software provided with the DE2-70 board features the Quartus II web edition design tools. It also includes a simple monitor program that allows the student to control various parts of the board in an easily understandable manner. There are also several applications that demonstrate the utility of the DE2-70 board. Traditionally, manufacturers of educational FPGA boards have provided a variety of boards and the CAD tools needed to implement designs on these boards. However, there has been a paucity of supporting materials that could be used directly for teaching purposes. Altera's DE2-70 board is a significant departure from this trend. In addition to the DE2-70 board, Altera Corporation provides a full set of associated exercises that can be performed in a laboratory setting for typical courses on logic design and computer organization. In effect, the DE2-70 board and the available exercises can be used as a ready-to-teach platform for such laboratories. Of course, the DE2-70 board is also likely to be suitable for exercises that have been developed for

other hardware platforms and can be ported to the DE2-70 platform.

3. Installation and USB-Blaster Driver

The DE2-70 board is shipped in a package that includes all parts necessary for its operation. The only essential parts are the 12-volt power adapter and the USB cable. There is also a protective plexiglass cover that may be used in the laboratory environment to protect the board from accidental physical damage. Plug in the 12-volt adapter to provide power to the board. Use the USB cable to connect the USB connector (the one closest to the power switch) on the DE2-70 board to a USB port on a computer that runs the Quartus II software. Turn on the power switch on the DE2-70 board.

The computer will recognize the new hardware connected to its USB port, but it will be unable to proceed if it does not have the required driver already installed. The DE2-70 board is programmed by using Altera's USB-Blaster mechanism. If the USB-Blaster driver is not already installed, the New Hardware Wizard in Figure 1 will appear.

Figure 1. Found New HardwareWizard.

Since the desired driver is not available on the Windows Update Web site, select No, not this time in response to the question asked and click Next. This leads to the window in Figure 2.

The driver is available within the Quartus II software. Hence, select Install from a specific location and clickNext to get to Figure 3.

Now, choose Search for the best driver in these locations and click Browse to get to the pop-up box in Figure 4. Find the desired driver, which is at location \<Quartus II system directory>\drivers\usb-blaster. Click OK (For Quartus II version 6.1 and later, select the appropriate driver version based on the 64-bit or 32-bit Quartus II version.) and then upon returning to Figure 3 click Next. At this point the installation will commence, but a dialog box in Figure 5 will appear indicating that the driver has not passed the Windows Logo testing. Click

Continue Anyway.

The driver will now be installed as indicated in Figure 6. Click Finish and you can start using the DE2-70 board.

Figure 2. The driver is found in a specific location.
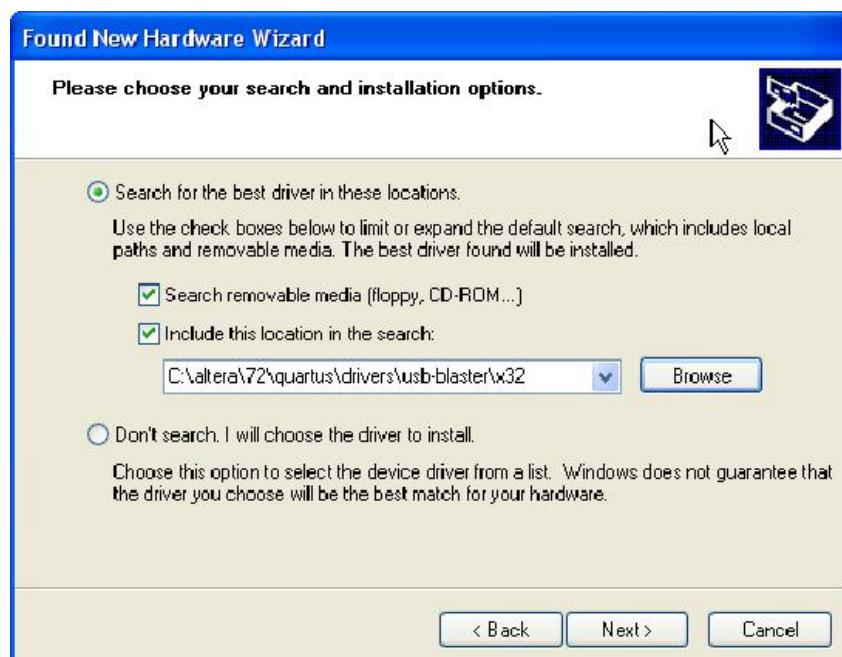
Figure 3. Specify the location of the driver.
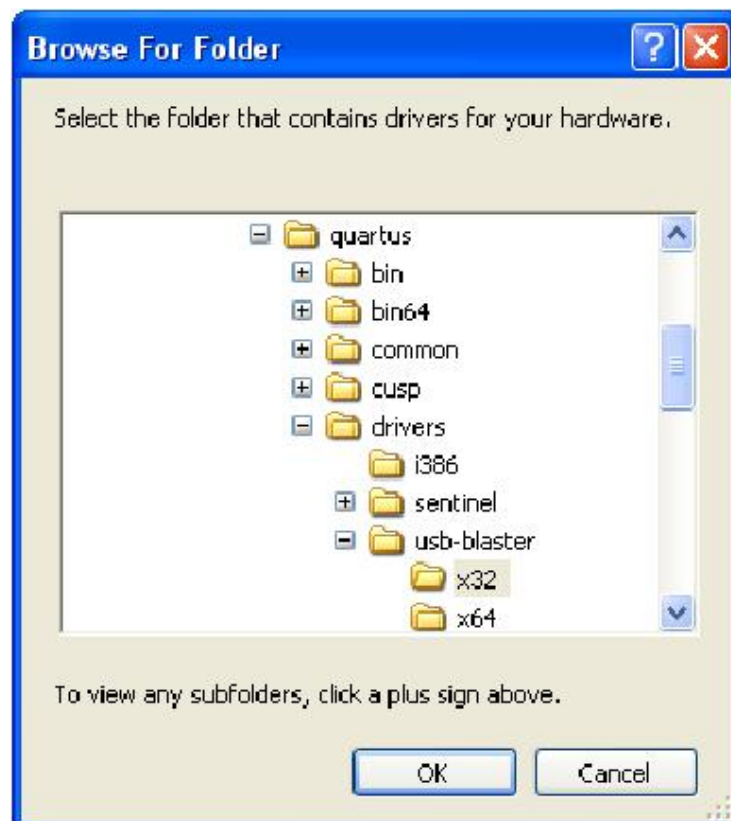
Figure 4. Browse to find the location.

Figure 5. There is no need to test the driver.

Figure 6. The driver is installed.

4. Using the DE2-70 Board

The DE2-70 board is used in conjuction with the Quartus II software. A reader who is not familiar with this software should read an introductory tutorial. There are three versions of the tutorial:

Quartus II Introduction Using Verilog Design

Quartus II Introduction Using VHDL Design

Quartus II Introduction Using Schematic Design

These tutorials cover the same aspects of the Quartus II software; they differ only in the design entry method that is used. They illustrate the entire process of implementing a design targetted for the DE2-70 board.

Detailed information about the DE2-70 board is given in the DE2-70 User Manual, which describes all of the features of the board. It also describes a Control Panel utility which allows the user to write/read data into/from various components on the board in a simple and direct manner. The user is encouraged to explore and make use of this utility.

Quartus II Introduction Using VHDL Design

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

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

The last step in the design process involves configuring the designed circuit in an actual FPGA device. To show how this is done, it is assumed that the user has access to the Altera DE2 Development and Education board connected to a computer that has Quartus II software installed. A reader who does not have access to the DE2 board will still find the tutorial useful to learn how the FPGA programming and configuration task is performed.

The screen captures in the tutorial were obtained using the Quartus II version 5.0; if other versions of the software are used, some of the images may be slightly different.

Computer Aided Design (CAD) software makes it easy to implement a desired logic circuit by using a pro-grammable logic device, such as a field-programmable gate array (FPGA) chip. A typical FPGA CAD flow is illustrated in Figure 7.

The CAD flow involves the following steps:

· **Design Entry** – the desired circuit is specified either by means of a schematic diagram, or by using a hardware description language, such as VHDL or Verilog

· **Synthesis** – the entered design is synthesized into a circuit that consists of the logic elements (LEs) provided in the FPGA chip

· **Functional Simulation** – the synthesized circuit is tested to verify its functional correctness; this simulation does not take into account any timing issues

· **Fitting** – the CAD Fitter tool determines the placement of the LEs defined in the netlist into the LEs in an actual FPGA chip; it also chooses routing wires in the chip to make the required connections between specific LEs

· **Timing Analysis** – propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit

· **Timing Simulation** – the fitted circuit is tested to verify both its functional correctness and timing

· **Programming and Configuration** – the designed circuit is implemented in a physical FPGA chip by pro-gramming the configuration switches that configure the LEs and establish the required wiring connections

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

this tutorial, the reader will learn about:

- Creating a project
- Design entry using VHDL code
- Synthesizing a circuit specified in VHDL code
- Fitting a synthesized circuit into an Altera FPGA
- Assigning the circuit inputs and outputs to specific pins on the FPGA
- Simulating the designed circuit
- Programming and configuring the FPGA chip on Altera's DE2 board
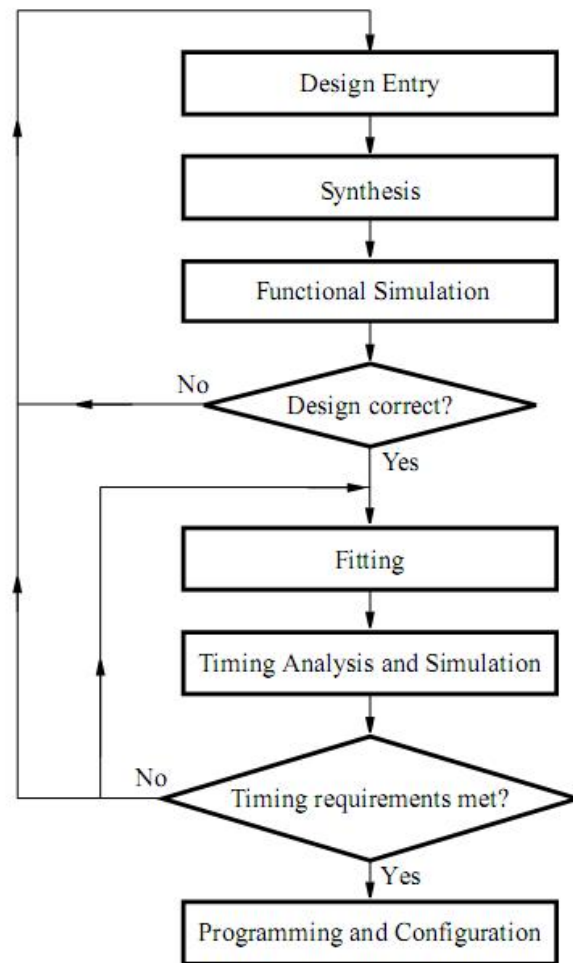
Figure 7. Typical CAD flow

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 8. 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 8 clicking the left mouse button on the menu named File opens the menu shown in Figure 9. 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.
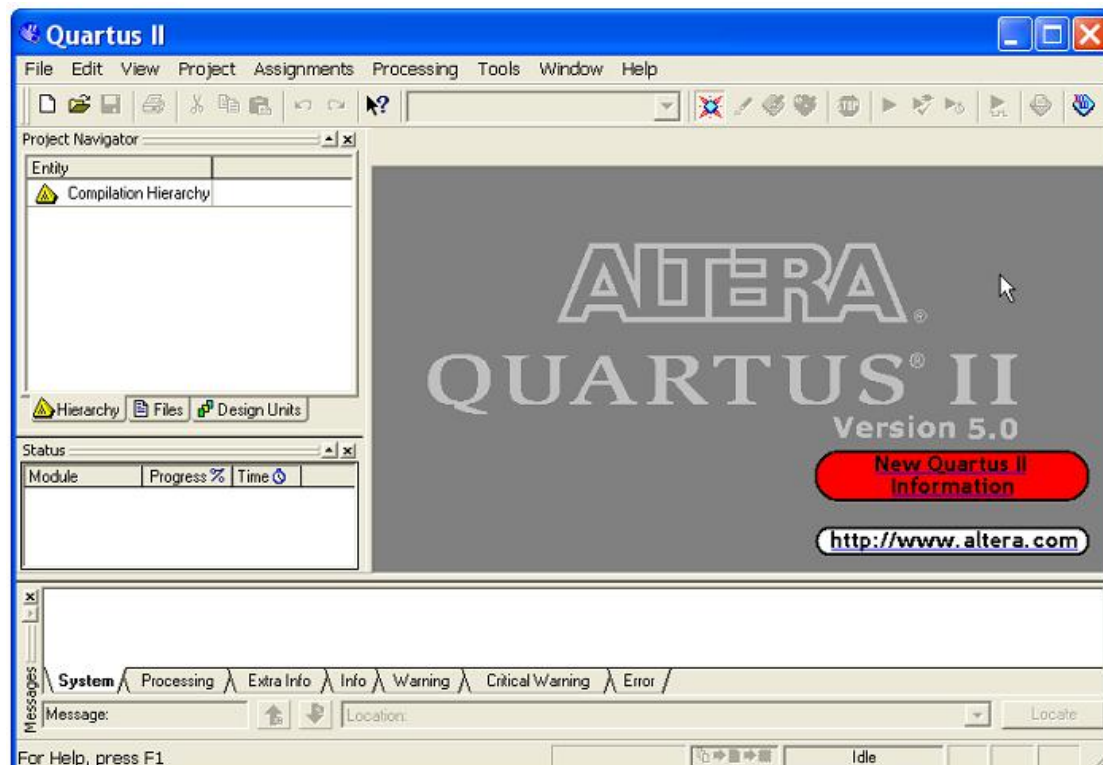

Figure 8. The main Quartus II display.
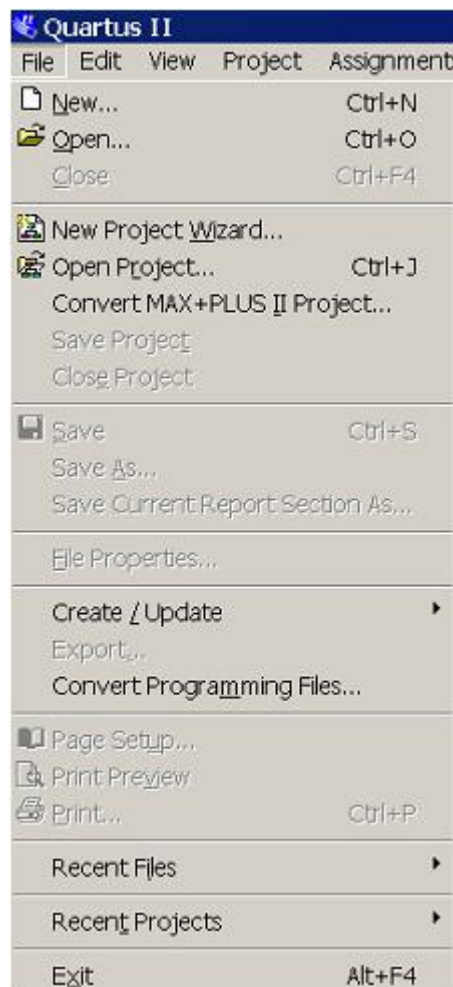
Figure 9. An example of the File menu.

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.

1.1 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 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 menu.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 most applications,

pressing the F1 function key on the keyboard opens a Help display that shows the commands available for the application.

2. 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 to reach the window in Figure 10, which indicates the capability of this wizard. You can skip this window in subsequent projects by checking the box Don't show me this intro-duction again. Press Next to get the window shown in Figure 11.

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 11. Press Next. Since we have not yet created the directory introtutorial, Quartus II software displays the pop-up box in Figure 12 asking if it should create the desired directory. Click Yes, which leads to the window in Figure 13.
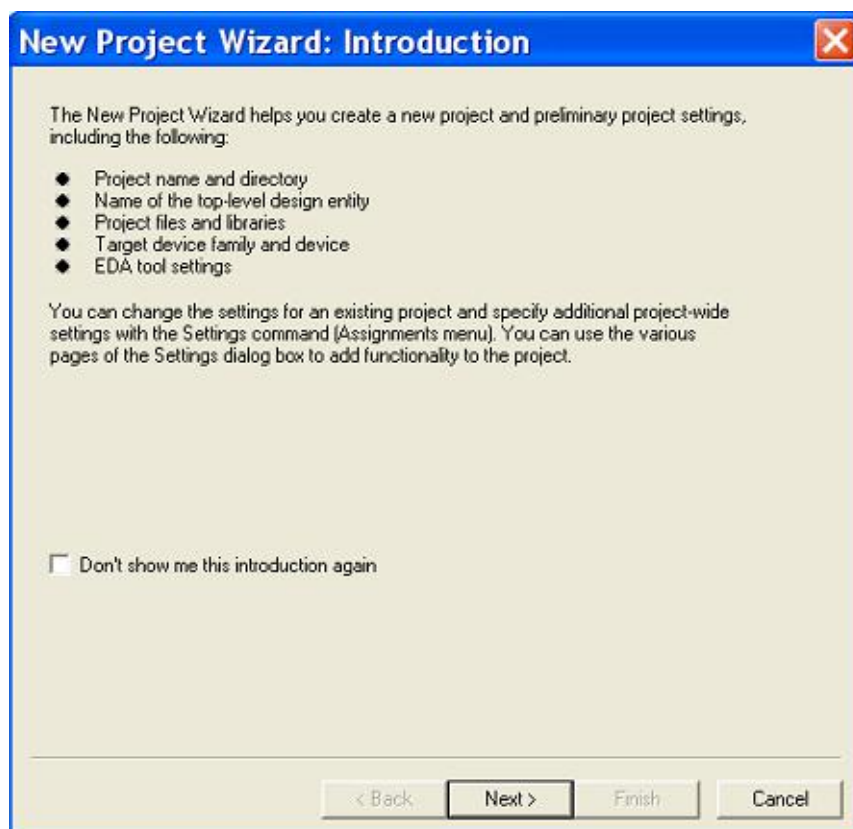
Figure 10. Tasks performed by the wizard.
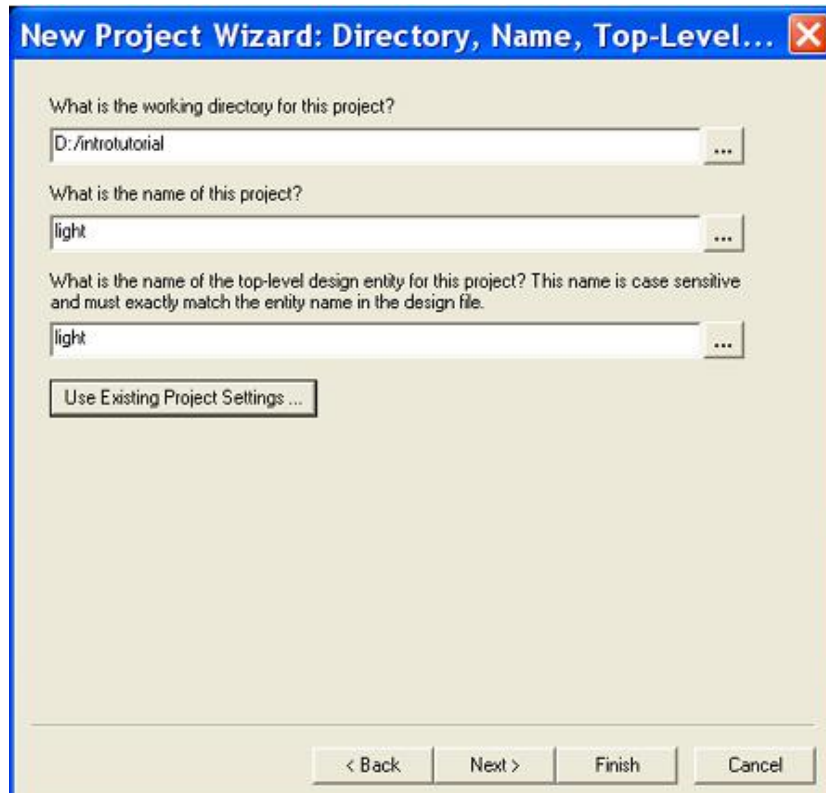
Figure 11. Creation of a new project.
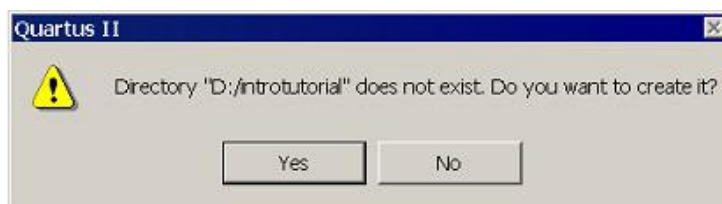
Figure 12. Quartus II software can create a new directory for the project.
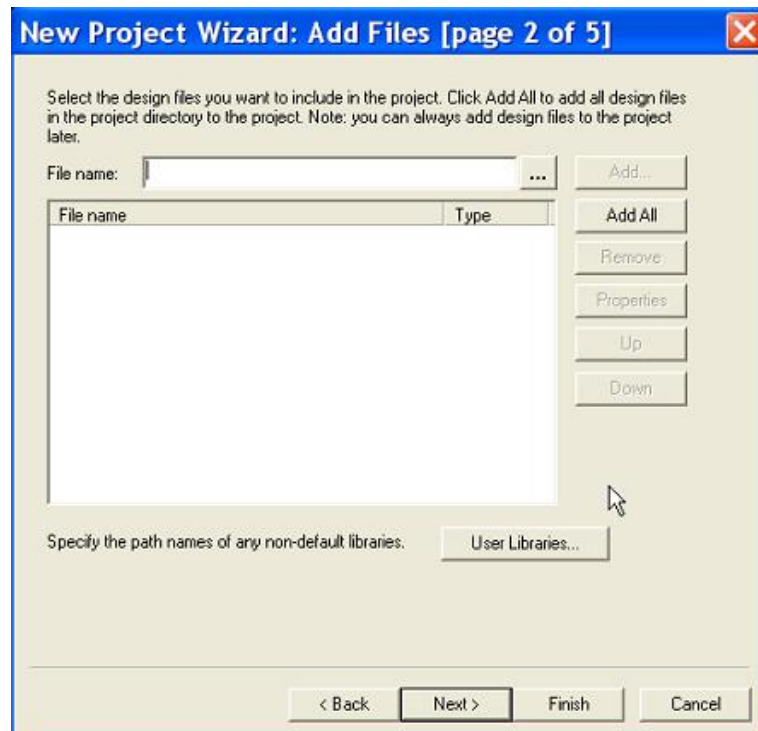
Figure 13. The wizard can include user-specified design files.

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 14.
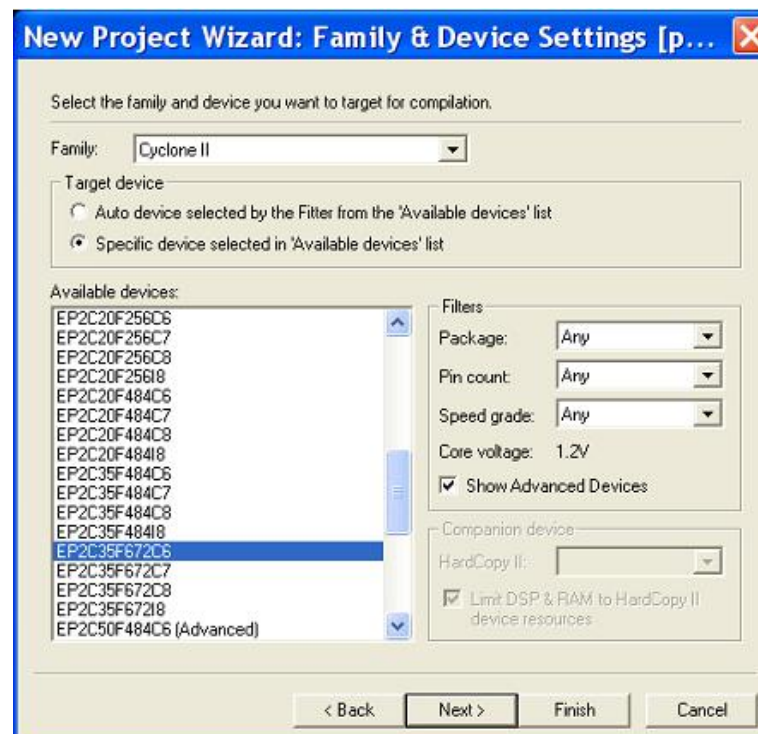
Figure 14. Choose the device family and a specific device.

4. We have to specify the type of device in which the designed circuit will be implemented. Choose Cyclone^{TM} II as the target device family. 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 device called EP2C70F896C6 which is the FPGA used on Altera's DE2 board. Press Next, which opens the window in Figure 15.
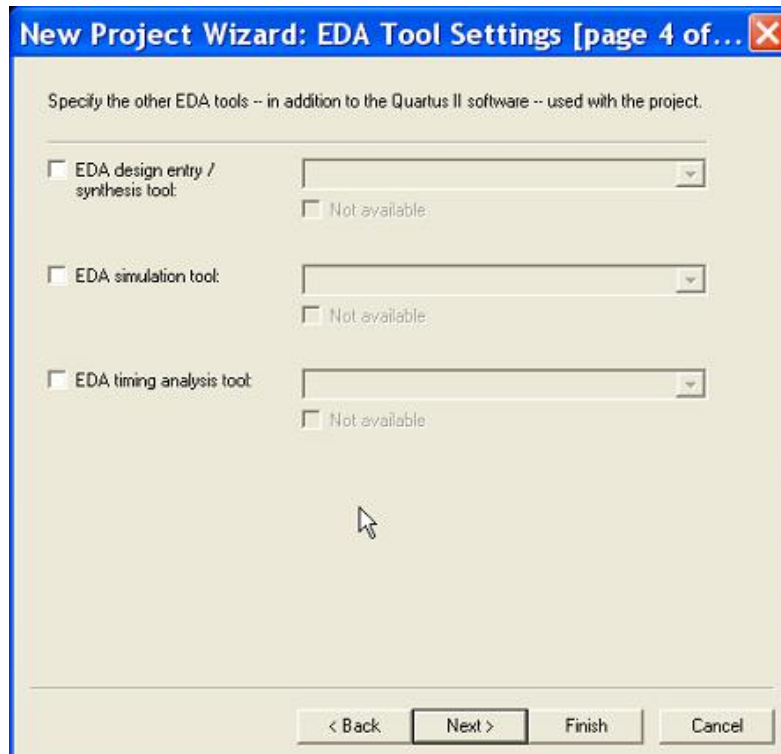
Figure 15. Other EDA tools can be specified.

