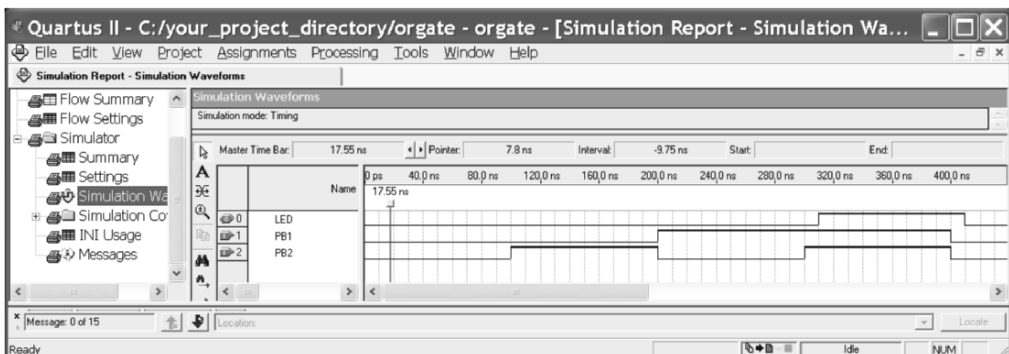
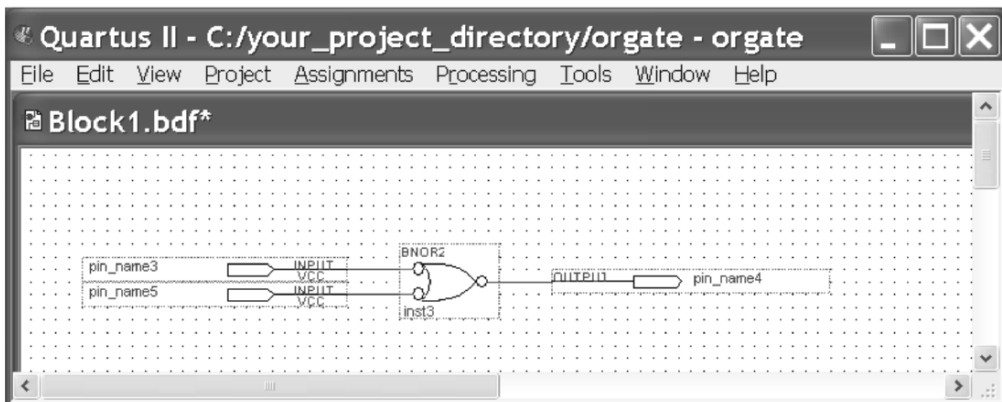


# CHAPTER 1

## *Tutorial I: The 15-Minute Design*



## 1 Tutorial I: The 15 Minute Design

---

*The purpose of this tutorial is to introduce the user to the Altera CAD tools and the University Program (DE1, DE2, UP3, UP2, or UP1) FPGA Development Boards in the shortest possible time. The format is an aggressive introduction to schematic, VHDL, and Verilog entry for those who want to get started quickly. The approach is tutorial and utilizes a path that is similar to most digital design processes.*

Once you have completed this tutorial, you will understand and be able to:

- Navigate the Altera schematic entry environment,
- Compile a VHDL or Verilog design file,
- Simulate, debug, and test your designs,
- Generate and verify timing characteristics, and
- Download and run your design on a DE1, DE2, UP3, UP2, or UP1 board.

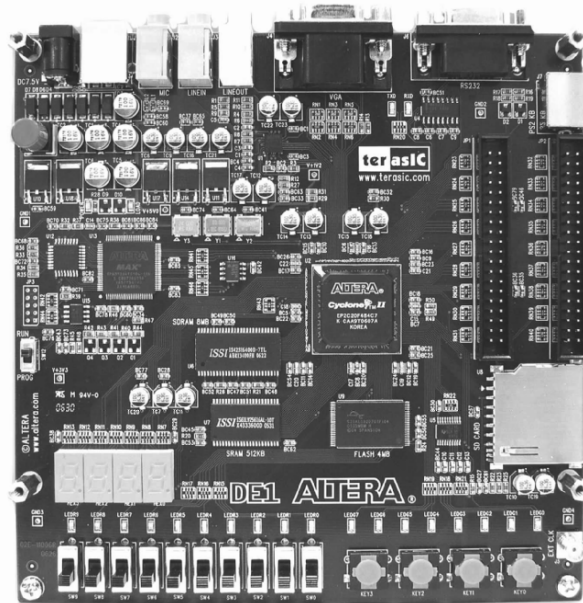
### Determining your FPGA Board Type

The first step is to identify which one of the various Altera Educational FPGA boards you are using for the tutorial. Examine the photographs in Figures 1.1 to 1.4 and compare them to your board to determine which type of board you are using.

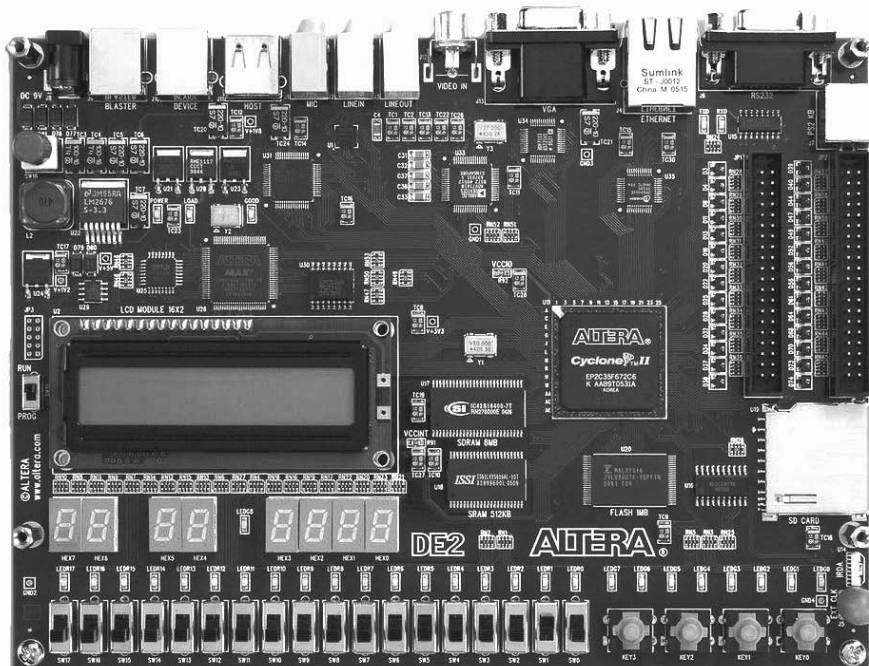
There will be some minor variations in the instructions later on that depend on which board type you are using. After identifying your board, be sure to remember which model of Altera FPGA board you have (i.e., DE1, DE2, UP3, UP2 or UP1).

If your board looks like Figure 1.4 and you see