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 16. 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 17.
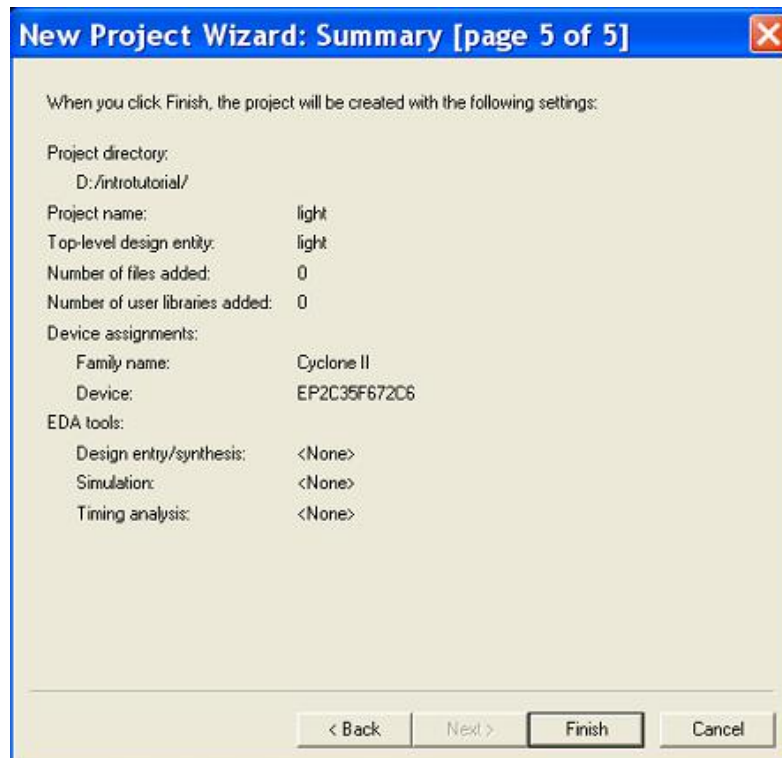
Figure 16. Summary of the project settings.
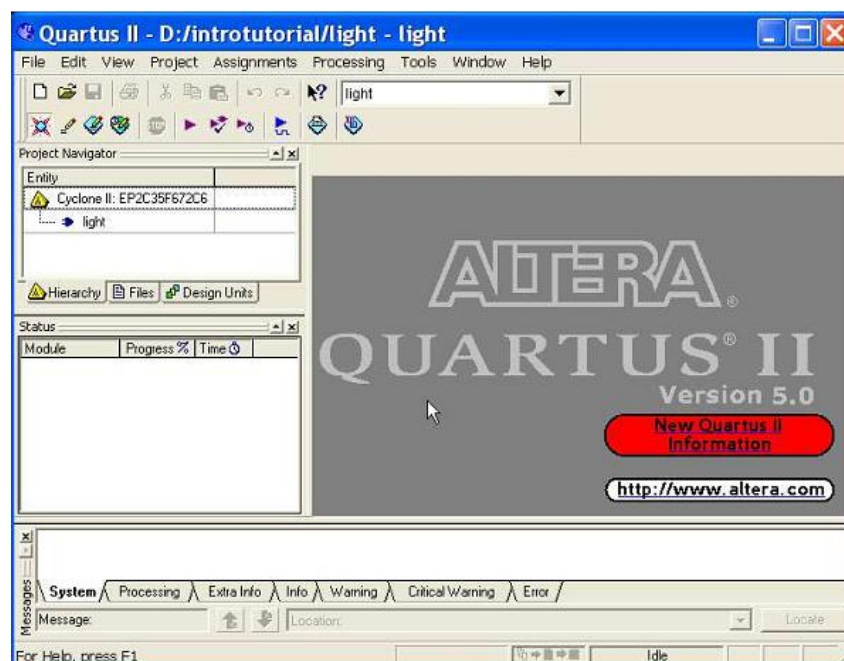
Figure 17. The Quartus II display for the created project.

3 . Design Entry Using VHDL Code

As a design example, we will use the two-way light controller circuit shown in Figure 18. The circuit can be used to control a single light from either of the two switches, x1 and x2, where a closed switch corresponds to the logic value 1. The truth table for the circuit is also given in the

figure. Note that this is just the Exclusive-OR function of the inputs x1 and x2, but we will specify it using the gates shown.
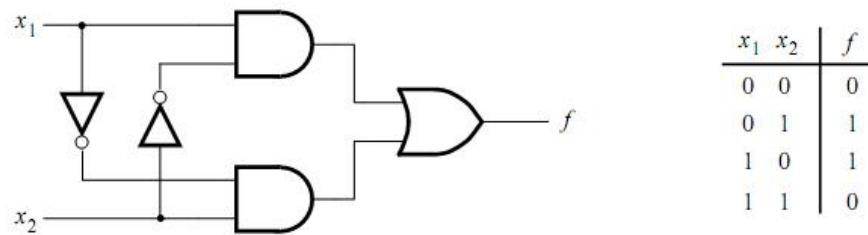
Figure 18. The light controller circuit.

The required circuit is described by the VHDL code as next. Note that the VHDL entity is called light to match the name given in Figure 17, which was specified when the project was created. This code can be typed into a file by using any text editor that stores ASCII files, or by using the Quartus II text editing facilities. While the file can be given any name, it is a common designers' practice to use the same name as the name of the top-level VHDL entity. The file name must include the extension vhd, which indicates a VHDL file. So, we will use the name light.vhd.

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

3.1 Using the Quartus II Text Editor

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

Select File > New to get the window in Figure 19, choose VHDL File, and click OK. This opens the Text Editor window. The first step is to specify a name for the file that will be created. Select File > Save As to open the pop-up box depicted in Figure 20. In the box labeled Save as type choose VHDL File. In the box labeled File name type light. Put a checkmark in the box Add file to current project. Click Save, which puts the file into the directory introtutorial and leads to the Text Editor window shown in Figure 21. Maximize the Text Editor window and enter the VHDL code into it. Save the file by typing File > Save, or by typing the shortcut Ctrl-s.
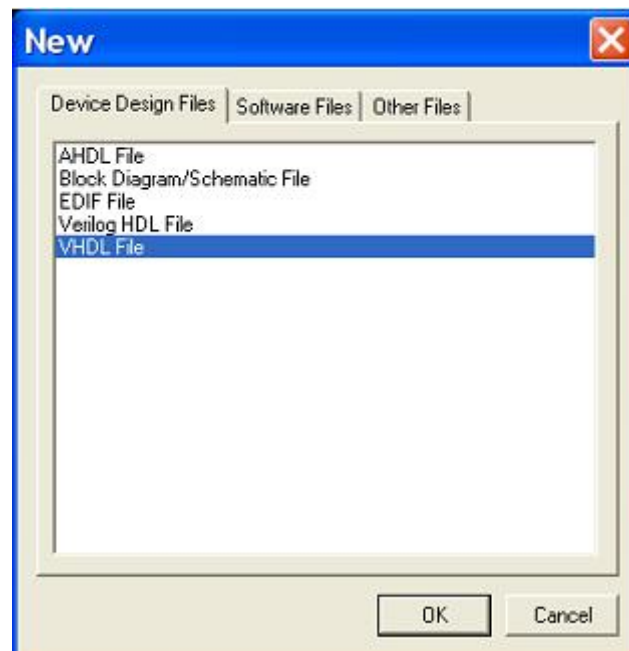
Figure 19. Choose to prepare a VHDL file.
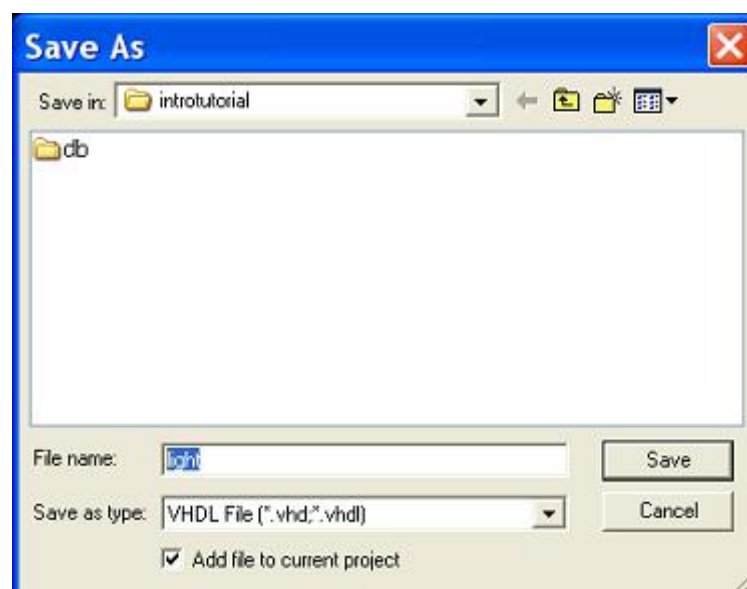
Figure 20. Name the file.

Figure 21. Text Editor window.

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

3.1.1 Using VHDL Templates

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

3.2 Adding Design Files to a Project

As we indicated when discussing Figure 13, you can tell Quartus II software which design files it should use as part of the current project. To see the list of files already included in the light project, select Assignments > Settings, which leads to the window in Figure 22. As indicated on the left side of the figure, click on the item Files. An alternative way of making this selection is to choose Project > Add/Remove Files in Project.
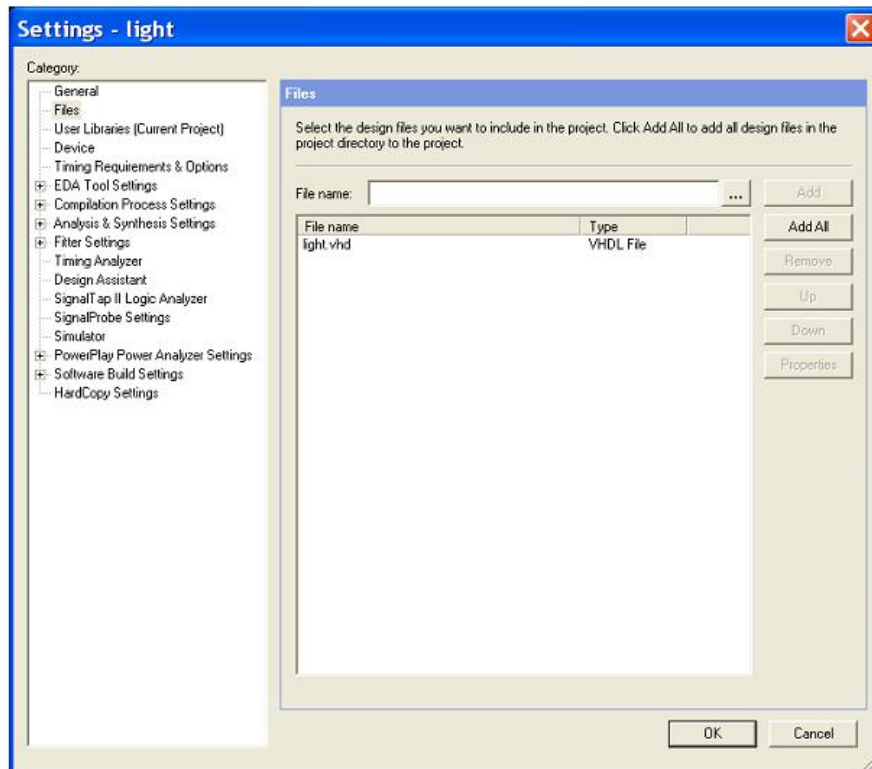
Figure 22. Settings window.

If you used the Quartus II Text Editor to create the file and checked the box labeled Add file to current project, as described in Section 3.1, then the light.vhd file is already a part of the project and will be listed in the window in Figure 22. Otherwise, the file must be added to the project. So, if you did not use the Quartus II Text Editor, then place a copy of the file light.vhd, which you created using some other text editor, into the directory introtutorial. To add this file to the project, click on the File name: button in Figure 22 to get the pop-up window in Figure 23. Select the light.vhd file and click Open. The selected file is now indicated in the Files window of Figure 23. Click OK to include the light.vhd file in the project. We should mention that in many cases the Quartus II software is able to automatically find the right files to use for each entity referenced in VHDL code, even if the file has not been explicitly added to the project. However, for complex projects that involve many files it is a good design practice to specifically add the needed files to the project, as described above.
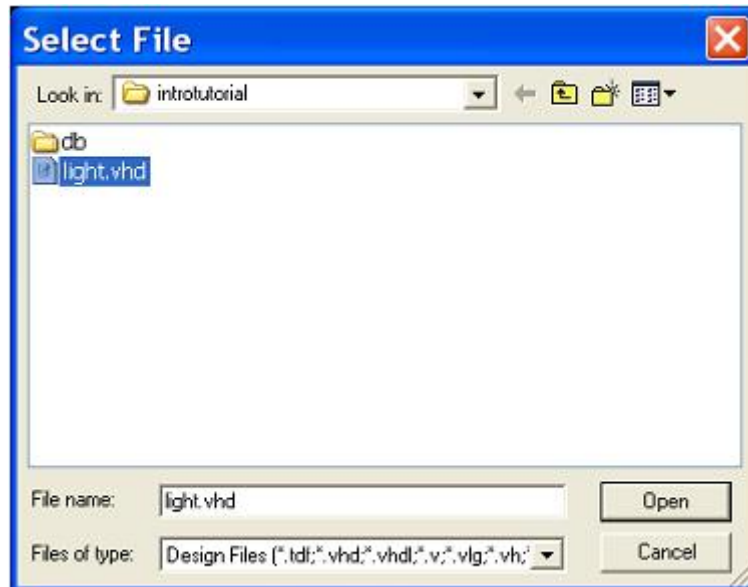
Figure 23. Select the file.

4. Compiling the Designed Circuit

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

Run the Compiler by selecting Processing > Start Compilation, or by clicking on the toolbar icon ▶ that looks like a purple triangle. 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 24. In the message window, at the bottom of the figure, various messages are displayed. In case of errors, there will be appropriate messages given.

When the compilation is finished, a compilation report is produced. A window showing this report is opened automatically, as seen in Figure 24. The window can be resized, maximized, or 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 of its window. Figure 24 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. Another section is shown in Figure 25. It is reached by selecting Analysis & Synthesis > Equations on the left side of the compilation report. Here we see the logic expressions produced by the Compiler when synthesizing the designed circuit. Observe that f is the output derived as

$$f = x2 \ \$ \ x1$$

where the $ sign is used to represent the Exclusive-OR operation. Obviously, the Compiler recognized that the logic expression in our design file is equivalent to this expression.
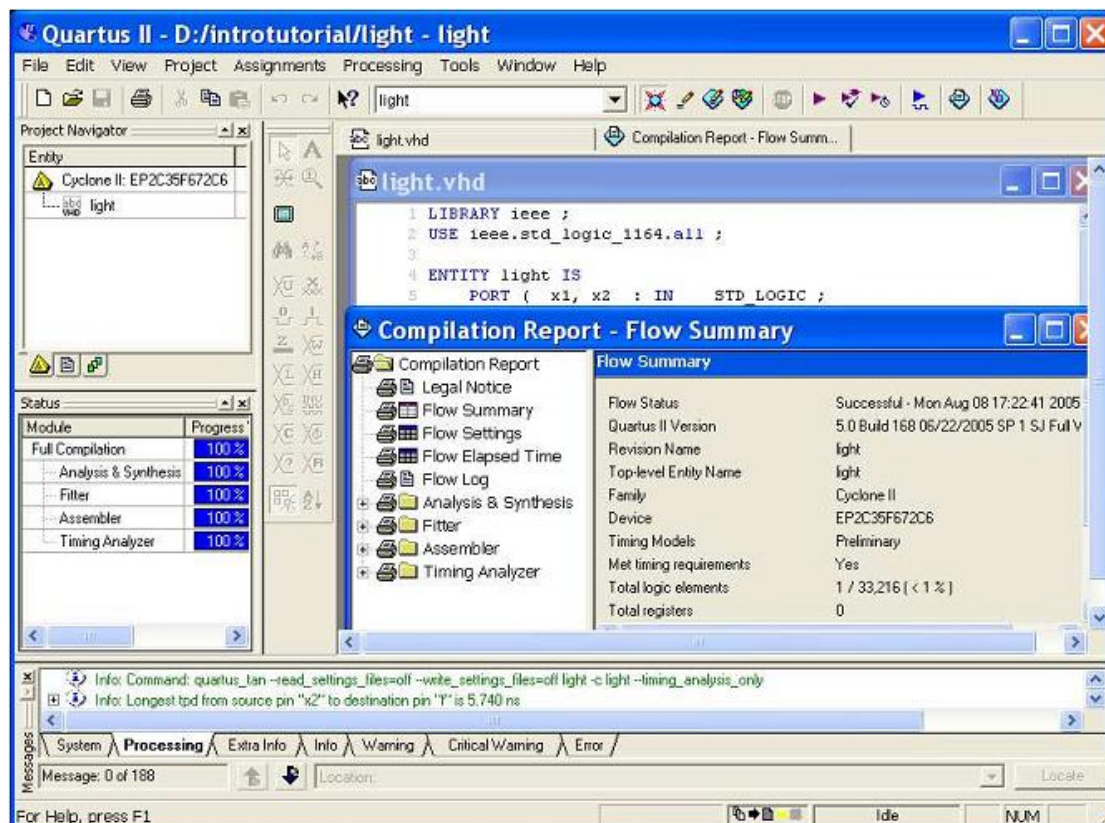
Figure 24. Display after a successful compilation.
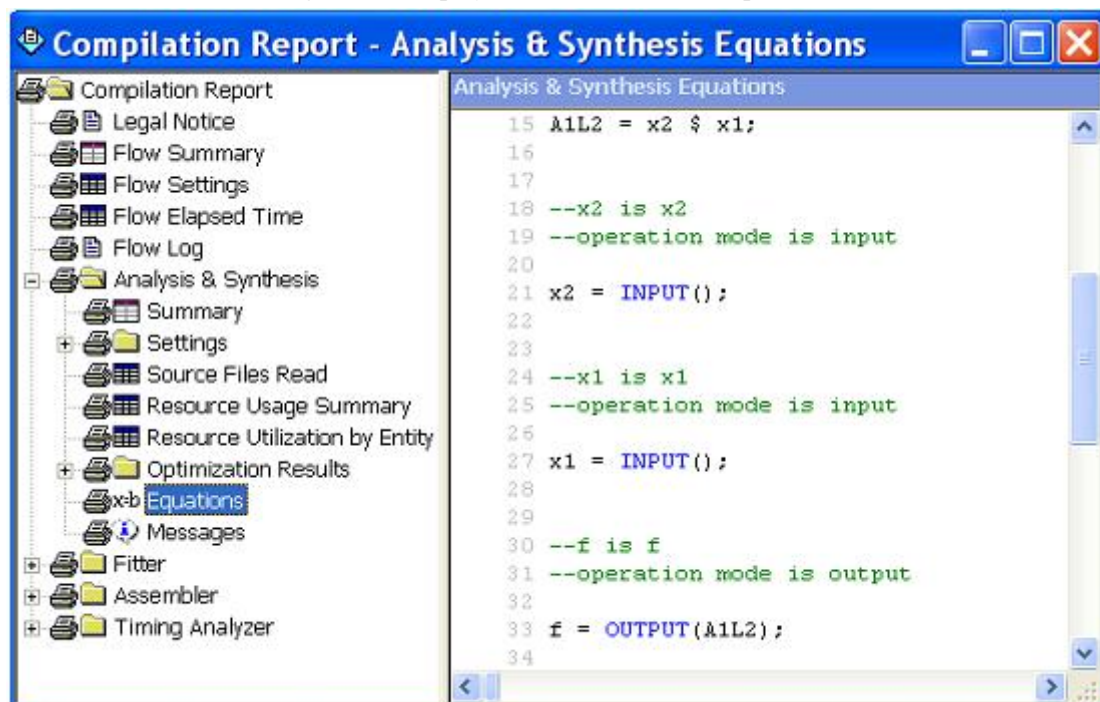
Figure 25. Compilation report showing the synthesized equations.

4.1 Errors

Quartus II software displays messages produced during compilation in theMessages window. If the VHDL design file is correct, one of the messages will state that the compilation was

successful and that there are no errors.

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

To see the effect of an error, open the file light.vhd. Remove the semicolon in the statement that defines the function f, illustrating a typographical error that is easily made. Compile the erroneous design file by clicking on the icon ▶. A pop-up box will ask if the changes made to the light.vhd file should be saved; click Yes. After trying to compile the circuit, Quartus II software will display a pop-up box indicating that the compilation was not successful. Acknowledge it by clicking OK. The compilation report summary, given in Figure 26, now confirms the failed result. Expand the Analysis & Synthesis part of the report and then select Messages to have the messages displayed as shown in Figure 27. Double-click on the first error message. Quartus II software responds by opening the light.vhd file and highlighting the statement which is affected by the error, as shown in Figure 28. Correct the error and recompile the design.
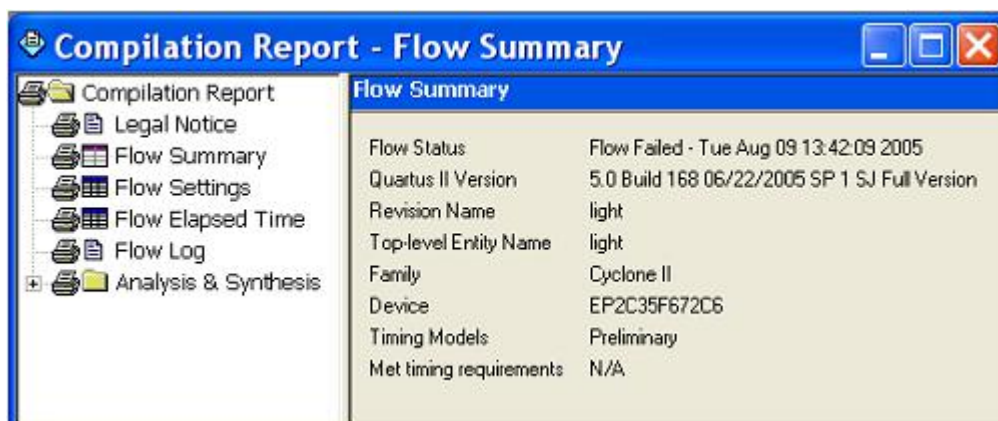
Figure 26. Compilation report for the failed design.
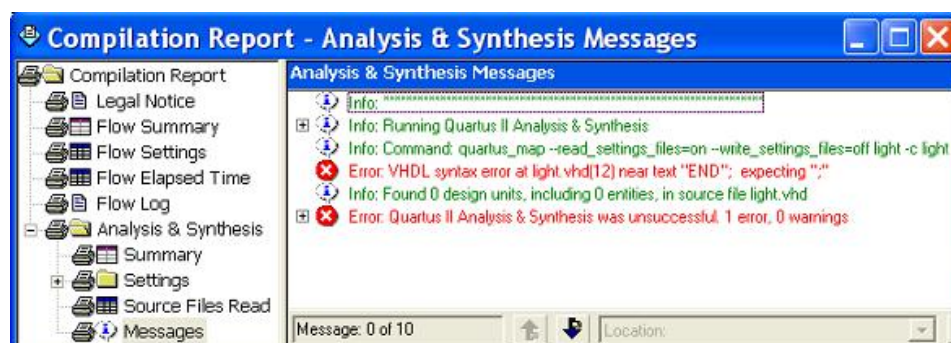
Figure 27. Error messages.
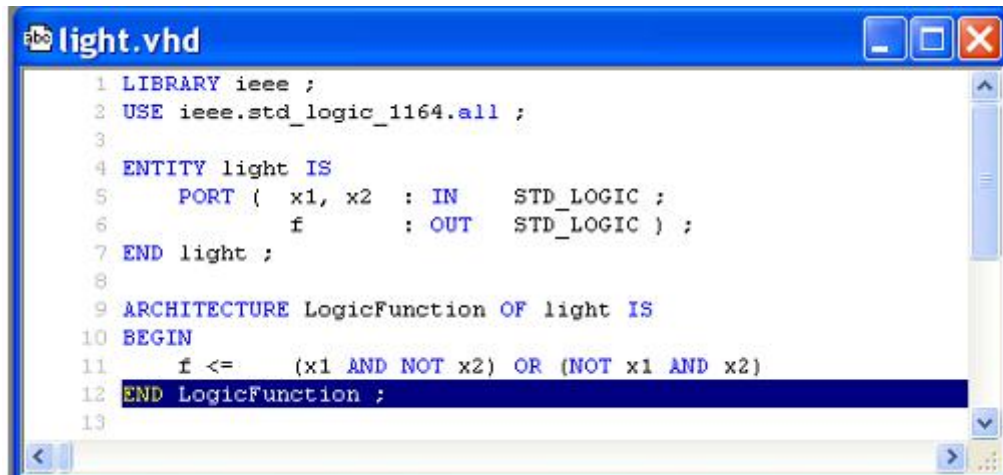
Figure 28. Identifying the location of the error.

5. 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 DE2 board has hardwired connections between the FPGA pins and the other components on the board. We will use two toggle switches, labeled SW0 and SW1, to provide the external inputs,x1 and x2, to our example circuit. These switches are connected to the FPGA pins N25 and N26, respectively. We will connect the output f to the green light-emitting diode labeled LEDG0, which is hardwired to the FPGA pin AE22.

Pin assignments are made by using the Assignment Editor. Select Assignments > Pins to reach the window in Figure 29. Under Category select Pin. Double-click on the entry <<new>> which is highlighted in blue in the column labeled To. The drop-down menu in Figure 30 will appear. Click on x1 as the first pin to be assigned; this will enter x1 in the displayed table. Follow this by double-clicking on the box to the right of this new x1 entry, in the column labeled Location. Now, the drop-down menu in Figure 31 appears. Scroll down and select PIN_N25. Instead of scrolling down the menu to find the desired pin, you can just type the name of the pin (N25) in the Location box. Use the same procedure to assign input x2 to pin N26 and output f to pin AE22, which results in the image in Figure 32. To save the assignments made, choose File > Save. You can also simply close the Assignment Editor window, in which case a pop-up box will ask if you want to save the changes to assignments; click Yes. Recompile the circuit, so that it will be compiled with the correct pin assignments.
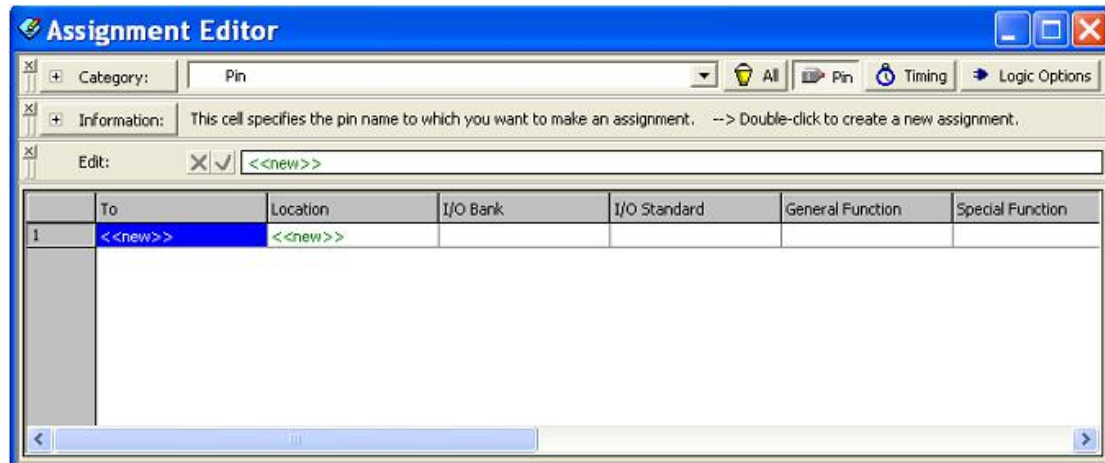
Figure 29. The Assignment Editor window.
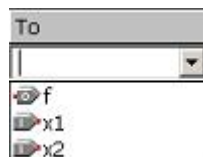
Figure 30. The drop-down menu displays the input and output names.

Figure 31. The available pins.
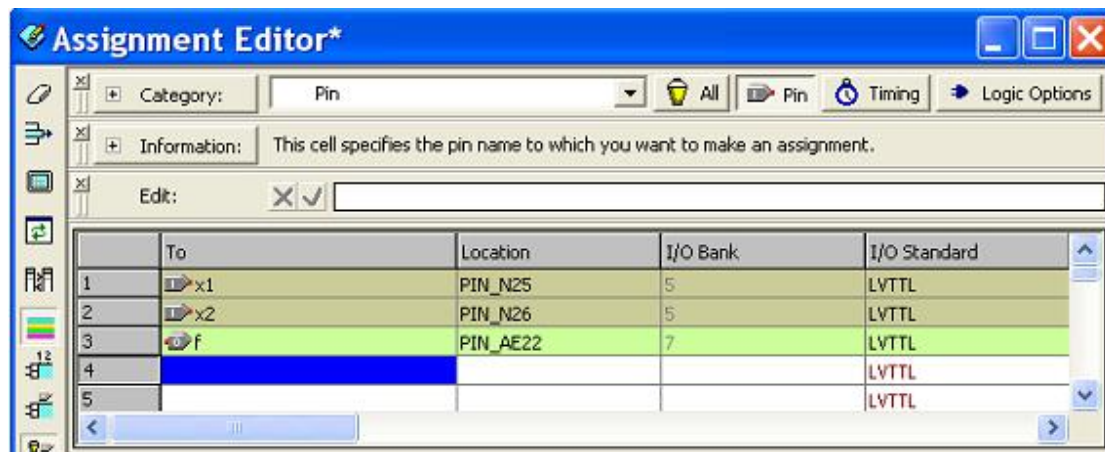
Figure 32. The complete assignment.

The DE2 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 comma separated value (CSV) format, which is a common text file format that contains comma-delimited values. This file format is often used in conjunction with the Microsoft Excel spreadsheet program, but the file can also be created by hand using any plain ASCII text editor. The format for the file for our simple project is

<div align="center">

To, Location

x1, PIN_N25

x2, PIN_N26

f, PIN_AE22

</div>

NOTE:

Refer to the table A of   DE2_70_pin_assignments

By adding lines to the file, any number of pin assignments can be created. Such csv 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 33. Now, select File > Export which leads to the window in Figure 34. Here, the file light.csv is available for export. Click on Export. If you now look in the directory introtutorial, you will see that the file light.csv has been created.
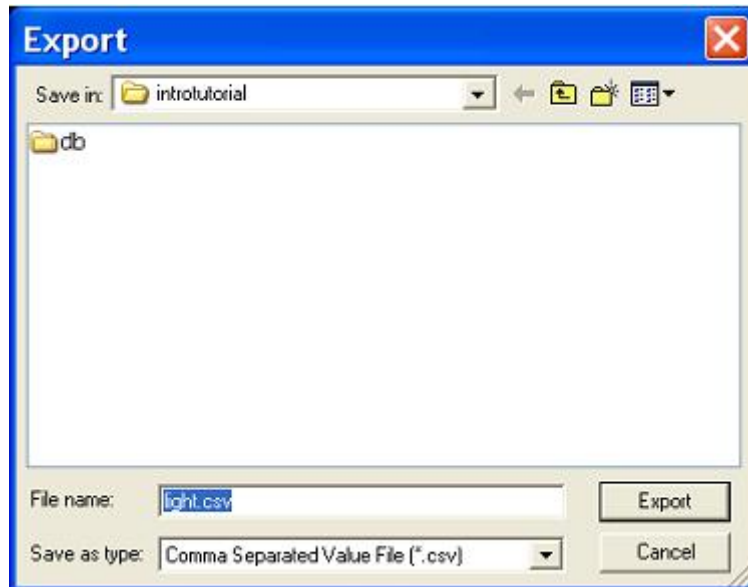
Figure 33. Exporting the pin assignment.

You can import a pin assignment by choosing Assignments > Import Assignments. This opens the dia-logue in Figure 34 to select the file to import. Type the name of the file, including the csv 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.
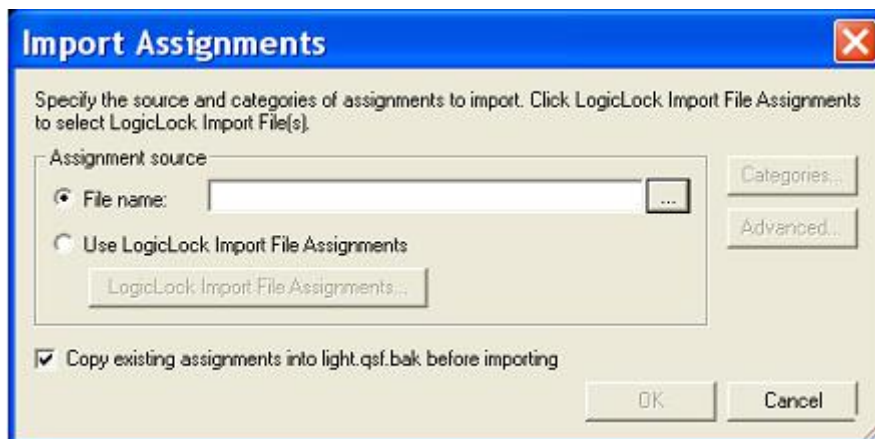
Figure 34. Importing the pin assignment.

For convenience when using large designs, all relevant pin assignments for the DE2 board are given in the file called DE2_pin_assignments.csv in the directory DE2_tutorials\design_files, which is included on the CD-ROM that accompanies the DE2 board and can also be found on Altera's DE2 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 DE2_pin_assignments.csv file as elements of arrays SW and LEDG, we must refer to them in the same way in the VHDL design file. For example, in the DE2_pin_assignments.csv 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).

6. Simulating the Designed Circuit

Before implementing the designed circuit in the FPGA chip on the DE2 board, it is prudent to simulate it to ascertain its correctness. Quartus II software includes a simulation tool that 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 <u>test vectors</u>, 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 Quartus II Waveform Editor to draw the test vectors, as follows:

1. Open the Waveform Editor window by selecting File > New, which gives the window shown in Figure 35.Click on the Other Files tab to reach the window displayed in Figure 36. Choose Vector Waveform File and click OK.
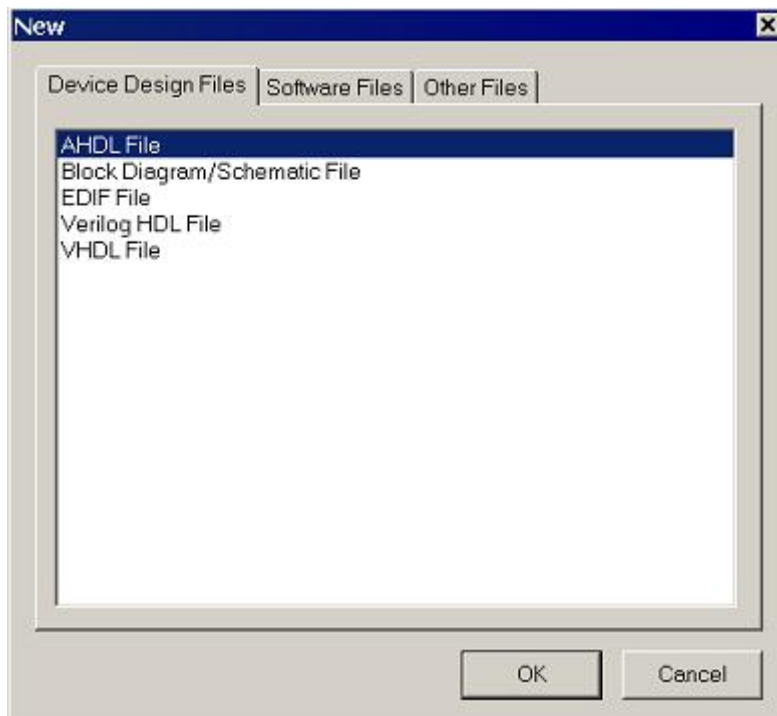
Figure 35. Need to prepare a new file.
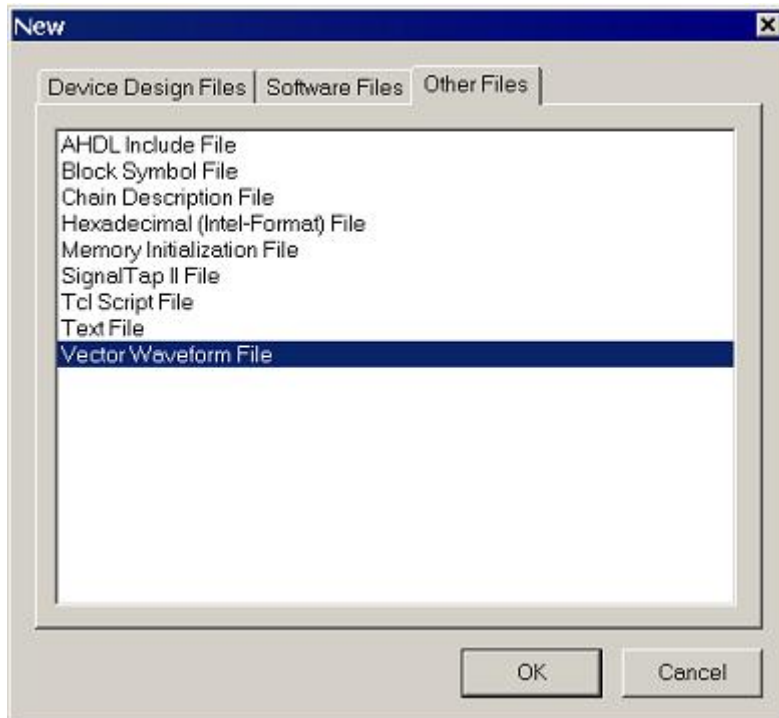
Figure 36. Choose to prepare a test-vector file.

2. TheWaveform Editor window is depicted in Figure 37. Save the file under the name light.vwf; note that this changes the name in the displayed window. Set the desired simulation to run from 0 to 200 ns by selecting Edit > 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 38. You may wish to resize the window to its maximum size.
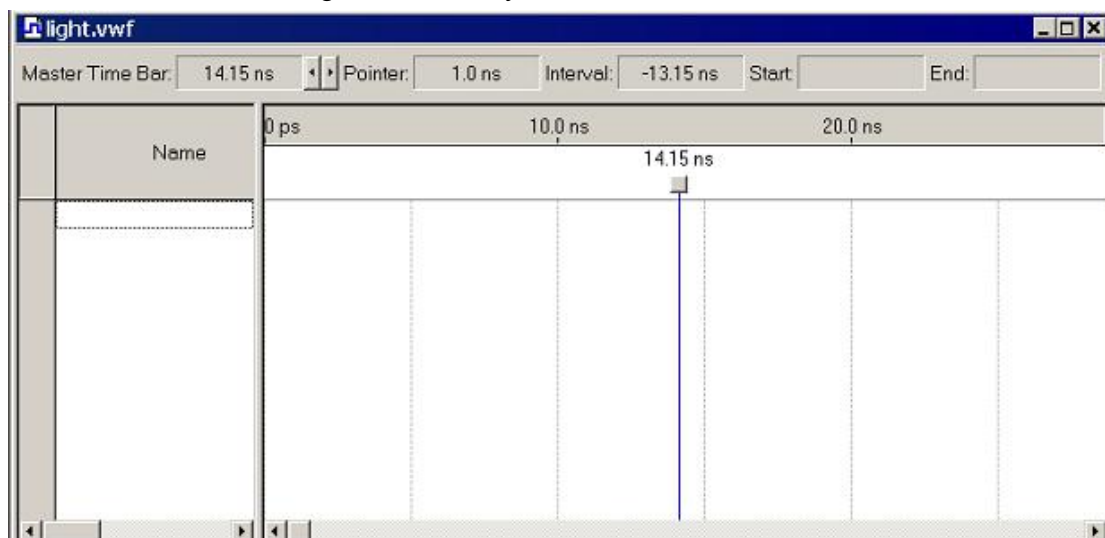
Figure 37. The Waveform Editor window.
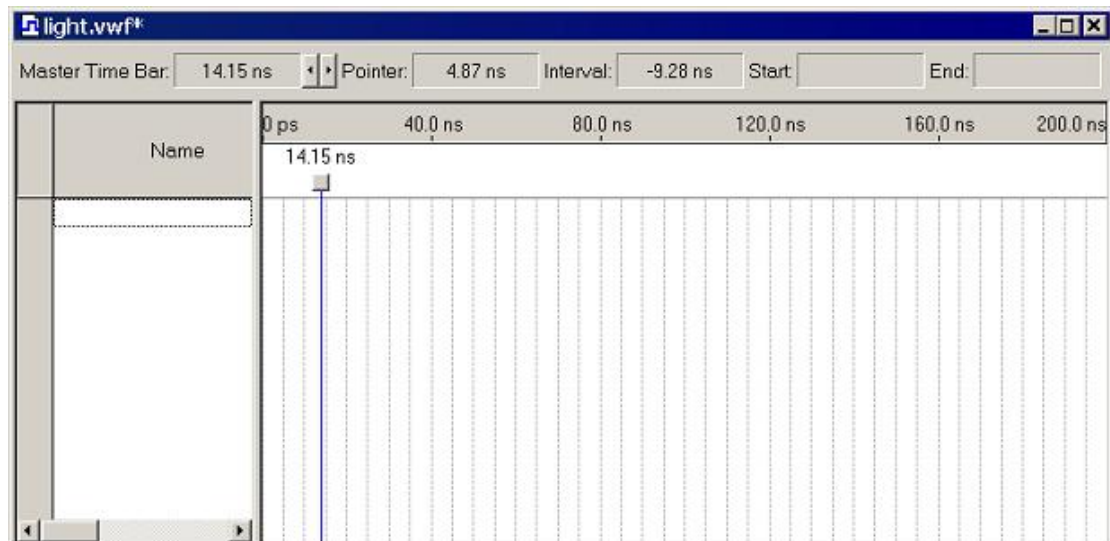
Figure 38. The augmentedWaveform Editor window.

3. Next, we want to include the input and output nodes of the circuit to be simulated. Click Edit > Insert Node or Bus to open the window in Figure 39. It is possible to type the name of a signal (pin) into the Name box, but it is easier to click on the button labeled Node Finder to open the window in Figure 40. 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.
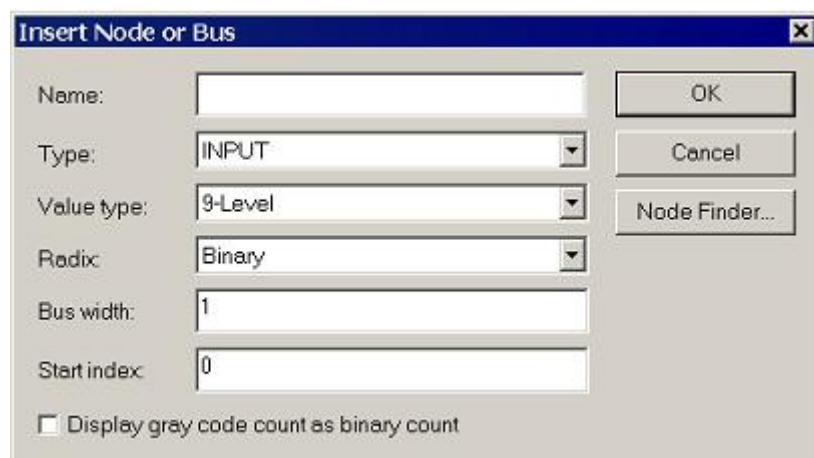
Figure 39. The Insert Node or Bus dialogue.
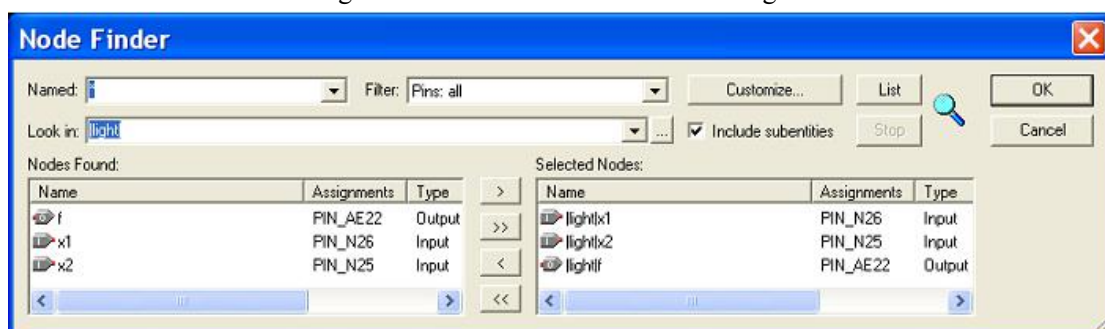
Figure 40. Selecting nodes to insert into the Waveform Editor.

Click on the x1 signal in the Nodes Found box in Figure 40, 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 39. This leaves a fully displayed Waveform Editor window, as shown in Figure 41. If you did not select the nodes in the same order as displayed in Figure 41, 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.
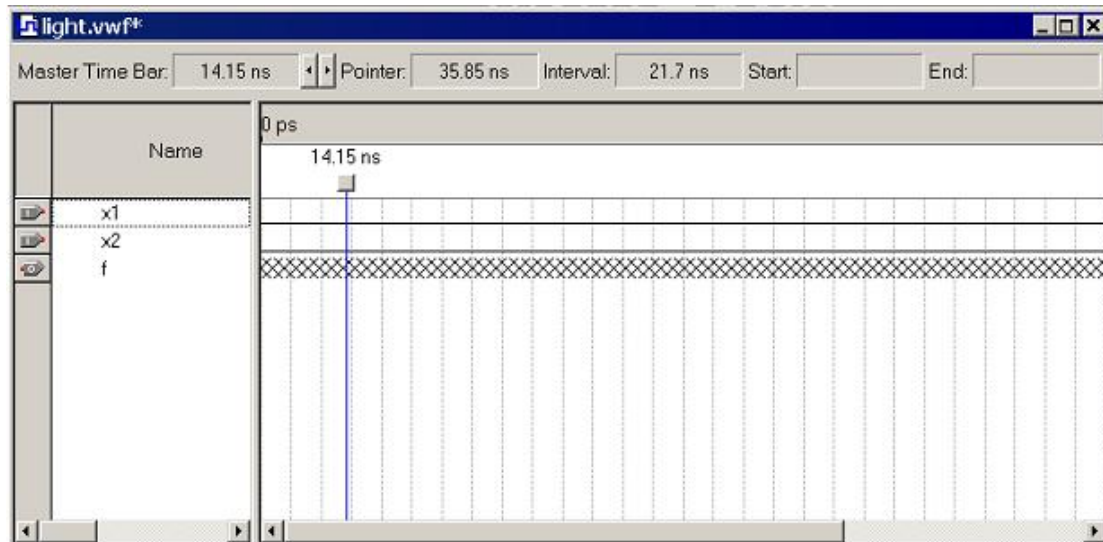
Figure 41. The nodes needed for simulation.

4. We will now specify the logic values to be used for the input signals x1 and x2 during simulation. The logic values at the output f will be generated automatically by the simulator. To make it easy to draw the desired waveforms, the Waveform Editor displays (by default) vertical guidelines and provides a drawing feature that snaps on these lines (which can otherwise be invoked by choosing View > 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 $\qquad$ in the toolbar, or the Waveform

Editing Tool, which is activated by the icon $\qquad$.

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 18. 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 name for the x1 node. Once a waveform is selected, the editing commands in theWaveform 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), don't care (DC), 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 Edit menu can also be opened by right-clicking on a

31

waveform name.

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 18. This should produce the image in Figure 42. 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.
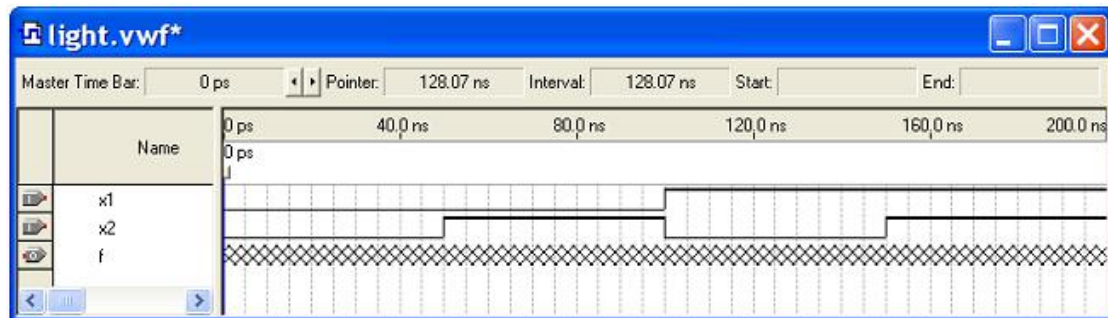
Figure 42. Setting of test values.

6.1 Performing the Simulation

A designed circuit can be simulated in two ways. The simplest way is to assume that logic elements and intercon-nection 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. This takes much less time, because the simulation can be performed simply by using the logic expressions that define the circuit.

6.1.1 Functional Simulation

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 43, choose Functional as the simulation mode, and click OK. The Quartus II simulator takes the inputs and generates the outputs defined in the light.vwf

file. Before running the functional simulation it is necessary to create the required netlist, which is done by se-lecting Processing > Generate Functional Simulation Netlist. A simulation run is

started by Processing >Start Simulation, or by using the icon 🔧. At the end of the simulation,

Quartus II software indicates its successful completion and displays a Simulation Report illustrated in Figure 44. If your report window does not show the entire simulation time range, click on the report window to select it and choose View > Fit in Window. Observe that the output f is as specified in the truth table of Figure 18.
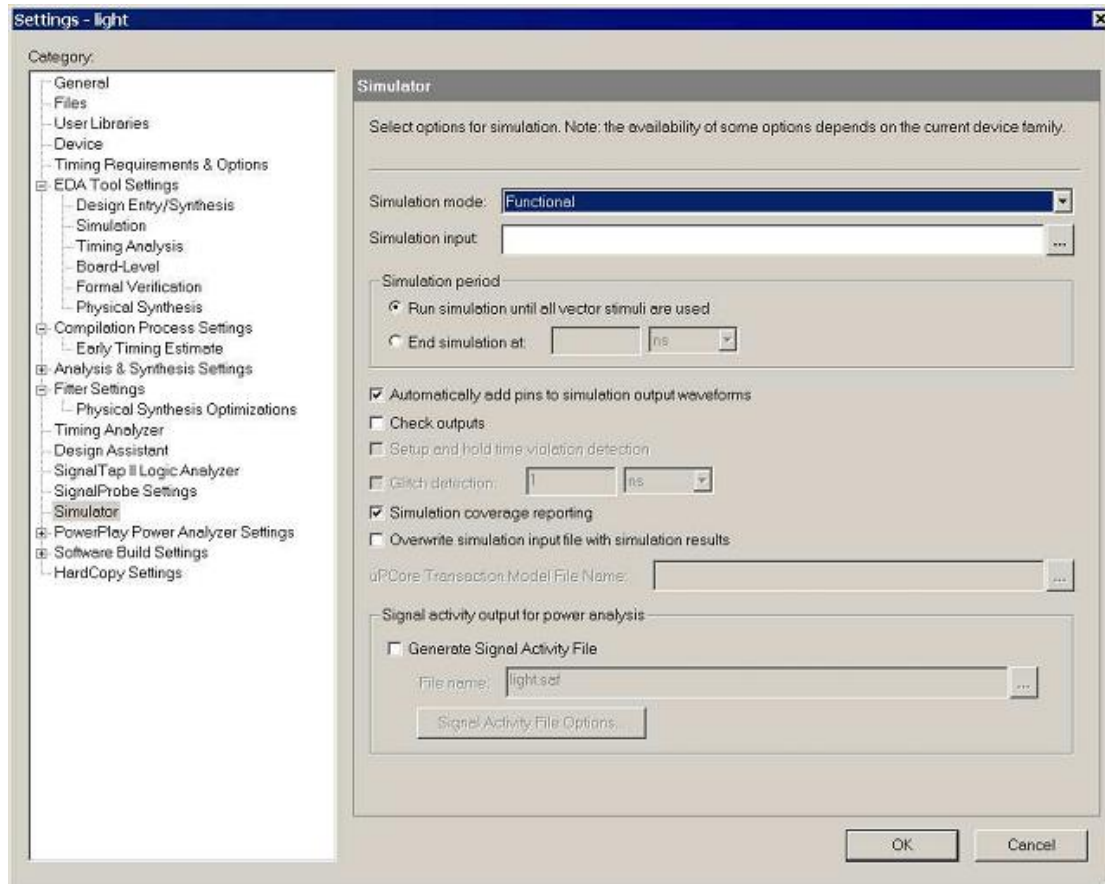
Figure 43. Specifying the simulation mode.
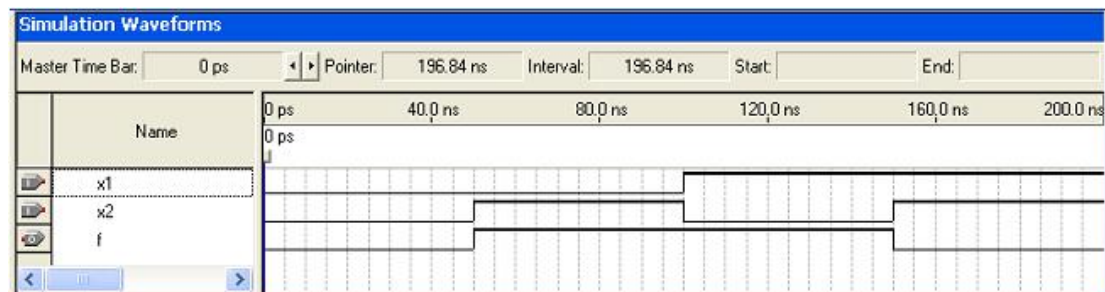
Figure 44.   The result of functional simulation.

6.1.2 Timing Simulation

Having ascertained that the designed circuit is functionally correct, we should now perform the timing simulation to see how it will behave when it is actually implemented in the chosen FPGA device. Select Assignments >Settings > Simulator to get to the window in Figure 43, choose Timing as the simulation mode, and click OK.Run the simulator, which should produce the waveforms in Figure 45. Observe that there is a delay of about 6ns in producing a change in the signal f from the time when the input signals, x1 and x2, change their values.This delay is due to the propagation delays in the logic element and the wires in the FPGA device. You may also notice that a momentary change in the value of f, from 1 to 0 and back to 1, occurs at about 106-ns point in the simulation. This glitch is also due to the propagation delays in the FPGA device, because changes in x1 and x2 may not arrive at exactly the same time at the logic element that generates f.
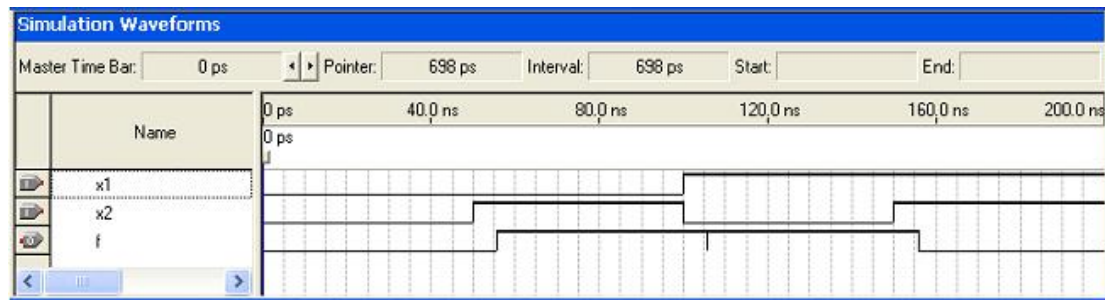
Figure 45. The result of timing simulation.

7. Programming and Configuring the FPGA Device

The FPGA device must be programmed and configured to implement the designed circuit. The required configura-tion file is generated by the Quartus II Compiler's Assembler module. Altera's DE2 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 DE2 Board 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 DE2 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 DE2 board. The RUN position selects the JTAG mode, while the PROG position selects the AS mode.

7.1 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 46. 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 47.
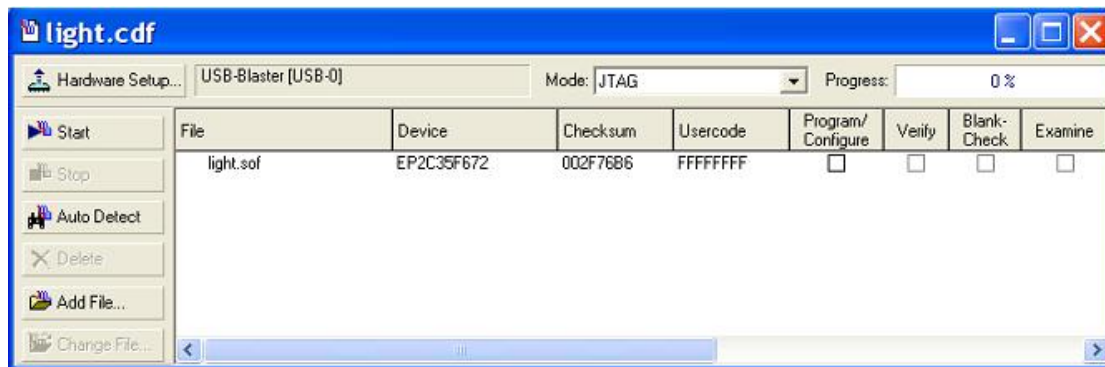
Figure 46. The Programmer window.

Observe that the configuration file light.sof is listed in the window in Figure 46. 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 48.
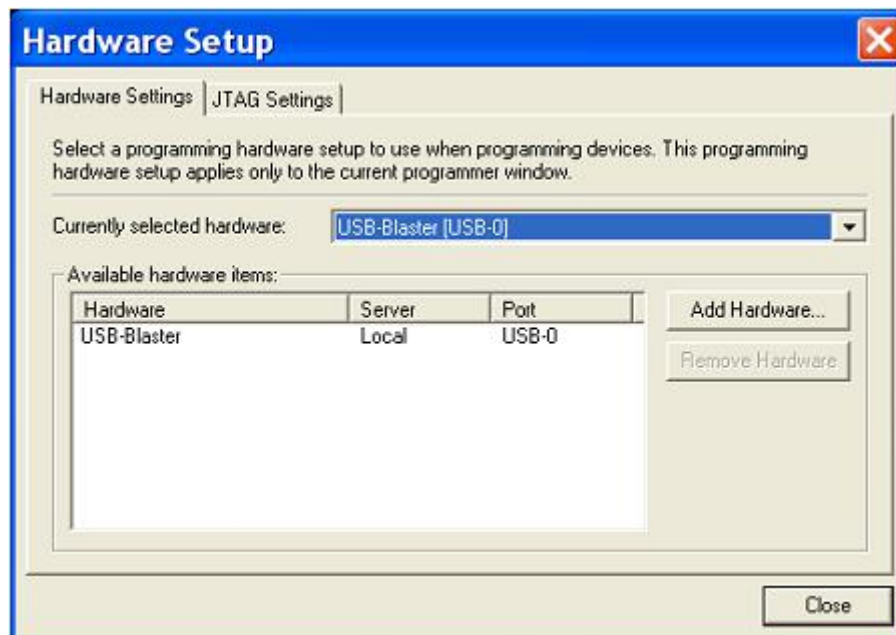
Figure 47. The Hardware Setup window.
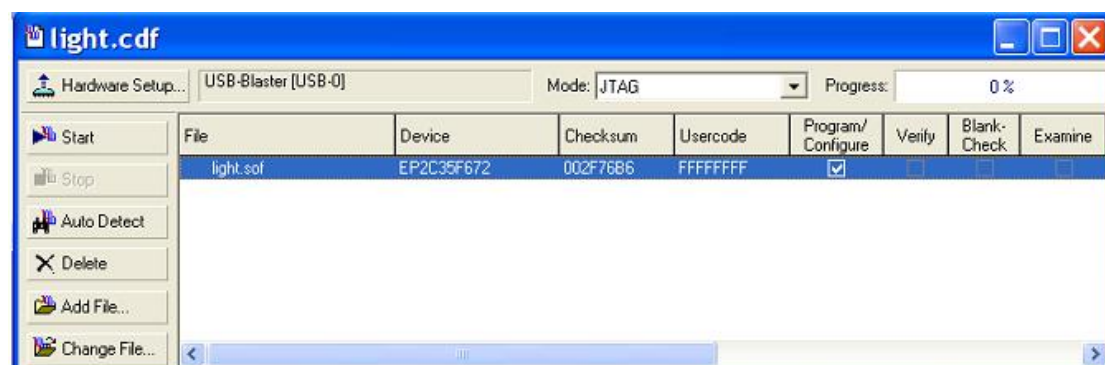
Figure 48. The updated Programmer window.

Now, press Start in the window in Figure 48. 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.

7.2 Active Serial Mode Programming

In this case, the configuration data has to be loaded into the configuration device on the DE2 board, which is identified by the name EPCS16. To specify the required configuration device select Assignments > Device, which leads to the window in Figure 49. Click on the Device & Pin Options button to reach the window in Figure 50. Now, click on the Configuration tab to obtain the window in Figure 51. In the Configuration device box (which may be set to Auto) choose EPCS16 and click OK. Upon returning to the window in Figure 49, click OK. Recompile the designed circuit.

The rest of the procedure is similar to the one described above for the JTAG mode. Select Tools > Program-mer to reach the window in Figure 46. 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 52 will appear, asking if you want to clear all devices. Click Yes. Now, the Programmer window shown in Figure 53 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 54 will appear. Select the file light.pof in the directory introtutorial 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 EPCS16 configuration device.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 55.

Flip the RUN/PROG switch on the DE2 board to the PROG position. Press Start in the window in Figure 55. An LED on the board will light up when the configuration data has been downloaded successfully. Also, the Progress box in Figure 55 will indicate when the configuration and programming process is completed, as shown
in Figure 56.
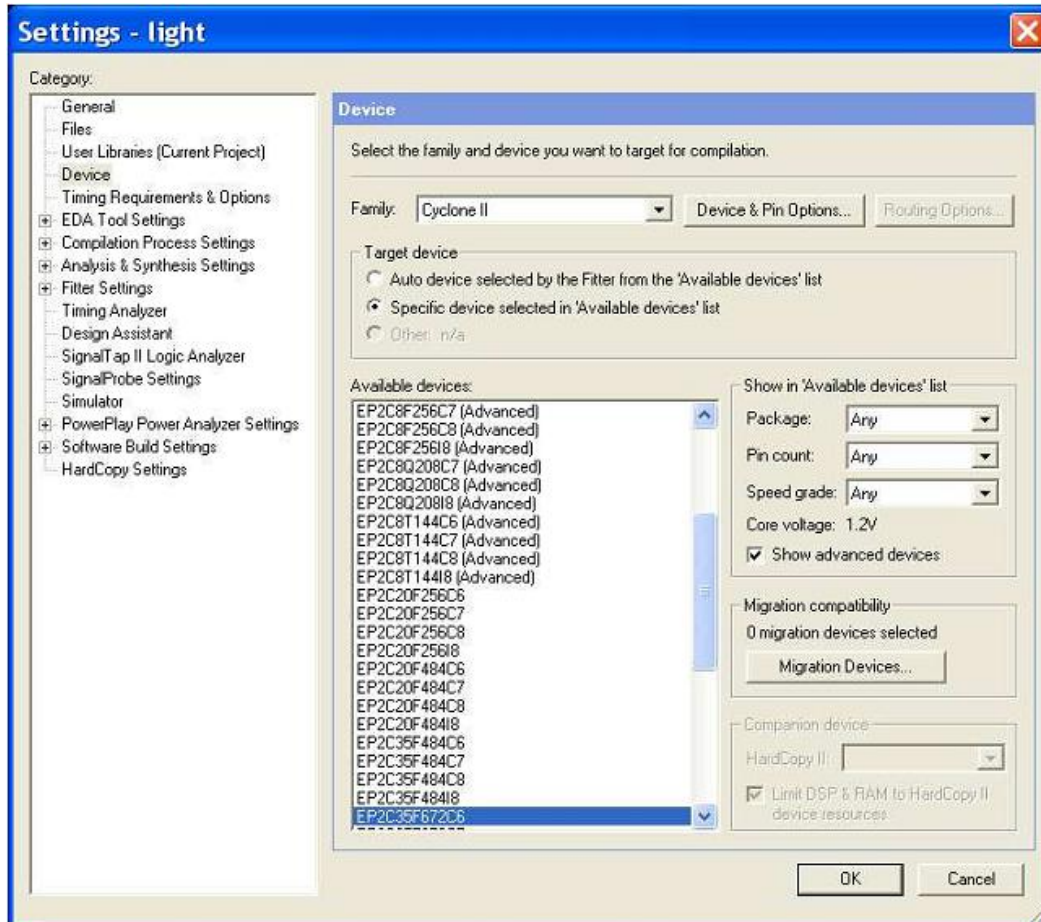
Figure 49. The Device Settings window.
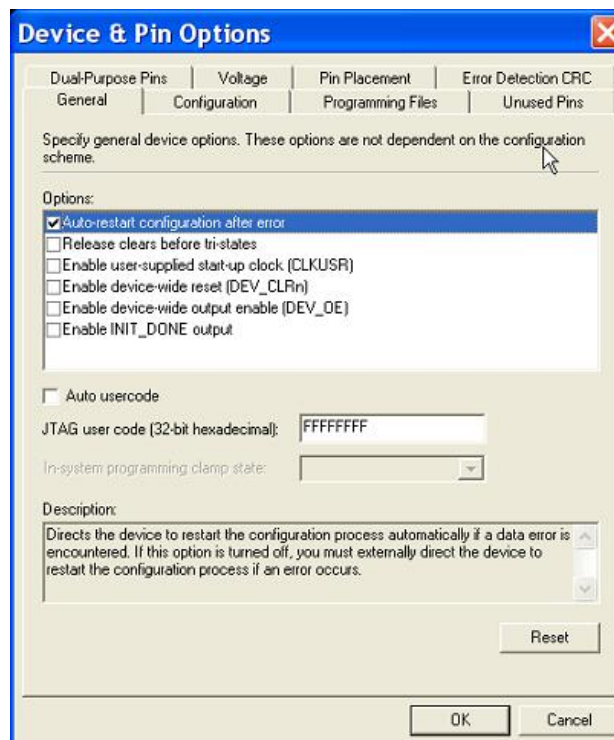
Figure 50. The Options window.
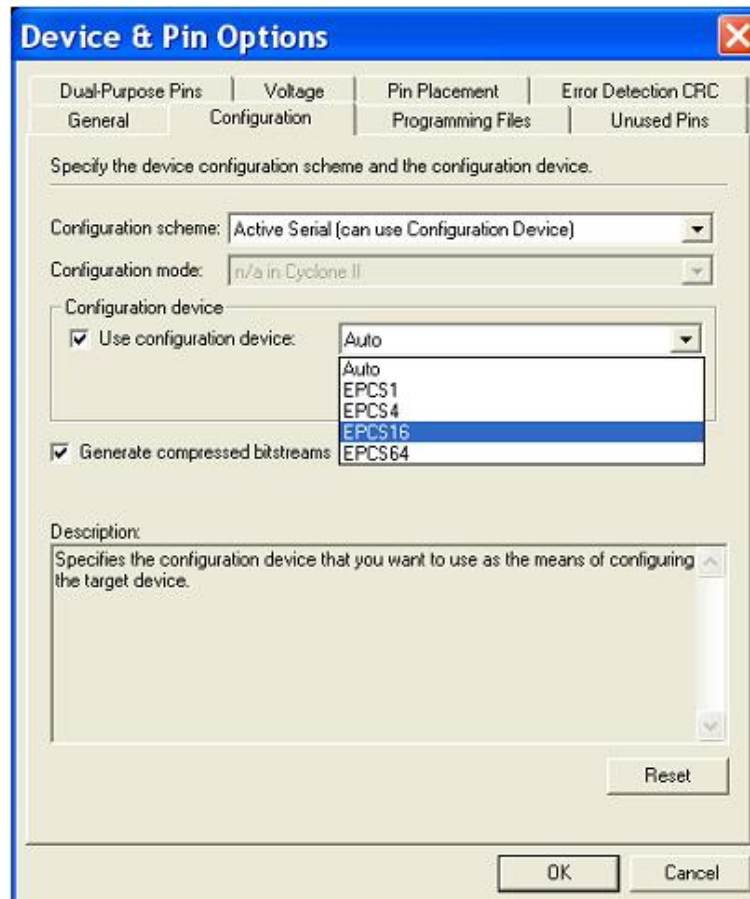
Figure 51. Specifying the configuration device.
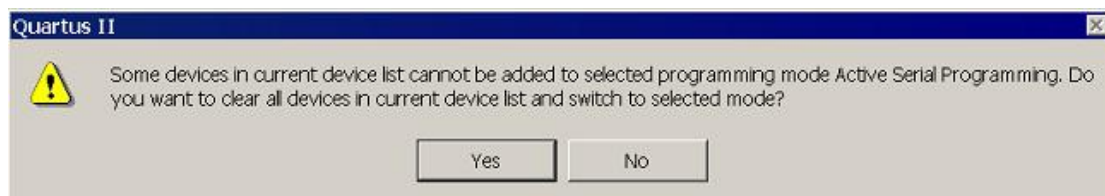
Figure 52. Clear the previously selected devices.
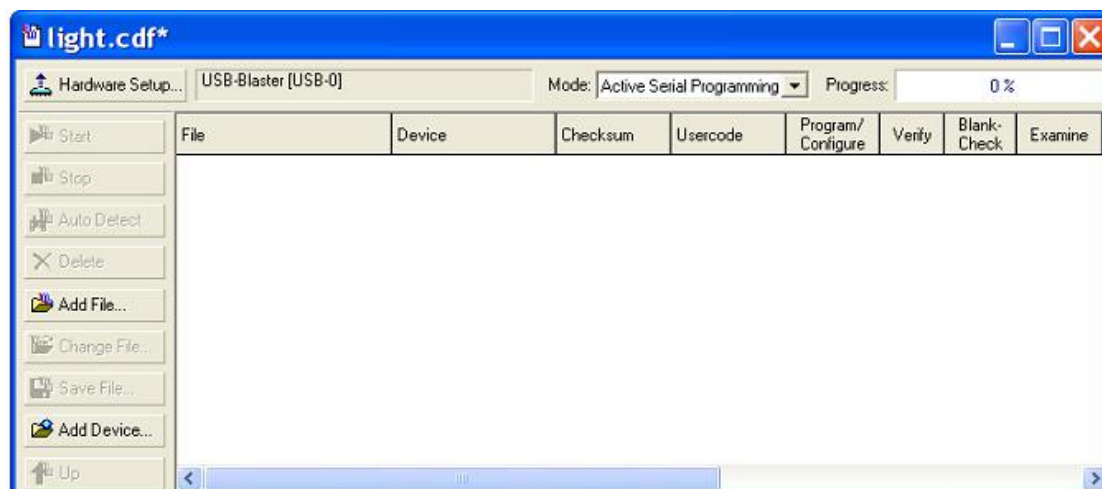
Figure 53. The Programmer window with Active Serial Programming selected.
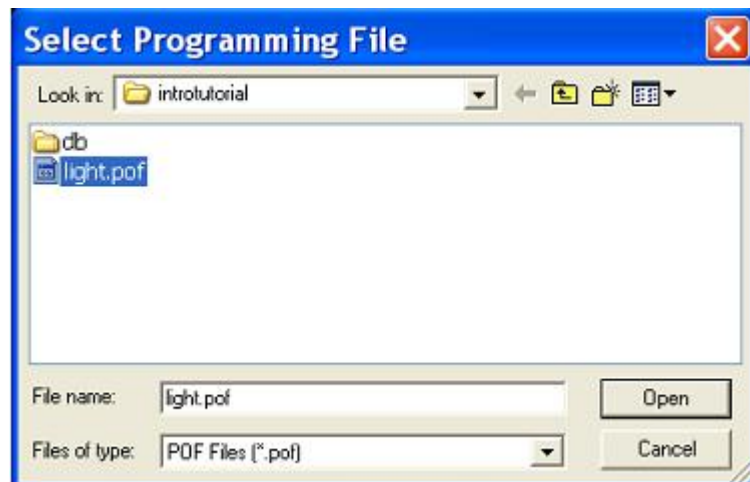
Figure 54. Choose the configuration file.
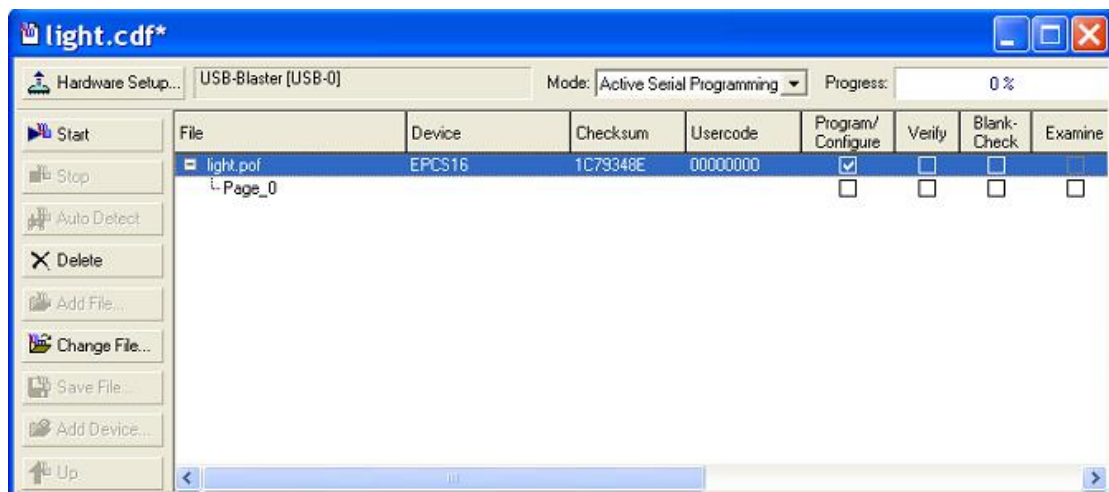
Figure 55. The updated Programmer window.
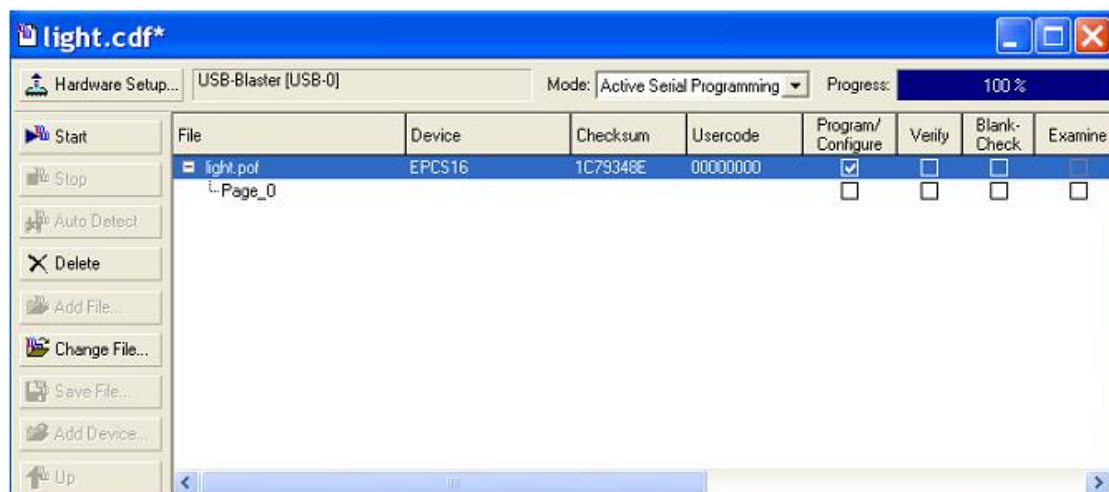
Figure 56. The Programmer window upon completion of programming.

8. 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 x1 and x2, by setting the corresponding states of the switches SW1 and SW0. Verify that the circuit implements the truth table in Figure18.

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.

Laboratory Exercise 1：Switches, Lights, and Multiplexers

The purpose of this exercise is to learn how to connect simple input and output devices to an FPGA chip and implement a circuit that uses these devices. We will use the switches SW17−0 on the DE2 board as inputs to the circuit. We will use light emitting diodes (LEDs) and 7-segment displays as output devices.

Part I

The DE2 board provides 18 toggle switches, called SW17−0, that can be used as inputs to a circuit, and 18 red lights, called LEDR17−0, that can be used to display output values. The codes shows a simple VHDL entity that uses these switches and shows their states on the LEDs. Since there are 18 switches and lights it is convenient to represent them as arrays in the VHDL code, as shown. We have used a single assignment statement for all 18 LEDR outputs, which is equivalent to the individual assignments

LEDR(17) <= SW(17);

LEDR(16) <= SW(16);

...

LEDR(0) <= SW(0);

The DE2 board has hardwired connections between its FPGA chip and the switches and lights. To use SW17−0 and LEDR17−0 it is necessary to include in your Quartus II project the correct pin assignments, which are given in the DE2 User Manual. For example, the manual specifies that SW0 is connected to the FPGA pin N25 and LEDR0 is connected to pin AE23. A good way to make the required pin assignments is to import into the Quartus II software the file called DE2 pin assignments.csv, which is provided on the DE2 System CD and in the University Program section of Altera's web site. The procedure for making pin assignments is described in the tutorial Quartus II Introduction using VHDL Design, which is also available from Altera.

It is important to realize that the pin assignments in the DE2 pin assignments.csv file are useful only if the pin names given in the file are exactly the same as the port names used in your VHDL entity. The file uses the names SW[0] ... SW[17] and LEDR[0] ... LEDR[17] for the switches and lights, which is the reason we used these names in the codes (note that the Quartus II software uses [ ] square brackets for array elements, while the VHDL syntax uses ( ) round brackets).

LIBRARY ieee;

USE ieee.std logic 1164.all;

- - Simple module that connects the SW switches to the LEDR lights

ENTITY part1 IS

PORT (SW : IN STD LOGIC VECTOR(17 DOWNTO 0);

LEDR : OUT STD LOGIC VECTOR(17 DOWNTO 0)); - - red LEDs

END part1;

ARCHITECTURE Behavior OF part1 IS

BEGIN

LEDR <=SW;

END Behavior

Perform the following steps to implement a circuit corresponding to the code on the DE2 board.

1. Create a new Quartus II project for your circuit. Select Cyclone II EP2C35F672C6 as the target chip, which is the FPGA chip on the Altera DE2 board.

2. Create a VHDL entity for the code and include it in your project.

3. Include in your project the required pin assignments for the DE2 board, as discussed above. Compile the project.

4. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit by toggling the switches and observing the LEDs.

Part II

Figure 57a shows a sum-of-products circuit that implements a 2-to-1 multiplexer with a select input s.If s =0 the multiplexer's output m is equal to the input x, and if s =1 the output is equal to y. Part b of the figure gives a truth table for this multiplexer, and part c shows its circuit symbol.

a) Circuit

b) Truth table   c) Symbol
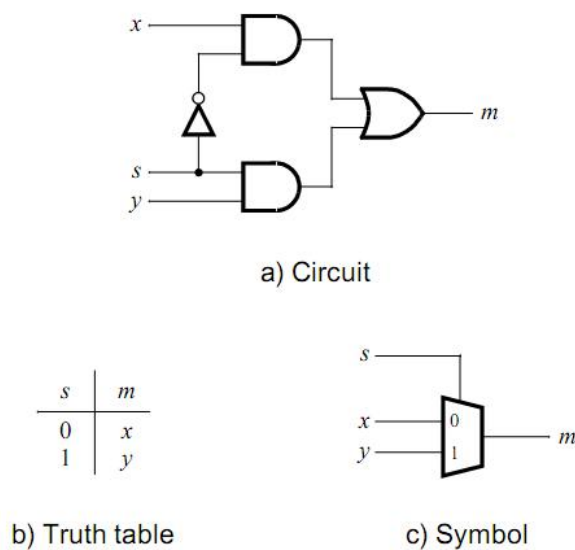
Figure 57. A 2-to-1 multiplexer.

The multiplexer can be described by the following VHDL statement:

m <= (NOT (s) AND x) OR (s AND y);

You are to write a VHDL entity that includes eight assignment statements like the one shown above to describe the circuit given in Figure 58a. This circuit has two eight-bit inputs, X and Y , and produces the eight-bit outputM. If s =0 thenM = X, while if s =1 thenM = Y . We refer to this circuit as an eight-bit wide 2-to-1 multiplexer. It has the circuit symbol shown in Figure 58b, in which X, Y , and M are depicted as eight-bit wires. Perform the steps shown below.

a) Circuit   b) Symbol
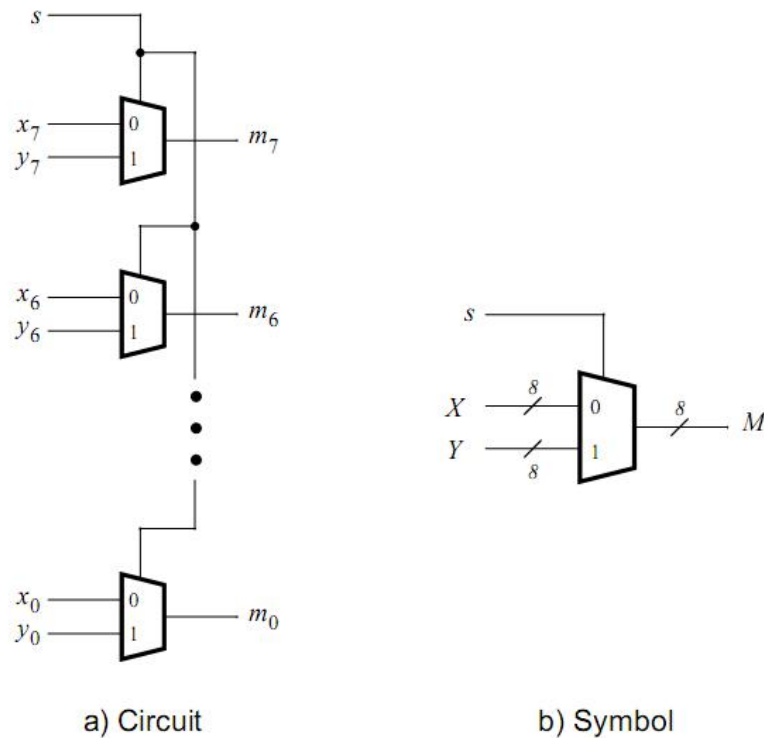
Figure 58. An eight-bit wide 2-to-1 multiplexer.

1. Create a new Quartus II project for your circuit.

2. Include your VHDL file for the eight-bit wide 2-to-1 multiplexer in your project. Use switch SW 17 on the DE2 board as the s input, switches SW7−0 as the X input and SW15−8 as the Y input. Connect the SW switches to the red lights LEDR and connect the outputM to the green lights LEDG7−0.

3. Include in your project the required pin assignments for the DE2 board. As discussed in Part I, these assignments ensure that the input ports of your VHDL code will use the pins on the Cyclone II FPGA that are connected to the SW switches, and the output ports of your VHDL code will use the FPGA pins connected to the LEDR and LEDG lights.

4. Compile the project.

5. Download the compiled circuit into the FPGA chip. Test the functionality of the eight-bit wide 2-to-1 multiplexer by toggling the switches and observing the LEDs.

Part III

In Figure 57 we showed a 2-to-1 multiplexer that selects between the two inputs x and y. For this part consider a circuit in which the output m has to be selected from five inputs u, v, w, x, and y. Part a of Figure 58 shows how we can build the required 5-to-1 multiplexer by using four 2-to-1multiplexers. The circuit uses a 3-bit select input

$s_2s_1s_0$ and implements the truth table shown in Figure 59b. A circuit symbol for this multiplexer is given in part c of the figure.

Recall from Figure 58 that an eight-bit wide 2-to-1 multiplexer can be built by using eight instances of a 2-to-1 multiplexer. Figure 60 applies this concept to define a three-bit wide 5-to-1 multiplexer. It contains three instances of the circuit in Figure 59a.
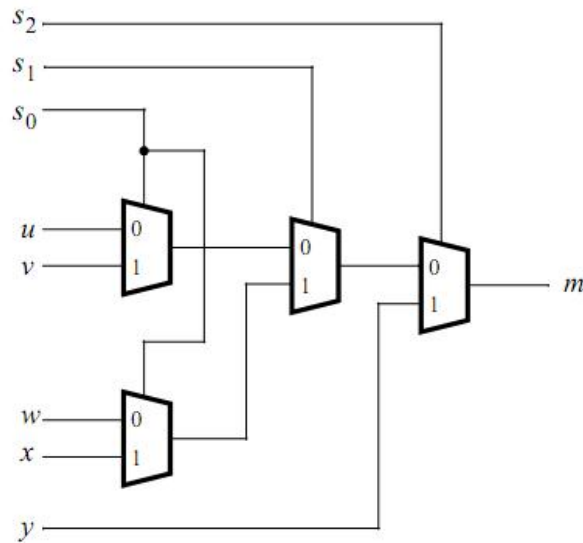
a) Circuit

s	m
0	x
1	y

b) Truth table
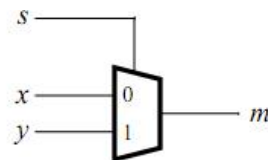
c) Symbol

Figure 59. A 5-to-1 multiplexer.
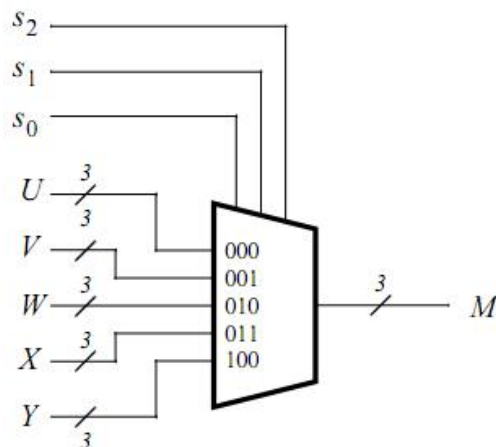
Figure 60. A three-bit wide 5-to-1 multiplexer.

Perform the following steps to implement the three-bit wide 5-to-1 multiplexer.

1. Create a new Quartus II project for your circuit.

2. Create a VHDL entity for the three-bit wide 5-to-1 multiplexer. Connect its select inputs to switches SW17−15, and use the remaining 15 switches SW14−0 to provide the five 3-bit inputs U to Y . Connect the SW switches to the red lights LEDR and connect the outputM to the green lights LEDG2−0.

3. Include in your project the required pin assignments for the DE2 board. Compile the project.

4. Download the compiled circuit into the FPGA chip. Test the functionality of the three-bit wide 5-to-1 multiplexer by toggling the switches and observing the LEDs. Ensure that each of the

inputs U to Y can be properly selected as the outputM.

Part IV

Figure 61 shows a 7-segment decoder module that has the three-bit input c 2c1c0. This decoder produces seven outputs that are used to display a character on a 7-segment display. Table 1 lists the characters that should be displayed for each valuation of c2c1c0. To keep the design simple, only four characters are included in the table
(plus the 'blank' character, which is selected for codes $100 - 111$).
The seven segments in the display are identified by the indices 0 to 6 shown in the figure. Each segment is illuminated by driving it to the logic value 0. You are to write a VHDL entity that implements logic functions that represent circuits needed to activate each of the seven segments. Use only simple VHDL assignment statements
in your code to specify each logic function using a Boolean expression.
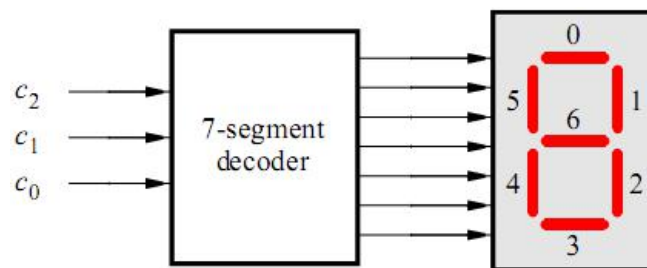
Figure 61. A 7-segment decoder.

$c_2 c_1 c_0$	Character
000	H
001	E
010	L
011	O
100	
101	
110	
111	

Table 1. Character codes.

Perform the following steps:
1. Create a new Quartus II project for your circuit.
2. Create a VHDL entity for the 7-segment decoder. Connect the c2c1c0 inputs to switches SW2−0, and connect the outputs of the decoder to the HEX0 display on the DE2 board. The segments in this display are called HEX00, HEX01, ..., HEX06, corresponding to Figure 61. You should declare the 7-bit port HEX0 : OUT STD LOGIC VECTOR(0 TO 6); in your VHDL code so that the names of these outputs match the corresponding names in the DE2 User Manual and the DE2 pin assignments.csv file.
3. After making the required DE2 board pin assignments, compile the project.
4. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit by toggling the SW2−0 switches and observing the 7-segment display.

Part V

Consider the circuit shown in Figure 62. It uses a three-bit wide 5-to-1 multiplexer to enable the selection of five characters that are displayed on a 7-segment display. Using the 7-segment decoder from Part IV this circuit can display any of the characters H, E, L, O, and 'blank'. The character codes are set according to Table 1 by using the switches SW14−0, and a specific character is selected for display by setting the switches SW17−15.

An outline of the VHDL code that represents this circuit is provided in Figure 63. Note that we have used the circuits from Parts III and IV as subcircuits in this code. You are to extend the code in Figure 63 so that it uses five 7-segment displays rather than just one. You will need to use five instances of each of the subcircuits. The purpose of your circuit is to display any word on the five displays that is composed of the characters in Table 1, and be able to rotate this word in a circular fashion across the displays when the switches SW17−15 are toggled. As an example, if the displayed word is HELLO, then your circuit should produce the output patterns illustrated in Table 2.
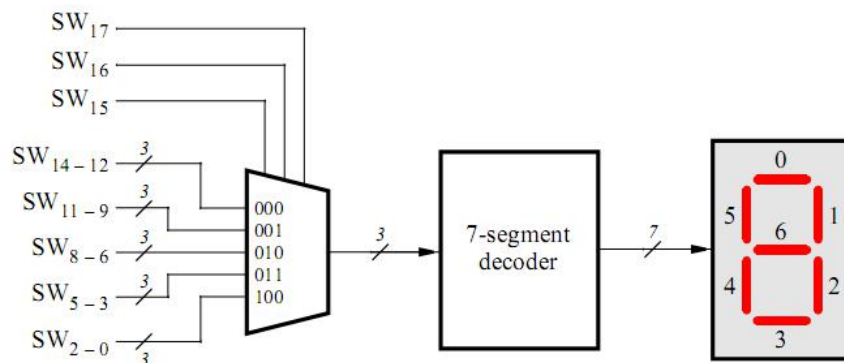
Figure 62. A circuit that can select and display one of five characters.

LIBRARY ieee;

USE ieee.std logic 1164.all;

ENTITY part5 IS

PORT (SW : IN STD LOGIC VECTOR(17 DOWNTO 0);

HEX0 : OUT STD LOGIC VECTOR(0 TO 6));

END part5;

ARCHITECTURE Behavior OF part5 IS

    COMPONENT mux 3bit 5to1

        PORT(S,U,V,W,X,Y :IN STD LOGIC VECTOR(2 DOWNTO 0);

            M : OUT STD LOGIC VECTOR(2 DOWNTO 0));

    END COMPONENT;

    COMPONENT char 7seg

        PORT (C : IN STD LOGIC VECTOR(2 DOWNTO 0);

            Display : OUT STD LOGIC VECTOR(0 TO 6));

    END COMPONENT;

    SIGNAL M : STD LOGIC VECTOR(2 DOWNTO 0);

    BEGIN

        M0: mux 3bit 5to1 PORT MAP (SW(17 DOWNTO 15), SW(14 DOWNTO 12),

SW(11 DOWNTO 9),

SW(8 DOWNTO 6), SW(5 DOWNTO 3), SW(2 DOWNTO 0), M);

H0: char 7seg PORT MAP (M, HEX0);

END Behavior;

LIBRARY ieee;

USE ieee.std logic 1164.all;

- - implements a 3-bit wide 5-to-1 multiplexer

ENTITY mux 3bit 5to1 IS

PORT (S, U, V, W, X, Y : IN STD LOGIC VECTOR(2 DOWNTO 0);

M : OUT STD LOGIC VECTOR(2 DOWNTO 0));

END mux 3bit 5to1;

ARCHITECTURE Behavior OF mux 3bit 5to1 IS

... code not shown

END Behavior;

LIBRARY ieee;

USE ieee.std logic 1164.all;

ENTITY char 7seg IS

PORT (C : IN STD LOGIC VECTOR(2 DOWNTO 0);

Display : OUT STD LOGIC VECTOR(0 TO 6));

END char 7seg;

ARCHITECTURE Behavior OF char 7seg IS

... code not shown

END Behavior;

SW_{17} SW_{16} SW_{15}	Character pattern				
000	H	E	L	L	O
001	E	L	L	O	H
010	L	L	O	H	E
011	L	O	H	E	L
100	O	H	E	L	L

Table 2. Rotating the word HELLO on five displays.

Perform the following steps.

1. Create a new Quartus II project for your circuit.

2. Include your VHDL entity in the Quartus II project. Connect the switches SW17−15 to the select inputs of each of the five instances of the three-bit wide 5-to-1 multiplexers. Also connect SW14−0 to each instance of the multiplexers as required to produce the patterns of characters shown in Table 2. Connect the outputs of the five multiplexers to the 7-segment displays HEX4, HEX3, HEX2, HEX1, and HEX0.

3. Include the required pin assignments for the DE2 board for all switches, LEDs, and 7-segment displays.Compile the project.

4. Download the compiled circuit into the FPGA chip. Test the functionality of the circuit by setting the proper character codes on the switches SW14−0 and then toggling SW17−15 to observe the rotation of the characters.

Laboratory Exercise 2：Numbers and Displays

This is an exercise in designing combinational circuits that can perform binary-to-decimal number conversion and binary-coded-decimal (BCD) addition.

Part I

We wish to display on the 7-segment displays HEX3 to HEX0 the values set by the switches SW15−0. Let the values denoted by SW15−12, SW11−8, SW7−4 and SW3−0 be displayed on HEX3, HEX2, HEX1 and HEX0, respectively. Your circuit should be able to display the digits from 0 to 9, and should treat the valuations 1010 to
1111 as don't-cares.
1. Create a new project which will be used to implement the desired circuit on the Altera DE2 board. The intent of this exercise is to manually derive the logic functions needed for the 7-segment displays. You should use only simple VHDL assignment statements in your code and specify each logic function as a Boolean expression.
2. Write a VHDL file that provides the necessary functionality. Include this file in your project and assign the pins on the FPGA to connect to the switches and 7-segment displays, as indicated in the UserManual for the DE2 board. The procedure for making pin assignments is described in the tutorial Quartus II Introduction using VHDL Design, which is available on the DE2 System CD and in the University Program section of Altera's web site.
3. Compile the project and download the compiled circuit into the FPGA chip.
4. Test the functionality of your design by toggling the switches and observing the displays.

Part II

You are to design a circuit that converts a four-bit binary number $V = v_3v_2v_1v_0$ into its two-digit decimal equiv-alent $D = d_1d_0$. Table 3 shows the required output values. A partial design of this circuit is given. It includes a comparator that checks when the value of V is greater than 9, and uses the output of this comparator
in the control of the 7-segment displays. You are to complete the design of this circuit by creating a VHDL entity which includes the comparator, multiplexers, and circuit A (do not include circuit B or the 7-segment decoder at this point). Your VHDL entity should have the four-bit input V , the four-bit output M and the output z. The intent of this exercise is to use simple VHDL assignment statements to specify the required logic functions using Boolean expressions. Your VHDL code should not include any IF-ELSE, CASE, or similar statements.

Binary value	Decimal digits	
0000	0	0
0001	0	1
0010	0	2
...	...	...
1001	0	9
1010	1	0
1011	1	1
1100	1	2
1101	1	3
1110	1	4
1111	1	5

Table 3. Binary-to-decimal conversion values.

Perform the following steps:

1. Make a Quartus II project for your VHDL entity.

2. Compile the circuit and use functional simulation to verify the correct operation of your comparator, multi-plexers, and circuit A.

3. Augment your VHDL code to include circuit B in Figure 63 as well as the 7-segment decoder. Change the inputs and outputs of your code to use switches SW3−0 on the DE2 board to represent the binary number V , and the displays HEX1 and HEX0 to show the values of decimal digits d1 and d0. Make sure to include in your project the required pin assignments for the DE2 board.

4. Recompile the project, and then download the circuit into the FPGA chip.

5. Test your circuit by trying all possible values of V and observing the output displays.
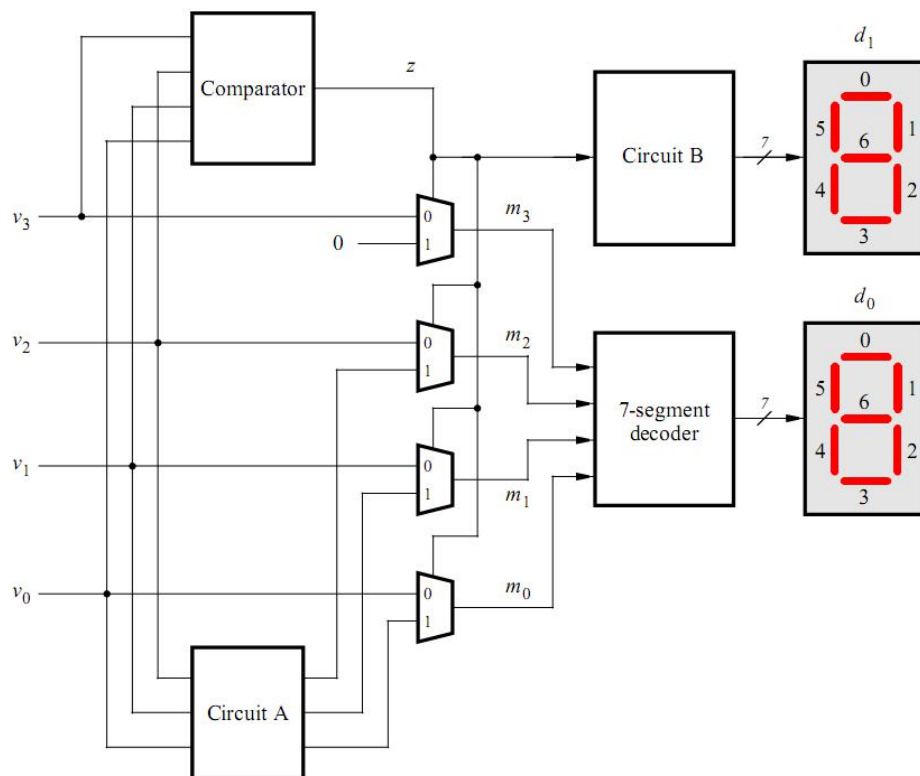
Figure 63. Partial design of the binary-to-decimal conversion circuit.

Part III

Figure 64a shows a circuit for a full adder, which has the inputs a, b, and ci, and produces the outputs s and co.Parts b and c of the figure show a circuit symbol and truth table for the full adder, which produces the two-bit binary sum $cos = a + b + ci$. Figure 64d shows how four instances of this full adder entity can be used to design a circuit that adds two four-bit numbers. This type of circuit is usually called a ripple-carry adder, because of the way that the carry signals are passed from one full adder to the next. Write VHDL code that implements this circuit, as described below.

a) Full adder circuit   b) Full adder symbol

b a c_i	c_o s
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

d) Four-bit ripple-carry adder circuit   c) Full adder truth table
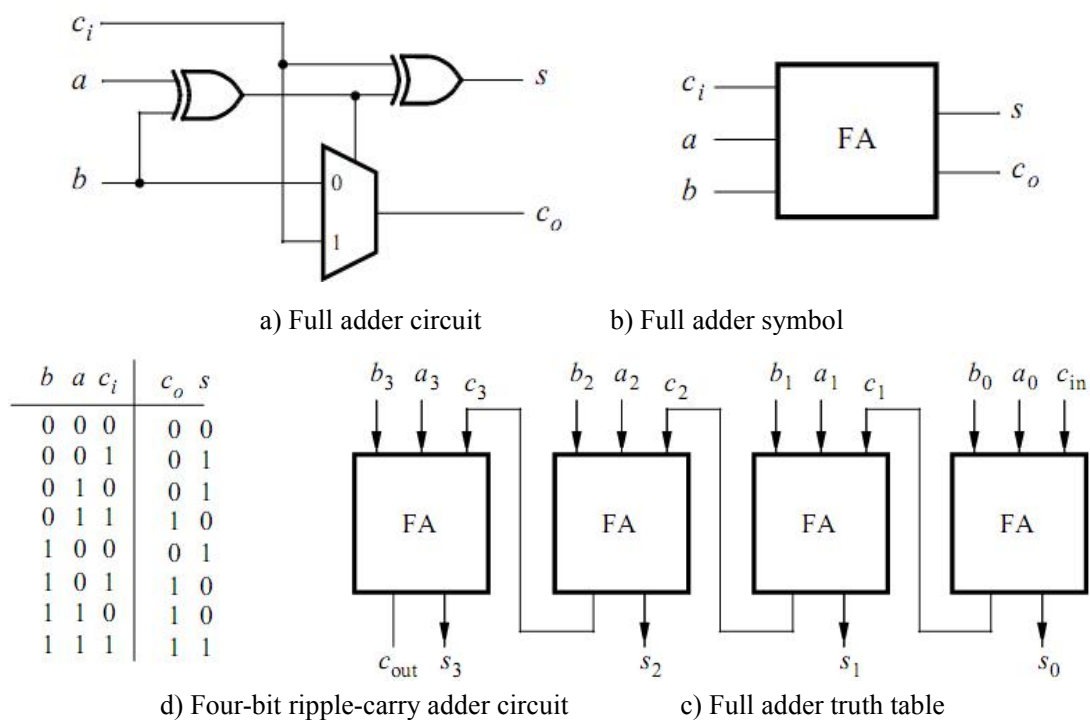
Figure 64. A ripple-carry adder circuit.

1. Create a new Quartus II project for the adder circuit. Write a VHDL entity for the full adder subcircuit and write a top-level VHDL entity that instantiates four instances of this full adder.
2. Use switches SW7−4 and SW3−0 to represent the inputs A and B, respectively. Use SW8 for the carry-in cin of the adder. Connect the SW switches to their corresponding red lights LEDR, and connect the outputs of the adder, cout and S, to the green lights LEDG.
3. Include the necessary pin assignments for the DE2 board, compile the circuit, and download it into the FPGA chip.
4. Test your circuit by trying different values for numbers A, B, and c_{in}.

Part IV

In part II we discussed the conversion of binary numbers into decimal digits. It is sometimes useful to build circuits that use this method of representing decimal numbers, in which each decimal digit is represented using four bits. This scheme is known as the binary coded decimal (BCD) representation. As an example, the decimal

value 59 is encoded in BCD form as 0101 1001.

You are to design a circuit that adds two BCD digits. The inputs to the circuit are BCD numbers A and B, plus a carry-in, cin. The output should be a two-digit BCD sum S1S0. Note that the largest sum that needs to be handled by this circuit is S1S0 =9+9+1=19. Perform the steps given below.

1. Create a new Quartus II project for your BCD adder. You should use the four-bit adder circuit from part III to produce a four-bit sum and carry-out for the operation A + B. A circuit that converts this five-bit result, which has the maximum value 19, into two BCD digits S1S0 can be designed in a very similar way as the binary-to-decimal converter from part II. Write your VHDL code using simple assignment statements to

specify the required logic functions–do not use other types of VHDL statements such as IF-ELSE or CASE statements for this part of the exercise.

2. Use switches SW7−4 and SW3−0 for the inputs A and B, respectively, and use SW8 for the carry-in. Connect the SW switches to their corresponding red lights LEDR, and connect the four-bit sum and carry-out produced by the operation A + B to the green lights LEDG. Display the BCD values of A and B on the 7-segment displays HEX6 and HEX4, and display the result S1S0 on HEX1 and HEX0.

3. Since your circuit handles only BCD digits, check for the cases when the input A or B is greater than nine.If this occurs, indicate an error by turning on the green light LEDG8.

4. Include the necessary pin assignments for the DE2 board, compile the circuit, and download it into the FPGA chip.

5. Test your circuit by trying different values for numbers A, B, and c_{in}.

Part V

Design a circuit that can add two 2-digit BCD numbers, A1A0 and B1B0 to produce the three-digit BCD sum S2S1S0. Use two instances of your circuit from part IV to build this two-digit BCD adder. Perform the steps below:

1. Use switches SW15−8 and SW7−0 to represent 2-digit BCD numbers A1A0 and B1B0, respectively. The value of A1A0 should be displayed on the 7-segment displays HEX7 and HEX6, while B1B0 should be on HEX5 and HEX4. Display the BCD sum, S2S1S0, on the 7-segment displays HEX2, HEX1 and HEX0.

2. Make the necessary pin assignments and compile the circuit.

3. Download the circuit into the FPGA chip, and test its operation.

Part VI

In part V you created VHDL code for a two-digit BCD adder by using two instances of the VHDL code for a one-digit BCD adder from part IV. A different approach for describing the two-digit BCD adder in VHDL code is to specify an algorithm like the one represented by the following pseudo-code:

1 T0 = A0 + B0

2if(T0 > 9) then

3 Z0 =10;

4 $c_1 = 1$;

5 else

6 $Z_0 = 0$;

7 $c_1 = 0$;

8 end if

9 $S_0 = T_0 - Z_0$

10 $T_1 = A_1 + B_1 + c_1$

11 if $(T_1 > 9)$ then

12 $Z_1 = 10$;

13 $c_2 = 1$;

14 else

15 $Z_1 = 0$;

16 $c_2 = 0$;

17 end if

18 $S_1 = T_1 - Z_1$

19 $S_2 = c_2$

It is reasonably straightforward to see what circuit could be used to implement this pseudo-code. Lines 1, 9, 10,and 18 represent adders, lines 2-8 and 11-17 correspond to multiplexers, and testing for the conditions $T_0 > 9$ and $T_1 > 9$ requires comparators. You are to write VHDL code that corresponds to this pseudo-code. Note that you can perform addition operations in your VHDL code instead of the subtractions shown in lines 9 and 18. The intent of this part of the exercise is to examine the effects of relying more on the VHDL compiler to design the circuit by using IF-ELSE statements along with the VHDL > and + operators. Perform the following steps:

1. Create a new Quartus II project for your VHDL code. Use the same switches, lights, and displays as in part V. Compile your circuit.

2. Use the Quartus II RTL Viewer tool to examine the circuit produced by compiling your VHDL code. Com-pare the circuit to the one you designed in Part V.

3. Download your circuit onto the DE2 board and test it by trying different values for numbers $A_1 A_0$ and $B_1 B_0$.

Part VII

Design a combinational circuit that converts a 6-bit binary number into a 2-digit decimal number represented in the BCD form. Use switches SW_{5-0} to input the binary number and 7-segment displays HEX1 and HEX0 to display the decimal number. Implement your circuit on the DE2 board and demonstrate its functionality.

Laboratory Exercise 3：Clocks and Timers

This is an exercise in implementing and using a real-time clock.

Part I

Implement a 3-digit BCD counter. Display the contents of the counter on the 7-segment displays, HEX2−0. Derive a control signal, from the 50-MHz clock signal provided on the Altera DE2 board, to increment the contents of the counter at one-second intervals. Use the pushbutton switch KEY0 to reset the counter to 0.

1. Create a new Quartus II project which will be used to implement the desired circuit on the DE2 board.

2. Write a VHDL file that specifies the desired circuit.

3. Include the VHDL file in your project and compile the circuit.

4. Simulate the designed circuit to verify its functionality.

5. Assign the pins on the FPGA to connect to the 7-segment displays and the pushbutton switch, as indicated in the User Manual for the DE2 board.

6. Recompile the circuit and download it into the FPGA chip.

7. Verify that your circuit works correctly by observing the display.

Part II

Design and implement a circuit on the DE2 board that acts as a time-of-day clock. It should display the hour (from 0 to 23) on the 7-segment displays HEX7−6, the minute (from 0 to 60) on HEX5−4 and the second (from 0 to 60)on HEX3−2. Use the switches SW15−0 to preset the hour and minute parts of the time displayed by the clock.

Part III

Design and implement on the DE2 board a reaction-timer circuit. The circuit is to operate as follows:

1. The circuit is reset by pressing the pushbutton switch KEY0.

2. After an elapsed time, the red light labeled LEDR0 turns on and a four-digit BCD counter starts counting in intervals of milliseconds. The amount of time in seconds from when the circuit is reset until LEDR 0 is turned on is set by switches SW7−0.

3. A person whose reflexes are being tested must press the pushbutton KEY3 as quickly as possible to turn the LED off and freeze the counter in its present state. The count which shows the reaction time will be displayed on the 7-segment displays HEX2-0.

Laboratory Exercise 4: Adders, Subtractors, and Multipliers

The purpose of this exercise is to examine arithmetic circuits that add, subtract, and multiply numbers. Each type of circuit will be implemented in two ways: first by writing VHDL code that describes the required function-ality, and second by making use of predefined subcircuits from Altera's library of parameterizedmodules (LPMs).

The results produced for various implementations will be compared, both in terms of the circuit structure and its speed of operation.

Part I

Consider again the four-bit ripple-carry adder circuit that was used in lab exercise 2; a diagram of this circuit is reproduced in Figure 65a. You are to create an 8-bit version of the adder and include it in the circuit shown in Figure 65b. Your circuit should be designed to support signed numbers in 2's-complement form, and the Overflow output should be set to 1 whenever the sum produced by the adder does not provide the correct signed value.Perform the steps shown below.
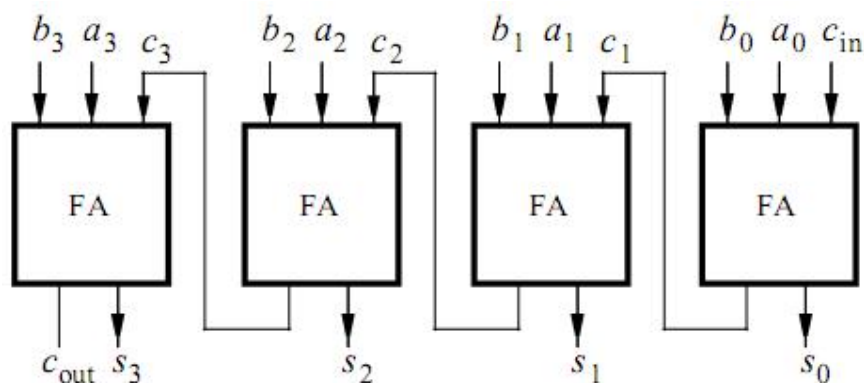
a) Four-bit ripple-carry adder circuit
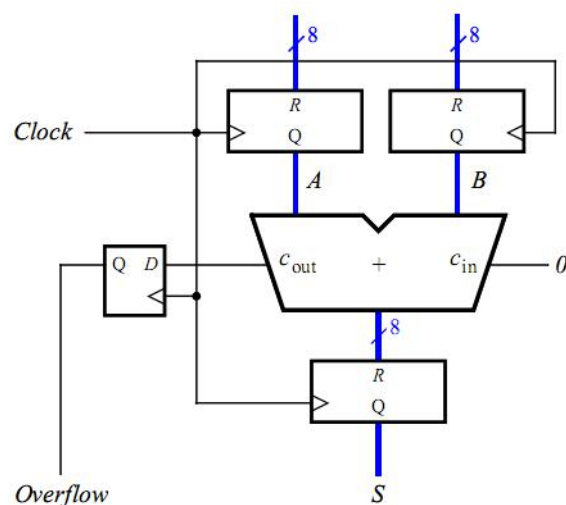
b) Eight-bit registered adder circuit

Figure 65. An 8-bit signed adder with registered inputs and outputs.

1. Make a new Quartus II project and write VHDL code that describes the circuit in Figure 65b. Use the circuit structure in Figure 65a to describe your adder.

2. Include the required input and output ports in your project to implement the adder circuit on the DE2 board.Connect the inputs A and B to switches SW15−8 and SW7−0, respectively. Use KEY0 as an active-low asynchronous reset input, and use KEY1 as a manual clock input. Display the sum outputs of the adder on the red LEDR7−0 lights and display the overflow output on the green LEDG8 light. The hexadecimal values of A and B should be shown on the displays HEX7-6 and HEX5-4, and the hexadecimal value of S should appear on HEX1-0.

3. Compile your code and use timing simulation to verify the correct operation of the circuit. Once the sim-ulation works properly, download the circuit onto the DE2 board and test it by using different values of A and B. Be sure to check for proper functionality of the Overflow output.

4. Open the Quartus II Compilation Report and examine the results reported by the Timing Analyzer. What is the maximum operating frequency, fmax, of your circuit? What is the longest path in the circuit in terms of delay?

Part II

Modify your circuit from Part I so that it can perform both addition and subtraction of eight-bit numbers. Use switch SW16 to specify whether addition or subtraction should be performed. Connect the other switches, lights, and displays as described for Part I.

1. Simulate your adder/subtractor circuit to show that it functions properly, and then download it onto the DE2 board and test it by using different switch settings.

2. Open the Quartus II Compilation Report and examine the results reported by the Timing Analyzer. What is the fmax of your circuit? What is the longest path in the circuit in terms of delay?

Part III

Repeat Part I using the predefined adder circuit called lpm add sub, instead of your ripple-carry adder structure from Figure 65. The lpm add sub module can be found in Altera's library of parameterized modules (LPMs), which is provided as part of the Quartus II system. The procedure for using these predefined modules in Quartus II projects is described in the tutorial Using Library Modules in VHDL Designs, which is available on the DE2 System CD and in the University Program section of Altera's web site.

1. Configure the lpm add sub module so that it performs only addition, to make the functionality comparable to Part I. Store your configuration of the lpm add sub module in the file lpm add8.v. After instantiating this module in your VHDL code, compile the project and use the Quartus II Chip Editor tool to examine some of the details of the implemented circuit.

One way to examine the adder subcircuit using the Chip Editor tool is illustrated in Figure 66. In the Quartus II Project Navigator window right-click on the part of your circuit hierarchy that

represents the lpm add8 subcircuit, and select the command Locate > Locate in Chip Editor. This opens the Chip Editor window shown in Figure 67. The logic elements in the Cyclone II FPGA that are used to implement the adder are highlighted in blue in the Chip Editor tool. Position your mouse pointer over any of these logic elements and double-click to open the Resource Property Editor window displayed in Figure 68. In the box labeled Node name you can select any of the nine logic elements that implement the adder module. The Resource Property Editor allows you to examine the contents of a logic element and to see how one logic element is connected others.

Figure 66. Locating the eight-bit adder in the Chip Editor tool.

Figure 67. The highlighted logic elements for the eight-bit adder.

Using the tools described above, and referencing the Data Sheet information for the Cyclone II FPGA,describe the eight-bit adder circuit implemented with the lpm add sub module.
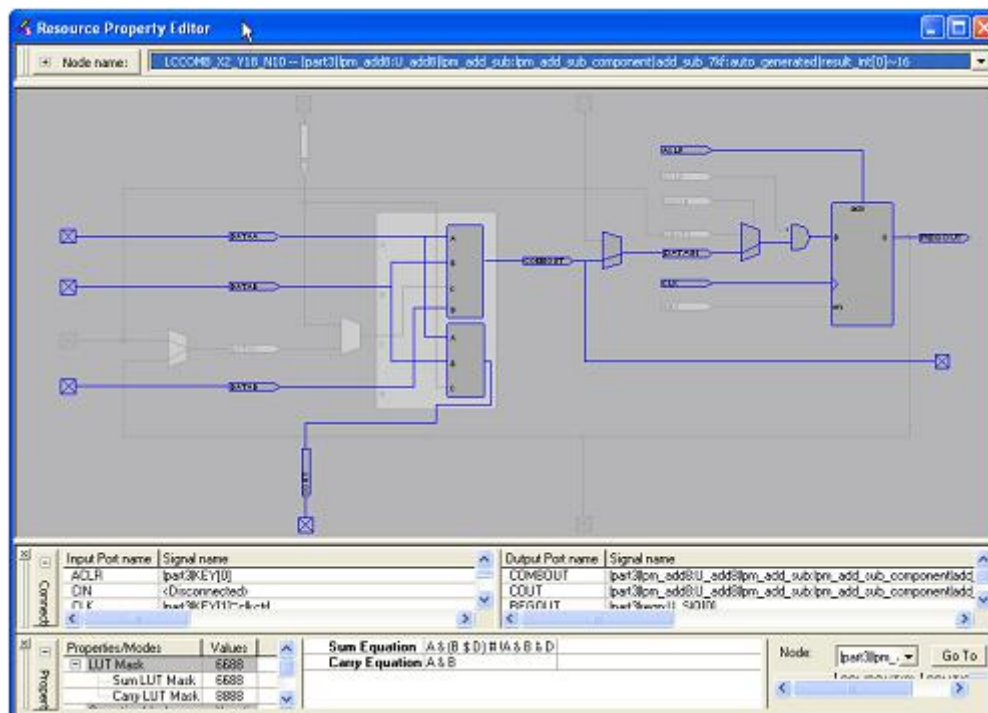
Figure 68. Examining details in a logic element using the Resource Property Editor.

2. Open the Quartus II Compilation Report and and compare the fmax of your adder circuit with the one designed in Part I. Discuss any differences in performance that are observed.

Part IV

Repeat Part II using the predefined adder circuit called lpm_add_sub, instead of your adder-subtractor circuit based on Figure 65.

Comment briefly on the circuit structure obtained using the LPM module, and compare the fmax of this circuit to the one from Part II. Describe how the lpm_add _sub module implements the Overflow signal.

Part V

Figure 69a gives an example of the traditional paper-and-pencil multiplication $P = A \times B$, where A =12 and B =11. We need to add two summands that are shifted versions of A to form the product P = 132. Part b of the figure shows the same example using four-bit binary numbers. Since each digit in B is either 1 or 0, the summands are either shifted versions of A or 0000. Figure 69c shows how each summand can be formed by using the Boolean AND operation of A with the appropriate bit in B.

$$
\begin{array}{r}
1\ 2 \\
\times\ 1\ 1 \\
\hline
1\ 2 \\
1\ 2 \\
\hline
1\ 3\ 2
\end{array}
$$

a) Decimal

$$
\begin{array}{r}
1\ 1\ 0\ 0 \\
\times\ 1\ 0\ 1\ 1 \\
\hline
1\ 1\ 0\ 0 \\
1\ 1\ 0\ 0 \\
0\ 0\ 0\ 0 \\
1\ 1\ 0\ 0 \\
\hline
1\ 0\ 0\ 0\ 0\ 1\ 0\ 0
\end{array}
$$

b) Binary

c) Implementation

Figure 69. Multiplication of binary numbers.

A four-bit circuit that implements $P = A \times B$ is illustrated in Figure 70. Because of its regular structure, this type of multiplier circuit is usually called an array multiplier. The shaded areas in the circuit correspond to the shaded columns in Figure 69c. In each row of the multiplier AND gates are used to produce the summands, and full adder modules are used to generate the required sums.
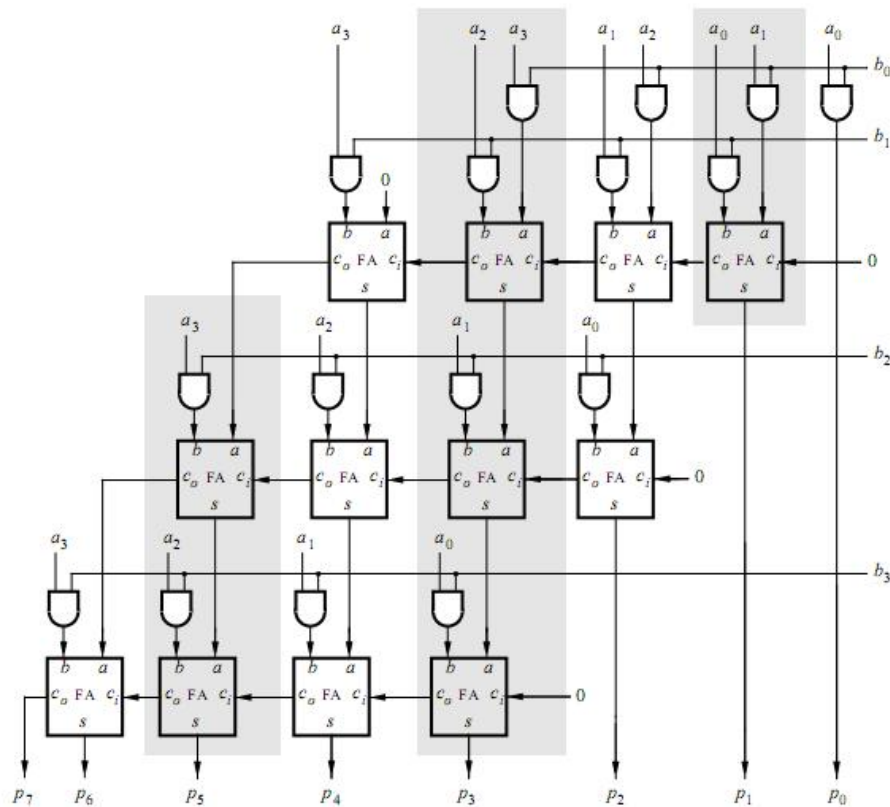
Figure 70. An array multiplier circuit.

58

Use the following steps to implement the array multiplier circuit:

1. Create a new Quartus II project which will be used to implement the desired circuit on the Altera DE2 board.

2. Generate the required VHDL file, include it in your project, and compile the circuit.

3. Use functional simulation to verify that your code is correct.

4. Augment your design to use switches SW11−8 to represent the number A and switches SW3−0 to represent B. The hexadecimal values of A and B are to be displayed on the 7-segment displays HEX6 and HEX4, respectively. The result P = A × B is to be displayed on HEX1 and HEX0.

5. Assign the pins on the FPGA to connect to the switches and 7-segment displays, as indicated in the User Manual for the DE2 board.

6. Recompile the circuit and download it into the FPGA chip.

7. Test the functionality of your design by toggling the switches and observing the 7-segment displays.

Part VI

Extend yourmultiplier to multiply 8-bit numbers and produce a 16-bit product. Use switches SW15−8 to represent the number A and switches SW7−0 to represent B. The hexadecimal values of A and B are to be displayed on the 7-segment displays HEX7−6 and HEX5−4, respectively. The result P = A × B is to be displayed on HEX3−0.Add registers to your circuit to store the values of A, B, and the product P, using a similar structure as shown for the registered adder in Figure 65.

After successfully compiling and testing your multiplier circuit, examine the results produced by the Quartus II Timing Analyzer to determine the fmax of your circuit. What is the longest path in terms of delay between registers?

Part VII

Change your VHDL code to implement the 8 x 8 multiplier by using the lpm mult module from the library of parameterized modules in the Quartus II system. Complete the design steps above. Compare the results in terms of the number of logic elements (LEs) needed and the circuit fmax.

Part VIII

It many applications of digital circuits it is useful to be able to perform some number of multiplications and then produce a summation of the results. For this part of the exercise you are to design a circuit that performs the calculation

S =(A × B)+(C × D)

The inputs A, B, C, and D are eight-bit unsigned numbers, and S provides a 16-bit result. Your circuit should also provide a carry-out signal, Cout. All of the inputs and outputs of the circuit should be registered, similar to the structure shown in Figure 65b.

1. Create a new Quartus II project which will be used to implement the desired circuit on the Altera DE2 board. Use the lpm_mult and lpm_add_sub modules to realize the multipliers and adders in your design.

2. Connect the inputs A and C to switches SW15−8 and connect the inputs B and D to switches SW7−0. Use switch SW16 to select between these two sets of inputs: A, B or C, D. Also, use the switch SW17 as a write enable (WE) input. Setting WE to 1 should allow data to be loaded into the input registers when an active clock edge occurs, while setting WE to 0 should prevent loading of these registers.

3. Use KEY0 as an active-low asynchronous reset input, and use KEY1 as a manual clock input.

4. Display the hexadecimal value of either A or C, as selected by SW16, on displays HEX7-6 and display either B or D on HEX5-4. The sum S should be shown on HEX3-0, and the Cout signal should appear on LEDG8.

5. Compile your code and use either functional or timing simulation to verify that your circuit works properly. Then download the circuit onto the DE2 board and test its operation.

6. It is often necessary to ensure that a digital circuit is able to meet certain speed requirements, such as a particular frequency of a signal applied to a clock input. Such requirements are provided to a CAD system in the form of timing constraints. The procedure for using timing constraints in the Quartus II CAD system is described in the tutorial Timing Considerations with VHDL-Based Designs, which is available on the DE2 System CD and in the University Program section of Altera's web site.

For this exercise we are using a manual clock that is applied by a pushbutton switch, so no realistic timing requirements exist. But to demonstrate the design issues involved, assume that your circuit is required to operate with a clock frequency of 220 MHz. Enter this frequency as a timing constraint in the Quartus II software, and recompile your project. The Timing Analyzer should report that it is unable to meet the timing requirements due to the lengths of various register-to-register paths in the circuit. Examine the timing analysis report and describe briefly the timing violations observed.

7. One way to increase the speed of operation of a given circuit is to insert registers into the circuit in a way that shortens the lengths of its longest paths. This technique is referred to as pipelining a circuit, and the inserted registers are often called pipeline registers. Insert pipeline registers into your design between the multipliers and the adder. Recompile your project and discuss the results obtained.

Part IX

The Quartus II software includes a predesigned module called altmult add that can perform calculations of the form S =(A × B)+(C × D). Repeat Part VIII using this module instead of the lpm mult and lpm add sub modules. Test your circuit using both simulation and by downloading the circuit onto the DE2 board.

Briefly describe how the implementation of your circuit differs when using the altmult_add module. Examine its performance both with and without the pipeline registers discussed in Part VIII.

Laboratory Exercise 5：Finite State Machines

This is an exercise in using finite state machines.

Part I

We wish to implement a finite state machine (FSM) that recognizes two specific sequences of applied input sym-bols, namely four consecutive 1s or four consecutive 0s. There is an input w and an output z. Whenever w =1 or w =0 for four consecutive clock pulses the value of z has to be 1; otherwise, z =0. Overlapping sequences are
allowed, so that if w =1 for five consecutive clock pulses the output z will be equal to 1 after the fourth and fifth pulses. Figure 71 illustrates the required relationship between w and z.
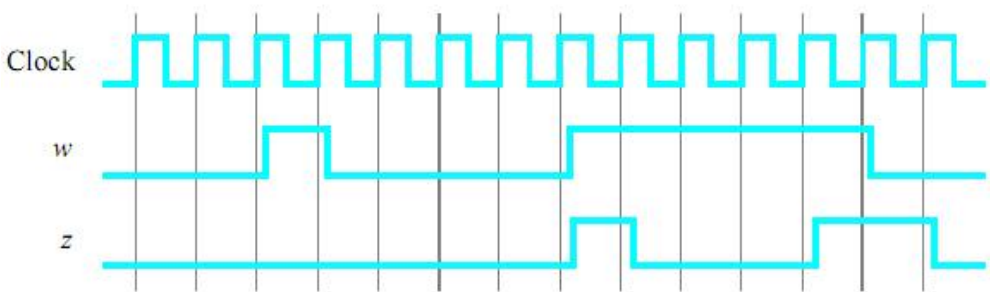
Figure 71. Required timing for the output z.

A state diagram for this FSM is shown in Figure 72. For this part you are to manually derive an FSM circuit that implements this state diagram, including the logic expressions that feed each of the state flip-flops. To implement the FSM use nine state flip-flops called y8,...,y0 and the one-hot state assignment given in Table 4.

Name	State Code $y_8y_7y_6y_5 y_4y_3 y_2y_1y_0$
A	000000001
B	000000010
C	000000100
D	000001000
E	000010000
F	000100000
G	001000000
H	010000000
I	100000000

9

Table 4. One-hot codes for the FSM.
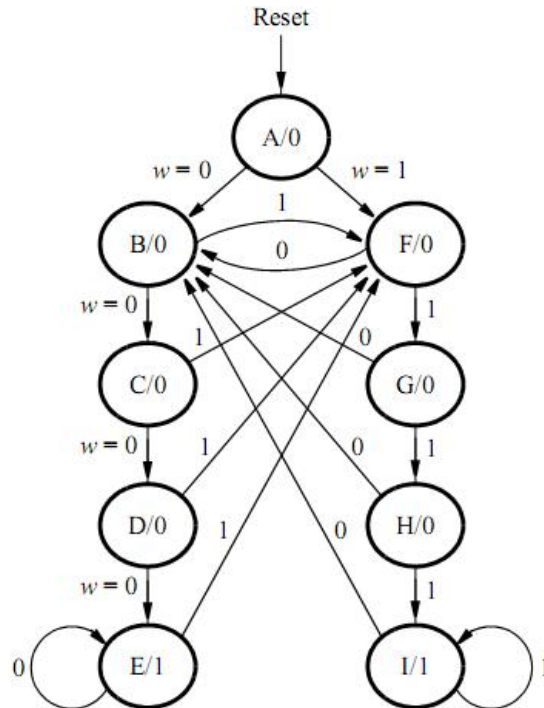
Figure 72. A state diagram for the FSM.

Design and implement your circuit on the DE2 board as follows.

1. Create a new Quartus II project for the FSMcircuit. Select as the target chip the Cyclone II EP2C35F672C6,which is the FPGA chip on the Altera DE2 board.

2. Write a VHDL file that instantiates the nine flip-flops in the circuit and which specifies the logic expressions that drive the flip-flop input ports. Use only simple assignment statements in your VHDL code to specify the logic feeding the flip-flops. Note that the one-hot code enables you to derive these expressions by inspection.

Use the toggle switch SW0 on the Altera DE2 board as an active-low synchronous reset input for the FSM, use SW1 as the w input, and the pushbutton KEY0 as the clock input which is applied manually. Use the green LED LEDG0 as the output z, and assign the state flip-flop outputs to the red LEDs LEDR8 to LEDR0.

3. Include the VHDL file in your project, and assign the pins on the FPGA to connect to the switches and the LEDs, as indicated in the User Manual for the DE2 board. Compile the circuit.

4. Simulate the behavior of your circuit.

5. Once you are confident that the circuit works properly as a result of your simulation, download the circuit into the FPGA chip. Test the functionality of your design by applying the input sequences and observing the output LEDs. Make sure that the FSM properly transitions between states as displayed on the red LEDs, and that it produces the correct output values on LEDG0.

6. Finally, consider a modification of the one-hot code given in Table 4. When an FSM is going to be im-plemented in an FPGA, the circuit can often be simplified if all flip-flop outputs are 0 when the FSM is in the reset state. This approach is preferable because the FPGA's flip-flops usually include a clear input port, which can be conveniently used to realize the reset state, but the flip-flops often do not include a set input port.

Table 5 shows a modified one-hot state assignment in which the reset state, A, uses all 0s. This is accom-plished by inverting the state variable y0. Create a modified version of your VHDL code

that implements this state assignment. Hint: you should need to make very few changes to the logic expressions in your circuit to implement the modified codes. Compile your new circuit and test it both through simulation and by downloading it onto the DE2 board.

Name	State Code $y_8y_7y_6y_5y_4y_3y_2y_1y_0$
A	000000000
B	000000011
C	000000101
D	000001001
E	000010001
F	000100001
G	001000001
H	010000001
I	100000001

Table 5. Modified one-hot codes for the FSM.

Part II

For this part you are to write another style of VHDL code for the FSM in Figure 72. In this version of the code you should not manually derive the logic expressions needed for each state flip-flop. Instead, describe the state table for the FSM by using a VHDL CASE statement in a PROCESS block, and use another PROCESS block to instantiate the state flip-flops. You can use a third PROCESS block or simple assignment statements to specify the output z.

A suggested skeleton of the VHDL code is given. Observe that the present and next state vectors for the FSMare defined as an enumerated type with possible values given by the symbols A to I. The VHDL compiler determines how many state flip-flops to use for the circuit, and it automatically chooses the state assignment.

LIBRARY ieee;
USE ieee.std_logic_1164.all;
    ENTITY part2 IS
        PORT (... define input and output ports...);
    END part2;
    ARCHITECTURE Behavior OF part2 IS
      ... declare signals
    LIBRARY ieee;
    USE ieee.std_logic_1164.all;
    ENTITY part2 IS
        PORT (... define input and output ports...);
    END part2;
    ARCHITECTURE Behavior OF part2 IS
    ... declare signals
    TYPE State_type IS (A, B, C, D, E, F, G, H, I);
     SIGNAL y_Q, Y_D : State_type; - - y_Q is present state, y_D is next state

```
        BEGIN
        ...
        PROCESS (w, y_Q) - - state table
        BEGIN
            case y_Q IS
            WHEN A IF (w = '0') THEN Y_D <= B;
            ELSE Y_D <= F;
            END IF;
            ... other states
            END CASE;
        END PROCESS; - - state table
        PROCESS (Clock) - - state flip-flops
            BEGIN
        ...
        END PROCESS;
        ... assignments for output z and the LEDs
END Behavior;
```

Implement your circuit as follows.

1. Create a new project for the FSM. Select as the target chip the Cyclone II EP2C70F896C6.

2. Include in the project your VHDL file that uses the style of code as given. Use the toggle switch SW0 on the Altera DE2 board as an active-low synchronous reset input for the FSM, use SW1 as the w input, and the pushbutton KEY0 as the clock input which is applied manually. Use the green LED LEDG0 as the output z,and use nine red LEDs, LEDR8 to LEDR0, to indicate the present state of the FSM. Assign the pins on the FPGA to connect to the switches and the LEDs, as indicated in the User Manual for the DE2 board.

3. Before compiling your code it is possible to tell the Synthesis tool in Quartus II what style of state as-signment it should use. Choose Assignments > Settings in Quartus II, and then click on the Analysis and Synthesis item on the left side of the window. As indicated in Figure 73, change the parameter State Machine Processing to the setting Minimal Bits.

4. To examine the circuit produced by Quartus II open the RTL Viewer tool. Double-click on the box shown in the circuit that represents the finite state machine, and determine whether the state diagram that it shows properly corresponds to the one in Figure 72. To see the state codes used for your FSM, open the Compilation Report, select the Analysis and Synthesis section of the report, and click on State Machines.

5. Simulate the behavior of your circuit.

6. Once you are confident that the circuit works properly as a result of your simulation, download the circuit into the FPGA chip. Test the functionality of your design by applying the input sequences and observing the output LEDs. Make sure that the FSM properly transitions between states as displayed on the red LEDs, and that it produces the correct output values on LEDG0.

7. In step 3 you instructed the Quartus II Synthesis tool to use the state assignment given in your VHDL code. To see the result of changing this setting, open again the Quartus II settings window by choosing Assignments > Settings, and click on the Analysis and Synthesis item. Change the setting for State Machine Processing from Minimal Bits to One-Hot. Recompile the circuit and then open the report file, select the Analysis and Synthesis section of the report, and click on State

Machines. Compare the state codes shown to those given in Table 5, and discuss any differences that you observe.
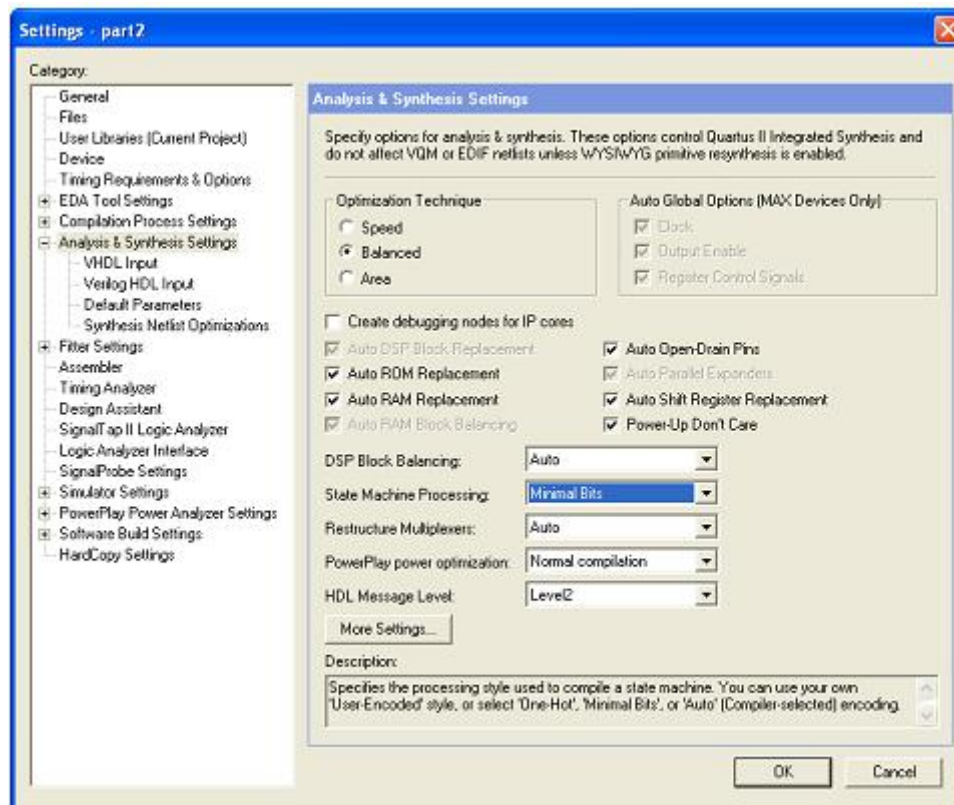
Figure 73. Specifying the state assignment method in Quartus II.

Laboratory Exercise 6：Counters

This is an exercise in using counters.

Part I

Consider the circuit in Figure 1. It is a 4-bit synchronous counter which uses four T-type flip-flops. The counter increments its count on each positive edge of the clock if the Enable signal is asserted. The counter is reset to 0 by using the Reset signal. You are to implement a 16-bit counter of this type.
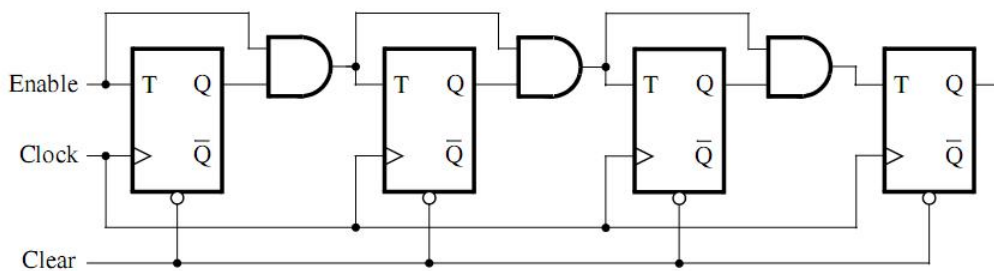
Figure 1. A 4-bit counter.

1. Write a VHDL file that defines a 16-bit counter by using the structure depicted in Figure 1, and compile the circuit.
2. Simulate your circuit to verify its correctness.
3. Augment your VHDL file to use the pushbutton KEY0 as the Clock input, switches SW1 and SW0 as Enable and Reset inputs, and 7-segment displays HEX3-0 to display the hexadecimal count as your circuit operates. Make the necessary pin assignments and compile the circuit.
4. Implement your circuit on the DE2 board and test its functionality by operating the implemented switches.
5. Implement a 4-bit version of your circuit and use the Quartus II RTL Viewer to see how Quartus II software synthesized your circuit.

Part II

Design and implement a circuit that displays the word HELLO, in ticker tape fashion, on the eight 7-segment displays HEX7 − 0. Make the letters move from right to left in intervals of about one second. The patterns that should be displayed in successive clock intervals are given in Table 1.

Clock cycle	Displayed pattern								
0				H	E	L	L	O	
1			H	E	L	L	O		
2		H	E	L	L	O			
3	H	E	L	L	O				
4	E	L	L	O					H
5	L	L	O					H	E
6	L	O					H	E	L
7	O					H	E	L	L
8				H	E	L	L	O	
. . .	and so on								

Table 1. Scrolling the word HELLO in ticker-tape fashion.

Laboratory Exercise 7: A Dice Game

This exercise is to design an electronic dice game. This game is popularly known as craps in the United States.The game involves two dice, each of which can have a value between 1 and 6. Two counters are used to simulate the roll of the dice. Each counter counts in the sequence 1, 2, 3, 4, 5, 6, 1, 2, ... Thus, after the "roll" of the dice, the sum of the values in the two counters will be in the range 2 through 12. The rules of the game are as follows.

1. After the first roll of the dice, the player wins if the sum is 7 or 11. The player loses if the sum is 2, 3, or,12. Otherwise, the sum the player obtained on the first roll is referred to as a point, and he or she must roll the dice again.

2. On the second or subsequent roll of the dice, the player wins if the sum equals the point, and he or she loses if the sum is 7. Otherwise, the player must roll again until he or she finally wins or loses.Figure 1 shows the block diagram for the dice game. The inputs to the dice game come from two push buttons, Rb (roll button) and Reset. Reset is used to initiate a new game. When the roll button is pushed, the dice counters count at a high speed, so the values cannot be read on the display. When the roll button is released, the values in the two counters are displayed via 7-segment displays.
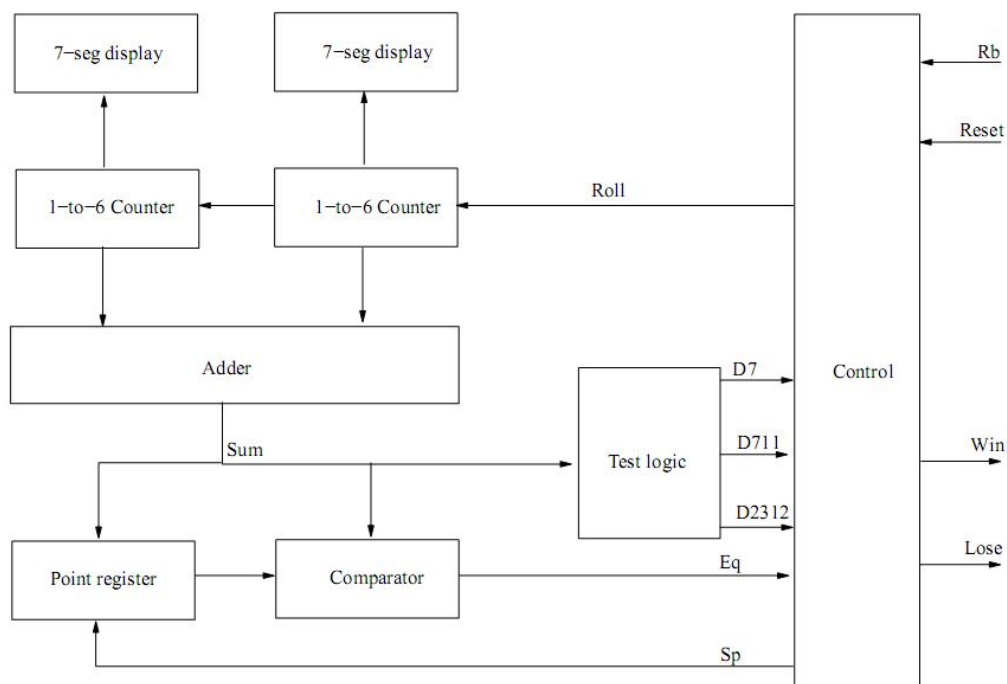
Figure 1. Block diagram for dice game.

The components for the dice game shown in the block diagram (Figure 1) include an adder, which adds the two counter outputs, a register to store the point, test logic to determine conditions for win or lose, and a control circuit.Input signals to the control circuit are defined as follows:

D7 = 1 if the sum of the dice is 7

D711 =1 if the sum of the dice is 7 or 11

D2312 =1 if the sum of the dice is 2, 3, or 12

Eq =1 if the sum of the dice equals the number stored in the point register

Rb =1 when the roll button is pressed

Reset =1 when the reset button is pressed

Outputs from the control circuit are defined as follows:

Roll =1 enables the dice counters

Sp =1 causes the sum to be stored in the point register

Win =1 turns on the win light

Lose =1 turns on the lose light

Design and implement the dice game shown in Figure 1 using VHDL code as follows:

1. Create a new Quartus II project for this exercise.

2. Generate the required VHDL file, include it in your project.

3. Compile the circuit and use functional simulation to verify the correct operation of your designed dice game.

4. Use the displays HEX1 and HEX0 to show the values of two counter outputs. Also, use push button KEY0 for Reset and KEY1 for the roll button. If the player wins, light on a green LED light on the board, and if the player loses, light on a red LED light on the board.

5. Add to your project the necessary pin assignments for the DE2 board. Compile the circuit and download it into the FPGA chip.

6. Test the functionality of your design by playing the dice game.

Laboratory Exercise 8： A Simple Processor

Figure 1 shows a digital system that contains a number of 16-bit registers, a multiplexer, an adder/subtracter unit, a counter, and a control unit. Data is input to this system via the 16-bit DIN input. This data can be loaded through the 16-bit wide multiplexer into the various registers, such as R0, . . . ,R7 and A. The multiplexer also allows data to be transferred from one register to another. The multiplexer's output wires are called a bus in the figure because this term is often used for wiring that allows data to be transferred from one location in a system to another.

Addition or subtraction is performed by using the multiplexer to first place one 16-bit number onto the bus wires and loading this number into register A. Once this is done, a second 16-bit number is placed onto the bus, the adder/subtracter unit performs the required operation, and the result is loaded into register G. The data in G can then be transferred to one of the other registers as required.
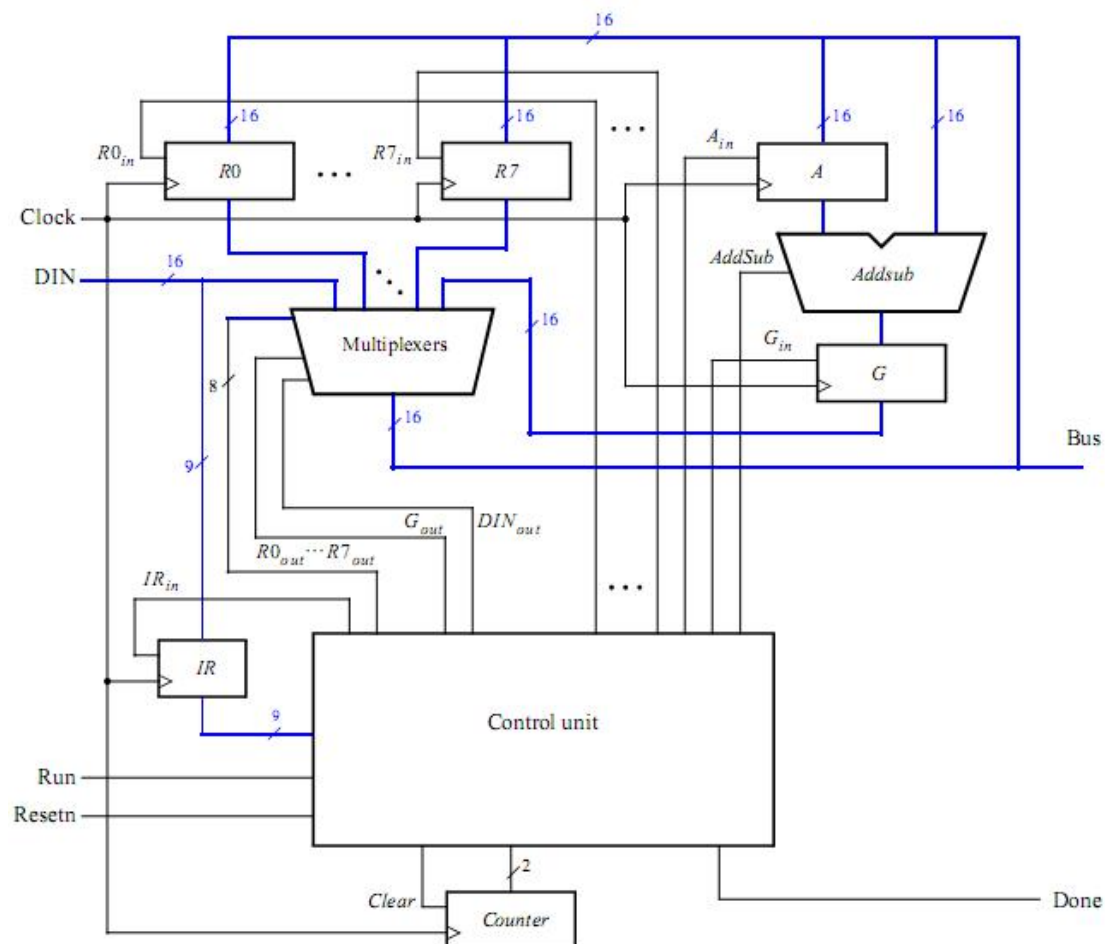
Figure 1. A digital system.

The system can perform different operations in each clock cycle, as governed by the control unit. This unit determines when particular data is placed onto the bus wires and it controls which of the registers is to be loaded with this data. For example, if the control unit asserts the signals R0out and Ain, then the multiplexer will place the contents of register R0 onto the bus and this data will be loaded by the next active clock edge into register A.

A system like this is often called a processor. It executes operations specified in the form of instructions.Table 1 lists the instructions that the processor has to support for this exercise. The left column shows the name of an instruction and its operand. The meaning of the syntax RX ← [RY] is that the contents of register RY are loaded into register RX. The mv (move) instruction allows data to be copied from one register to another. For the mvi (move immediate) instruction the expression RX ← D indicates that the 16-bit constant D is loaded into register RX.

Operation	Function performed
mv Rx, Ry	$Rx \leftarrow [Ry]$
mvi $Rx, \#D$	$Rx \leftarrow D$
add Rx, Ry	$Rx \leftarrow [Rx] + [Ry]$
sub Rx, Ry	$Rx \leftarrow [Rx] - [Ry]$

Table 1. Instructions performed in the processor.

Each instruction can be encoded and stored in the IR register using the 9-bit format IIIXXXYYY, where III represents the instruction, XXX gives the RX register, and YYY gives the RY register. Hence IR has to be connected to nine bits of the 16-bit DIN input, as indicated in Figure 1. For the mvi instruction the YYY field has no meaning, and the immediate data #D has to be supplied on the 16-bit DIN input after the mvi instruction word is stored into IR.

Some instructions, such as an addition or subtraction, take more than one clock cycle to complete, because multiple transfers have to be performed across the bus. The control unit uses the two-bit counter shown in Figure 1 to enable it to "step through" such instructions. The processor starts executing the instruction on the DIN input when the Run signal is asserted and the processor asserts the Done output when the instruction is finished. Table 2 indicates the control signals that can be asserted in each time step to implement the instructions in Table 1. Note that the only control signal asserted in time step 0 is IRin, so this time step is not shown in the table.

	T_1	T_2	T_3
(mv): I_0	$RY_{out}, RX_{in},$ Done		
(mvi): I_1	$DIN_{out}, RX_{in},$ Done		
(add): I_2	RX_{out}, A_{in}	RY_{out}, G_{in}	$G_{out}, RX_{in},$ Done
(sub): I_3	RX_{out}, A_{in}	$RY_{out}, G_{in},$ AddSub	$G_{out}, RX_{in},$ Done

Table 2. Control signals asserted in each instruction/time step.

Design and implement the processor shown in Figure 1 using VHDL code as follows:

1. Create a new Quartus II project for this exercise.

2. Generate the required VHDL file, include it in your project, and compile the circuit. A suggested skeleton of the VHDL code is shown in parts a and b of Figure 2, and some subcircuit entities that can be used in this code appear in parts c and d.

3. Use functional simulation to verify that your code is correct. An example of the output produced by a functional simulation for a correctly-designed circuit is given in Figure 3. It shows the value (2000)16 being loaded into IR from DIN at time 30 ns. This pattern represents the instruction mvi

R0,#D, where the value D = 5 is loaded into R0 on the clock edge at 50 ns. The simulation then shows the instruction mv R1,R0 at 90 ns, add R0,R1 at 110 ns, and sub R0,R0 at 190 ns. Note that the simulation output shows DIN as a 4-digit hexadecimal number, and it shows the contents of IR as a 3-digit octal number.

4. Create a new Quartus II project which will be used for implementation of the circuit on the Altera DE2 board. This project should consist of a top-level entity that contains the appropriate input and output ports for the Altera board. Instantiate your processor in this top-level entity. Use switches SW15−0 to drive the DIN input port of the processor and use switch SW17 to drive the Run input. Also, use push button KEY0 for Resetn and KEY1 for Clock. Connect the processor bus wires to LEDR15−0 and connect the Done signal to LEDR17.

5. Add to your project the necessary pin assignments for the DE2 board. Compile the circuit and download it into the FPGA chip.

6. Test the functionality of your design by toggling the switches and observing the LEDs. Since the processor's clock input is controlled by a push button switch, it is easy to step through the execution of instructions and observe the behavior of the circuit.

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
ENTITY proc IS
     PORT ( DIN : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
               Resetn, Clock, Run : IN STD_LOGIC;
               Done : BUFFER STD_LOGIC;
               BusWires : BUFFER STD_LOGIC_VECTOR(15 DOWNTO 0));
     END proc;
ARCHITECTURE Behavior OF proc IS
     . . . declare components
     . . . declare signals
BEGIN
     High <= '1';
     Clear <= . . .
     Tstep: upcount PORT MAP (Clear, Clock, Tstep_Q);
     I <= IR(1 TO 3);
     decX: dec3to8 PORT MAP (IR(4 TO 6), High, Xreg);
     decY: dec3to8 PORT MAP (IR(7 TO 9), High, Yreg);
```
                    Figure 2a. Skeleton VHDL code for the processor.
```
controlsignals: PROCESS (Tstep_Q, I, Xreg, Yreg)
BEGIN
     . . . specify initial values
     CASE Tstep_Q IS
          WHEN "00" => - - store DIN in IR as long as Tstep_Q = 0
               IRin <= '1';
          WHEN "01" => - - define signals in time step T1
               CASE I IS
                    . . .
               END CASE;
```

```
                    WHEN "10" => - - define signals in time step T2
                         CASE I IS
                               . . .
                         END CASE;
                    WHEN "11" => - - define signals in time step T3
                         CASE I IS
                               . . .
                         END CASE;
               END CASE;
          END PROCESS;
     reg_0: regn PORT MAP (BusWires, Rin(0), Clock, R0);
     . . . instantiate other registers and the adder/subtracter unit
     . . . define the bus
END Behavior;
```

Figure 2b. Skeleton VHDL code for the processor.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
ENTITY upcount IS
     PORT ( Clear, Clock : IN STD_LOGIC;
          Q : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END upcount;
ARCHITECTURE Behavior OF upcount IS
     SIGNAL Count : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
     PROCESS (Clock)
     BEGIN
          IF (Clock'EVENT AND Clock = '1') THEN
               IF Clear = '1' THEN
                    Count <= "00";
               ELSE
                    Count <= Count + 1;
               END IF;
          END IF;
     END PROCESS;
     Q <= Count;
END Behavior;
```

Figure 2c. Subcircuit entities for use in the processor.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dec3to8 IS
     PORT ( W : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          En : IN STD_LOGIC;
          Y : OUT STD_LOGIC_VECTOR(0 TO 7));
```

```
END dec3to8;
ARCHITECTURE Behavior OF dec3to8 IS
BEGIN
    PROCESS (W, En)
    BEGIN
        IF En = '1' THEN
            CASE W IS
                WHEN "000" => Y <= "10000000";
                WHEN "001" => Y <= "01000000";
                WHEN "010" => Y <= "00100000";
                WHEN "011" => Y <= "00010000";
                WHEN "100" => Y <= "00001000";
                WHEN "101" => Y <= "00000100";
                WHEN "110" => Y <= "00000010";
                WHEN "111" => Y <= "00000001";
            END CASE;
        ELSE
            Y <= "00000000";
        END IF;
    END PROCESS;
END Behavior;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY regn IS
    GENERIC (n : INTEGER := 16);
    PORT ( R : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
            Rin, Clock : IN STD_LOGIC;
            Q : BUFFER STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END regn;
ARCHITECTURE Behavior OF regn IS
BEGIN
    PROCESS (Clock)
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN
            IF Rin = '1' THEN
                Q <= R;
            END IF;
        END IF;
    END PROCESS;
END Behavior;
```

Figure 2d. Subcircuit entities for use in the processor.
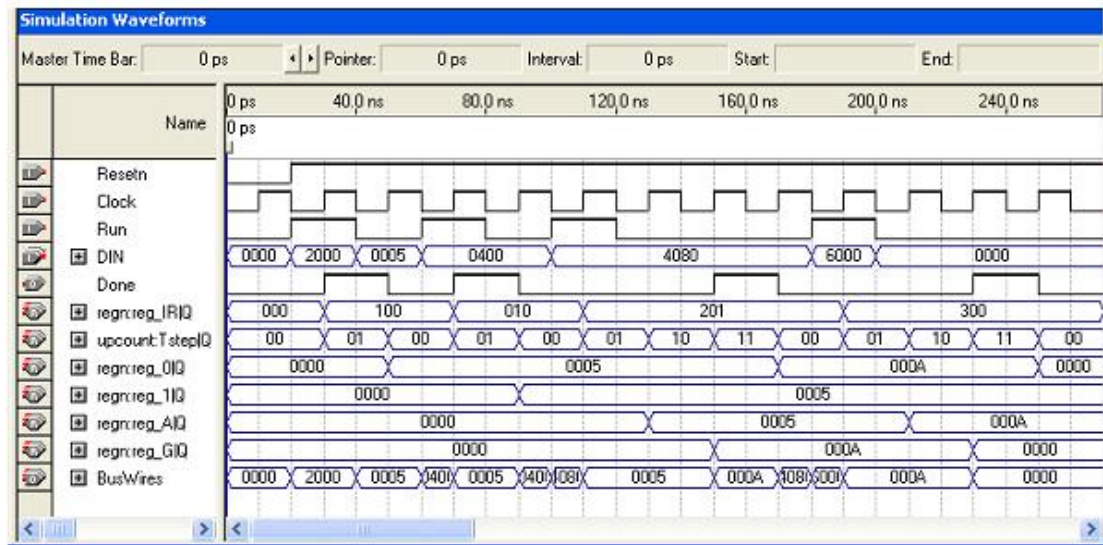
Figure 3. Simulation of the processor.

Table A

TabA.DE2_70_pin_assignment

From	To	Assignment Name	Value	Enabled
	oENET_CLK	Location	PIN_D27	Yes
	iCLK_28	Location	PIN_E16	Yes
	iCLK_50	Location	PIN_AD15	Yes
	iCLK_50_2	Location	PIN_D16	Yes
	iCLK_50_3	Location	PIN_R28	Yes
	iCLK_50_4	Location	PIN_R3	Yes
	iAUD_ADCDAT	Location	PIN_E19	Yes
	AUD_ADCLRCK	Location	PIN_F19	Yes
	AUD_BCLK	Location	PIN_E17	Yes
	oAUD_DACDAT	Location	PIN_F18	Yes
	AUD_DACLRCK	Location	PIN_G18	Yes
	oAUD_XCK	Location	PIN_D17	Yes
	DRAM_DQ[0]	Location	PIN_AC1	Yes
	DRAM_DQ[1]	Location	PIN_AC2	Yes
	DRAM_DQ[10]	Location	PIN_AF2	Yes
	DRAM_DQ[11]	Location	PIN_AF3	Yes
	DRAM_DQ[12]	Location	PIN_AG2	Yes
	DRAM_DQ[13]	Location	PIN_AG3	Yes
	DRAM_DQ[14]	Location	PIN_AH1	Yes
	DRAM_DQ[15]	Location	PIN_AH2	Yes
	DRAM_DQ[16]	Location	PIN_U1	Yes
	DRAM_DQ[17]	Location	PIN_U2	Yes
	DRAM_DQ[18]	Location	PIN_U3	Yes
	DRAM_DQ[19]	Location	PIN_V2	Yes
	DRAM_DQ[2]	Location	PIN_AC3	Yes
	DRAM_DQ[20]	Location	PIN_V3	Yes
	DRAM_DQ[21]	Location	PIN_W1	Yes
	DRAM_DQ[22]	Location	PIN_W2	Yes
	DRAM_DQ[23]	Location	PIN_W3	Yes
	DRAM_DQ[24]	Location	PIN_Y1	Yes
	DRAM_DQ[25]	Location	PIN_Y2	Yes
	DRAM_DQ[26]	Location	PIN_Y3	Yes
	DRAM_DQ[27]	Location	PIN_AA1	Yes
	DRAM_DQ[28]	Location	PIN_AA2	Yes
	DRAM_DQ[29]	Location	PIN_AA3	Yes
	DRAM_DQ[3]	Location	PIN_AD1	Yes
	DRAM_DQ[30]	Location	PIN_AB1	Yes

DRAM_DQ[31]	Location	PIN_AB2	Yes
DRAM_DQ[4]	Location	PIN_AD2	Yes
DRAM_DQ[5]	Location	PIN_AD3	Yes
DRAM_DQ[6]	Location	PIN_AE1	Yes
DRAM_DQ[7]	Location	PIN_AE2	Yes
DRAM_DQ[8]	Location	PIN_AE3	Yes
DRAM_DQ[9]	Location	PIN_AF1	Yes
oDRAM0_A[0]	Location	PIN_AA4	Yes
oDRAM0_A[1]	Location	PIN_AA5	Yes
oDRAM0_A[10]	Location	PIN_Y8	Yes
oDRAM0_A[11]	Location	PIN_AE4	Yes
oDRAM0_A[12]	Location	PIN_AF4	Yes
oDRAM0_A[2]	Location	PIN_AA6	Yes
oDRAM0_A[3]	Location	PIN_AB5	Yes
oDRAM0_A[4]	Location	PIN_AB7	Yes
oDRAM0_A[5]	Location	PIN_AC4	Yes
oDRAM0_A[6]	Location	PIN_AC5	Yes
oDRAM0_A[7]	Location	PIN_AC6	Yes
oDRAM0_A[8]	Location	PIN_AD4	Yes
oDRAM0_A[9]	Location	PIN_AC7	Yes
oDRAM0_BA[0]	Location	PIN_AA9	Yes
oDRAM0_BA[1]	Location	PIN_AA10	Yes
oDRAM0_CAS_N	Location	PIN_W10	Yes
oDRAM0_CKE	Location	PIN_AA8	Yes
oDRAM0_CLK	Location	PIN_AD6	Yes
oDRAM0_CS_N	Location	PIN_Y10	Yes
oDRAM0_LDQM0	Location	PIN_V9	Yes
oDRAM0_RAS_N	Location	PIN_Y9	Yes
oDRAM0_UDQM1	Location	PIN_AB6	Yes
oDRAM0_WE_N	Location	PIN_W9	Yes
oDRAM1_A[0]	Location	PIN_T5	Yes
oDRAM1_A[1]	Location	PIN_T6	Yes
oDRAM1_A[10]	Location	PIN_T4	Yes
oDRAM1_A[11]	Location	PIN_Y4	Yes
oDRAM1_A[12]	Location	PIN_Y7	Yes
oDRAM1_A[2]	Location	PIN_U4	Yes
oDRAM1_A[3]	Location	PIN_U6	Yes
oDRAM1_A[4]	Location	PIN_U7	Yes
oDRAM1_A[5]	Location	PIN_V7	Yes
oDRAM1_A[6]	Location	PIN_V8	Yes
oDRAM1_A[7]	Location	PIN_W4	Yes
oDRAM1_A[8]	Location	PIN_W7	Yes
oDRAM1_A[9]	Location	PIN_W8	Yes
oDRAM1_BA[0]	Location	PIN_T7	Yes

oDRAM1_BA[1]	Location	PIN_T8	Yes
oDRAM1_CAS_N	Location	PIN_N8	Yes
oDRAM1_CKE	Location	PIN_L10	Yes
oDRAM1_CLK	Location	PIN_G5	Yes
oDRAM1_CS_N	Location	PIN_P9	Yes
oDRAM1_LDQM0	Location	PIN_M10	Yes
oDRAM1_RAS_N	Location	PIN_N9	Yes
oDRAM1_UDQM1	Location	PIN_U8	Yes
oDRAM1_WE_N	Location	PIN_M9	Yes
oENET_CMD	Location	PIN_B27	Yes
oENET_CS_N	Location	PIN_C28	Yes
ENET_D[0]	Location	PIN_A23	Yes
ENET_D[1]	Location	PIN_C22	Yes
ENET_D[10]	Location	PIN_B25	Yes
ENET_D[11]	Location	PIN_A25	Yes
ENET_D[12]	Location	PIN_C24	Yes
ENET_D[13]	Location	PIN_B24	Yes
ENET_D[14]	Location	PIN_A24	Yes
ENET_D[15]	Location	PIN_B23	Yes
ENET_D[2]	Location	PIN_B22	Yes
ENET_D[3]	Location	PIN_A22	Yes
ENET_D[4]	Location	PIN_B21	Yes
ENET_D[5]	Location	PIN_A21	Yes
ENET_D[6]	Location	PIN_B20	Yes
ENET_D[7]	Location	PIN_A20	Yes
ENET_D[8]	Location	PIN_B26	Yes
ENET_D[9]	Location	PIN_A26	Yes
iENET_INT	Location	PIN_C27	Yes
oENET_IOR_N	Location	PIN_A28	Yes
oENET_IOW_N	Location	PIN_B28	Yes
oENET_RESET_N	Location	PIN_B29	Yes
iEXT_CLOCK	Location	PIN_R29	Yes
oFLASH_A[0]	Location	PIN_AF24	Yes
oFLASH_A[1]	Location	PIN_AG24	Yes
oFLASH_A[10]	Location	PIN_AH26	Yes
oFLASH_A[11]	Location	PIN_AJ26	Yes
oFLASH_A[12]	Location	PIN_AK26	Yes
oFLASH_A[13]	Location	PIN_AJ25	Yes
oFLASH_A[14]	Location	PIN_AK25	Yes
oFLASH_A[15]	Location	PIN_AH24	Yes
oFLASH_A[16]	Location	PIN_AG25	Yes
oFLASH_A[17]	Location	PIN_AF21	Yes
oFLASH_A[18]	Location	PIN_AD21	Yes
oFLASH_A[19]	Location	PIN_AK28	Yes

oFLASH_A[2]	Location	PIN_AE23	Yes
oFLASH_A[20]	Location	PIN_AJ28	Yes
oFLASH_A[21]	Location	PIN_AE20	Yes
oFLASH_A[3]	Location	PIN_AG23	Yes
oFLASH_A[4]	Location	PIN_AF23	Yes
oFLASH_A[5]	Location	PIN_AG22	Yes
oFLASH_A[6]	Location	PIN_AH22	Yes
oFLASH_A[7]	Location	PIN_AF22	Yes
oFLASH_A[8]	Location	PIN_AH27	Yes
oFLASH_A[9]	Location	PIN_AJ27	Yes
oFLASH_BYTE_N	Location	PIN_Y29	Yes
oFLASH_CE_N	Location	PIN_AG28	Yes
FLASH_DQ[0]	Location	PIN_AF29	Yes
FLASH_DQ[1]	Location	PIN_AE28	Yes
FLASH_DQ[10]	Location	PIN_AD29	Yes
FLASH_DQ[11]	Location	PIN_AC28	Yes
FLASH_DQ[12]	Location	PIN_AC30	Yes
FLASH_DQ[13]	Location	PIN_AB30	Yes
FLASH_DQ[14]	Location	PIN_AA30	Yes
FLASH_DQ15_AM1	Location	PIN_AE24	Yes
FLASH_DQ[2]	Location	PIN_AE30	Yes
FLASH_DQ[3]	Location	PIN_AD30	Yes
FLASH_DQ[4]	Location	PIN_AC29	Yes
FLASH_DQ[5]	Location	PIN_AB29	Yes
FLASH_DQ[6]	Location	PIN_AA29	Yes
FLASH_DQ[7]	Location	PIN_Y28	Yes
FLASH_DQ[8]	Location	PIN_AF30	Yes
FLASH_DQ[9]	Location	PIN_AE29	Yes
oFLASH_OE_N	Location	PIN_AG29	Yes
oFLASH_RST_N	Location	PIN_AH28	Yes
iFLASH_RY_N	Location	PIN_AH30	Yes
oFLASH_WE_N	Location	PIN_AJ29	Yes
oFLASH_WP_N	Location	PIN_AH29	Yes
GPIO_CLKIN_N0	Location	PIN_T25	Yes
GPIO_CLKIN_N1	Location	PIN_AH14	Yes
GPIO_CLKIN_P0	Location	PIN_T24	Yes
GPIO_CLKIN_P1	Location	PIN_AG15	Yes
GPIO_CLKOUT_N0	Location	PIN_H23	Yes
GPIO_CLKOUT_N1	Location	PIN_AF27	Yes
GPIO_CLKOUT_P0	Location	PIN_G24	Yes
GPIO_CLKOUT_P1	Location	PIN_AF28	Yes
GPIO_0[0]	Location	PIN_C30	Yes
GPIO_0[1]	Location	PIN_C29	Yes
GPIO_0[10]	Location	PIN_F29	Yes

GPIO_0[11]	Location	PIN_G29	Yes
GPIO_0[12]	Location	PIN_F30	Yes
GPIO_0[13]	Location	PIN_G30	Yes
GPIO_0[14]	Location	PIN_H29	Yes
GPIO_0[15]	Location	PIN_H30	Yes
GPIO_0[16]	Location	PIN_J29	Yes
GPIO_0[17]	Location	PIN_H25	Yes
GPIO_0[18]	Location	PIN_J30	Yes
GPIO_0[19]	Location	PIN_H24	Yes
GPIO_0[2]	Location	PIN_E28	Yes
GPIO_0[20]	Location	PIN_J25	Yes
GPIO_0[21]	Location	PIN_K24	Yes
GPIO_0[22]	Location	PIN_J24	Yes
GPIO_0[23]	Location	PIN_K25	Yes
GPIO_0[24]	Location	PIN_L22	Yes
GPIO_0[25]	Location	PIN_M21	Yes
GPIO_0[26]	Location	PIN_L21	Yes
GPIO_0[27]	Location	PIN_M22	Yes
GPIO_0[28]	Location	PIN_N22	Yes
GPIO_0[29]	Location	PIN_N25	Yes
GPIO_0[3]	Location	PIN_D29	Yes
GPIO_0[30]	Location	PIN_N21	Yes
GPIO_0[31]	Location	PIN_N24	Yes
GPIO_1[0]	Location	PIN_G27	Yes
GPIO_1[1]	Location	PIN_G28	Yes
GPIO_1[2]	Location	PIN_H27	Yes
GPIO_1[3]	Location	PIN_L24	Yes
GPIO_1[4]	Location	PIN_H28	Yes
GPIO_1[5]	Location	PIN_L25	Yes
GPIO_1[6]	Location	PIN_K27	Yes
GPIO_1[7]	Location	PIN_L28	Yes
GPIO_0[4]	Location	PIN_E27	Yes
GPIO_1[8]	Location	PIN_K28	Yes
GPIO_1[9]	Location	PIN_L27	Yes
GPIO_1[10]	Location	PIN_K29	Yes
GPIO_1[11]	Location	PIN_M25	Yes
GPIO_1[12]	Location	PIN_K30	Yes
GPIO_1[13]	Location	PIN_M24	Yes
GPIO_1[14]	Location	PIN_L29	Yes
GPIO_1[15]	Location	PIN_L30	Yes
GPIO_1[16]	Location	PIN_P26	Yes
GPIO_1[17]	Location	PIN_P28	Yes
GPIO_0[5]	Location	PIN_D28	Yes
GPIO_1[18]	Location	PIN_P25	Yes

GPIO_1[19]	Location	PIN_P27	Yes
GPIO_1[20]	Location	PIN_M29	Yes
GPIO_1[21]	Location	PIN_R26	Yes
GPIO_1[22]	Location	PIN_M30	Yes
GPIO_1[23]	Location	PIN_R27	Yes
GPIO_1[24]	Location	PIN_P24	Yes
GPIO_1[25]	Location	PIN_N28	Yes
GPIO_1[26]	Location	PIN_P23	Yes
GPIO_1[27]	Location	PIN_N29	Yes
GPIO_0[6]	Location	PIN_E29	Yes
GPIO_1[28]	Location	PIN_R23	Yes
GPIO_1[29]	Location	PIN_P29	Yes
GPIO_1[30]	Location	PIN_R22	Yes
GPIO_1[31]	Location	PIN_P30	Yes
GPIO_0[7]	Location	PIN_G25	Yes
GPIO_0[8]	Location	PIN_E30	Yes
GPIO_0[9]	Location	PIN_G26	Yes
oHEX0_D[0]	Location	PIN_AE8	Yes
oHEX0_D[1]	Location	PIN_AF9	Yes
oHEX0_D[2]	Location	PIN_AH9	Yes
oHEX0_D[3]	Location	PIN_AD10	Yes
oHEX0_D[4]	Location	PIN_AF10	Yes
oHEX0_D[5]	Location	PIN_AD11	Yes
oHEX0_D[6]	Location	PIN_AD12	Yes
oHEX0_DP	Location	PIN_AF12	Yes
oHEX1_D[0]	Location	PIN_AG13	Yes
oHEX1_D[1]	Location	PIN_AE16	Yes
oHEX1_D[2]	Location	PIN_AF16	Yes
oHEX1_D[3]	Location	PIN_AG16	Yes
oHEX1_D[4]	Location	PIN_AE17	Yes
oHEX1_D[5]	Location	PIN_AF17	Yes
oHEX1_D[6]	Location	PIN_AD17	Yes
oHEX1_DP	Location	PIN_AC17	Yes
oHEX2_D[0]	Location	PIN_AE7	Yes
oHEX2_D[1]	Location	PIN_AF7	Yes
oHEX2_D[2]	Location	PIN_AH5	Yes
oHEX2_D[3]	Location	PIN_AG4	Yes
oHEX2_D[4]	Location	PIN_AB18	Yes
oHEX2_D[5]	Location	PIN_AB19	Yes
oHEX2_D[6]	Location	PIN_AE19	Yes
oHEX2_DP	Location	PIN_AC19	Yes
oHEX3_D[0]	Location	PIN_P6	Yes
oHEX3_D[1]	Location	PIN_P4	Yes
oHEX3_D[2]	Location	PIN_N10	Yes

oHEX3_D[3]	Location	PIN_N7	Yes
oHEX3_D[4]	Location	PIN_M8	Yes
oHEX3_D[5]	Location	PIN_M7	Yes
oHEX3_D[6]	Location	PIN_M6	Yes
oHEX3_DP	Location	PIN_M4	Yes
oHEX4_D[0]	Location	PIN_P1	Yes
oHEX4_D[1]	Location	PIN_P2	Yes
oHEX4_D[2]	Location	PIN_P3	Yes
oHEX4_D[3]	Location	PIN_N2	Yes
oHEX4_D[4]	Location	PIN_N3	Yes
oHEX4_D[5]	Location	PIN_M1	Yes
oHEX4_D[6]	Location	PIN_M2	Yes
oHEX4_DP	Location	PIN_L6	Yes
oHEX5_D[0]	Location	PIN_M3	Yes
oHEX5_D[1]	Location	PIN_L1	Yes
oHEX5_D[2]	Location	PIN_L2	Yes
oHEX5_D[3]	Location	PIN_L3	Yes
oHEX5_D[4]	Location	PIN_K1	Yes
oHEX5_D[5]	Location	PIN_K4	Yes
oHEX5_D[6]	Location	PIN_K5	Yes
oHEX5_DP	Location	PIN_K6	Yes
oHEX6_D[0]	Location	PIN_H6	Yes
oHEX6_D[1]	Location	PIN_H4	Yes
oHEX6_D[2]	Location	PIN_H7	Yes
oHEX6_D[3]	Location	PIN_H8	Yes
oHEX6_D[4]	Location	PIN_G4	Yes
oHEX6_D[5]	Location	PIN_F4	Yes
oHEX6_D[6]	Location	PIN_E4	Yes
oHEX6_DP	Location	PIN_K2	Yes
oHEX7_D[0]	Location	PIN_K3	Yes
oHEX7_D[1]	Location	PIN_J1	Yes
oHEX7_D[2]	Location	PIN_J2	Yes
oHEX7_D[3]	Location	PIN_H1	Yes
oHEX7_D[4]	Location	PIN_H2	Yes
oHEX7_D[5]	Location	PIN_H3	Yes
oHEX7_D[6]	Location	PIN_G1	Yes
oHEX7_DP	Location	PIN_G2	Yes
oI2C_SCLK	Location	PIN_J18	Yes
I2C_SDAT	Location	PIN_H18	Yes
iIRDA_RXD	Location	PIN_W22	Yes
oIRDA_TXD	Location	PIN_W21	Yes
iKEY[0]	Location	PIN_T29	Yes
iKEY[1]	Location	PIN_T28	Yes
iKEY[2]	Location	PIN_U30	Yes

iKEY[3]	Location	PIN_U29	Yes
oLCD_BLON	Location	PIN_G3	Yes
LCD_D[0]	Location	PIN_E1	Yes
LCD_D[1]	Location	PIN_E3	Yes
LCD_D[2]	Location	PIN_D2	Yes
LCD_D[3]	Location	PIN_D3	Yes
LCD_D[4]	Location	PIN_C1	Yes
LCD_D[5]	Location	PIN_C2	Yes
LCD_D[6]	Location	PIN_C3	Yes
LCD_D[7]	Location	PIN_B2	Yes
oLCD_EN	Location	PIN_E2	Yes
oLCD_ON	Location	PIN_F1	Yes
oLCD_RS	Location	PIN_F2	Yes
oLCD_RW	Location	PIN_F3	Yes
oLEDR[0]	Location	PIN_AJ6	Yes
oLEDR[1]	Location	PIN_AK5	Yes
oLEDR[10]	Location	PIN_AC13	Yes
oLEDR[11]	Location	PIN_AB13	Yes
oLEDR[12]	Location	PIN_AC12	Yes
oLEDR[13]	Location	PIN_AB12	Yes
oLEDR[14]	Location	PIN_AC11	Yes
oLEDR[15]	Location	PIN_AD9	Yes
oLEDR[16]	Location	PIN_AD8	Yes
oLEDR[17]	Location	PIN_AJ7	Yes
oLEDG[8]	Location	PIN_AC14	Yes
oLEDG[0]	Location	PIN_W27	Yes
oLEDR[2]	Location	PIN_AJ5	Yes
oLEDG[1]	Location	PIN_W25	Yes
oLEDG[2]	Location	PIN_W23	Yes
oLEDG[3]	Location	PIN_Y27	Yes
oLEDG[4]	Location	PIN_Y24	Yes
oLEDG[5]	Location	PIN_Y23	Yes
oLEDG[6]	Location	PIN_AA27	Yes
oLEDG[7]	Location	PIN_AA24	Yes
oLEDR[3]	Location	PIN_AJ4	Yes
oLEDR[4]	Location	PIN_AK3	Yes
oLEDR[5]	Location	PIN_AH4	Yes
oLEDR[6]	Location	PIN_AJ3	Yes
oLEDR[7]	Location	PIN_AJ2	Yes
oLEDR[8]	Location	PIN_AH3	Yes
oLEDR[9]	Location	PIN_AD14	Yes
oOTG_A[0]	Location	PIN_E9	Yes
oOTG_A[1]	Location	PIN_D8	Yes
oOTG_CS_N	Location	PIN_E10	Yes

OTG_D[0]	Location	PIN_H10	Yes
OTG_D[1]	Location	PIN_G9	Yes
OTG_D[10]	Location	PIN_D6	Yes
OTG_D[11]	Location	PIN_E7	Yes
OTG_D[12]	Location	PIN_D7	Yes
OTG_D[13]	Location	PIN_E8	Yes
OTG_D[14]	Location	PIN_D9	Yes
OTG_D[15]	Location	PIN_G10	Yes
OTG_D[2]	Location	PIN_G11	Yes
OTG_D[3]	Location	PIN_F11	Yes
OTG_D[4]	Location	PIN_J12	Yes
OTG_D[5]	Location	PIN_H12	Yes
OTG_D[6]	Location	PIN_H13	Yes
OTG_D[7]	Location	PIN_G13	Yes
OTG_D[8]	Location	PIN_D4	Yes
OTG_D[9]	Location	PIN_D5	Yes
oOTG_DACK0_N	Location	PIN_D12	Yes
oOTG_DACK1_N	Location	PIN_E12	Yes
iOTG_DREQ0	Location	PIN_G12	Yes
iOTG_DREQ1	Location	PIN_F12	Yes
OTG_FSPEED	Location	PIN_F7	Yes
iOTG_INT0	Location	PIN_F13	Yes
iOTG_INT1	Location	PIN_J13	Yes
OTG_LSPEED	Location	PIN_F8	Yes
oOTG_OE_N	Location	PIN_D10	Yes
oOTG_RESET_N	Location	PIN_H14	Yes
oOTG_WE_N	Location	PIN_E11	Yes
PS2_KBCLK	Location	PIN_F24	Yes
PS2_KBDAT	Location	PIN_E24	Yes
PS2_MSCLK	Location	PIN_D26	Yes
PS2_MSDAT	Location	PIN_D25	Yes
oSD_CLK	Location	PIN_T26	Yes
SD_CMD	Location	PIN_W28	Yes
SD_DAT	Location	PIN_W29	Yes
SD_DAT3	Location	PIN_Y30	Yes
oSRAM_A[0]	Location	PIN_AG8	Yes
oSRAM_A[1]	Location	PIN_AF8	Yes
oSRAM_A[10]	Location	PIN_AF14	Yes
oSRAM_A[11]	Location	PIN_AG14	Yes
oSRAM_A[12]	Location	PIN_AE15	Yes
oSRAM_A[13]	Location	PIN_AF15	Yes
oSRAM_A[14]	Location	PIN_AC16	Yes
oSRAM_A[15]	Location	PIN_AF20	Yes
oSRAM_A[16]	Location	PIN_AG20	Yes

oSRAM_A[17]	Location	PIN_AE11	Yes
oSRAM_A[18]	Location	PIN_AF11	Yes
oSRAM_A[2]	Location	PIN_AH7	Yes
oSRAM_A[3]	Location	PIN_AG7	Yes
oSRAM_A[4]	Location	PIN_AG6	Yes
oSRAM_A[5]	Location	PIN_AG5	Yes
oSRAM_A[6]	Location	PIN_AE12	Yes
oSRAM_A[7]	Location	PIN_AG12	Yes
oSRAM_A[8]	Location	PIN_AD13	Yes
oSRAM_A[9]	Location	PIN_AE13	Yes
oSRAM_ADSC_N	Location	PIN_AG17	Yes
oSRAM_ADSP_N	Location	PIN_AC18	Yes
oSRAM_ADV_N	Location	PIN_AD16	Yes
oSRAM_BE_N[0]	Location	PIN_AC21	Yes
oSRAM_BE_N[1]	Location	PIN_AC20	Yes
oSRAM_BE_N[2]	Location	PIN_AD20	Yes
oSRAM_BE_N[3]	Location	PIN_AH20	Yes
oSRAM_CE1_N	Location	PIN_AH19	Yes
oSRAM_CE2	Location	PIN_AG19	Yes
oSRAM_CE3_N	Location	PIN_AD22	Yes
oSRAM_CLK	Location	PIN_AD7	Yes
SRAM_DPA[0]	Location	PIN_AK9	Yes
SRAM_DPA[1]	Location	PIN_AJ23	Yes
SRAM_DPA[2]	Location	PIN_AK20	Yes
SRAM_DPA[3]	Location	PIN_AJ9	Yes
SRAM_DQ[0]	Location	PIN_AH10	Yes
SRAM_DQ[1]	Location	PIN_AJ10	Yes
SRAM_DQ[10]	Location	PIN_AH17	Yes
SRAM_DQ[11]	Location	PIN_AJ18	Yes
SRAM_DQ[12]	Location	PIN_AH18	Yes
SRAM_DQ[13]	Location	PIN_AK19	Yes
SRAM_DQ[14]	Location	PIN_AJ19	Yes
SRAM_DQ[15]	Location	PIN_AK23	Yes
SRAM_DQ[16]	Location	PIN_AJ20	Yes
SRAM_DQ[17]	Location	PIN_AK21	Yes
SRAM_DQ[18]	Location	PIN_AJ21	Yes
SRAM_DQ[19]	Location	PIN_AK22	Yes
SRAM_DQ[2]	Location	PIN_AK10	Yes
SRAM_DQ[20]	Location	PIN_AJ22	Yes
SRAM_DQ[21]	Location	PIN_AH15	Yes
SRAM_DQ[22]	Location	PIN_AJ15	Yes
SRAM_DQ[23]	Location	PIN_AJ16	Yes
SRAM_DQ[24]	Location	PIN_AK14	Yes
SRAM_DQ[25]	Location	PIN_AJ14	Yes

SRAM_DQ[26]	Location	PIN_AJ13	Yes
SRAM_DQ[27]	Location	PIN_AH13	Yes
SRAM_DQ[28]	Location	PIN_AK12	Yes
SRAM_DQ[29]	Location	PIN_AK7	Yes
SRAM_DQ[3]	Location	PIN_AJ11	Yes
SRAM_DQ[30]	Location	PIN_AJ8	Yes
SRAM_DQ[31]	Location	PIN_AK8	Yes
SRAM_DQ[4]	Location	PIN_AK11	Yes
SRAM_DQ[5]	Location	PIN_AH12	Yes
SRAM_DQ[6]	Location	PIN_AJ12	Yes
SRAM_DQ[7]	Location	PIN_AH16	Yes
SRAM_DQ[8]	Location	PIN_AK17	Yes
SRAM_DQ[9]	Location	PIN_AJ17	Yes
oSRAM_GW_N	Location	PIN_AG18	Yes
oSRAM_OE_N	Location	PIN_AD18	Yes
oSRAM_WE_N	Location	PIN_AF18	Yes
iSW[0]	Location	PIN_AA23	Yes
iSW[1]	Location	PIN_AB26	Yes
iSW[10]	Location	PIN_W5	Yes
iSW[11]	Location	PIN_V10	Yes
iSW[12]	Location	PIN_U9	Yes
iSW[13]	Location	PIN_T9	Yes
iSW[14]	Location	PIN_L5	Yes
iSW[15]	Location	PIN_L4	Yes
iSW[16]	Location	PIN_L7	Yes
iSW[17]	Location	PIN_L8	Yes
iSW[2]	Location	PIN_AB25	Yes
iSW[3]	Location	PIN_AC27	Yes
iSW[4]	Location	PIN_AC26	Yes
iSW[5]	Location	PIN_AC24	Yes
iSW[6]	Location	PIN_AC23	Yes
iSW[7]	Location	PIN_AD25	Yes
iSW[8]	Location	PIN_AD24	Yes
iSW[9]	Location	PIN_AE27	Yes
iTD1_CLK27	Location	PIN_G15	Yes
iTD1_D[0]	Location	PIN_A6	Yes
iTD1_D[1]	Location	PIN_B6	Yes
iTD1_D[2]	Location	PIN_A5	Yes
iTD1_D[3]	Location	PIN_B5	Yes
iTD1_D[4]	Location	PIN_B4	Yes
iTD1_D[5]	Location	PIN_C4	Yes
iTD1_D[6]	Location	PIN_A3	Yes
iTD1_D[7]	Location	PIN_B3	Yes
iTD1_HS	Location	PIN_E13	Yes

oTD1_RESET_N	Location	PIN_D14	Yes
iTD1_VS	Location	PIN_E14	Yes
iTD2_CLK27	Location	PIN_H15	Yes
iTD2_D[0]	Location	PIN_C10	Yes
iTD2_D[1]	Location	PIN_A9	Yes
iTD2_D[2]	Location	PIN_B9	Yes
iTD2_D[3]	Location	PIN_C9	Yes
iTD2_D[4]	Location	PIN_A8	Yes
iTD2_D[5]	Location	PIN_B8	Yes
iTD2_D[6]	Location	PIN_A7	Yes
iTD2_D[7]	Location	PIN_B7	Yes
iTD2_HS	Location	PIN_E15	Yes
oTD2_RESET_N	Location	PIN_B10	Yes
iTD2_VS	Location	PIN_D15	Yes
oUART_CTS	Location	PIN_G22	Yes
iUART_RTS	Location	PIN_F23	Yes
iUART_RXD	Location	PIN_D21	Yes
oUART_TXD	Location	PIN_E21	Yes
oVGA_B[0]	Location	PIN_B16	Yes
oVGA_B[1]	Location	PIN_C16	Yes
oVGA_B[2]	Location	PIN_A17	Yes
oVGA_B[3]	Location	PIN_B17	Yes
oVGA_B[4]	Location	PIN_C18	Yes
oVGA_B[5]	Location	PIN_B18	Yes
oVGA_B[6]	Location	PIN_B19	Yes
oVGA_B[7]	Location	PIN_A19	Yes
oVGA_B[8]	Location	PIN_C19	Yes
oVGA_B[9]	Location	PIN_D19	Yes
oVGA_BLANK_N	Location	PIN_C15	Yes
oVGA_CLOCK	Location	PIN_D24	Yes
oVGA_G[0]	Location	PIN_A10	Yes
oVGA_G[1]	Location	PIN_B11	Yes
oVGA_G[2]	Location	PIN_A11	Yes
oVGA_G[3]	Location	PIN_C12	Yes
oVGA_G[4]	Location	PIN_B12	Yes
oVGA_G[5]	Location	PIN_A12	Yes
oVGA_G[6]	Location	PIN_C13	Yes
oVGA_G[7]	Location	PIN_B13	Yes
oVGA_G[8]	Location	PIN_B14	Yes
oVGA_G[9]	Location	PIN_A14	Yes
oVGA_HS	Location	PIN_J19	Yes
oVGA_R[0]	Location	PIN_D23	Yes
oVGA_R[1]	Location	PIN_E23	Yes
oVGA_R[2]	Location	PIN_E22	Yes

oVGA_R[3]	Location	PIN_D22	Yes
oVGA_R[4]	Location	PIN_H21	Yes
oVGA_R[5]	Location	PIN_G21	Yes
oVGA_R[6]	Location	PIN_H20	Yes
oVGA_R[7]	Location	PIN_F20	Yes
oVGA_R[8]	Location	PIN_E20	Yes
oVGA_R[9]	Location	PIN_G20	Yes
oVGA_SYNC_N	Location	PIN_B15	Yes
oVGA_VS	Location	PIN_H19	Yes
|	Partition Hierarchy	no_file_for_top_partition	Yes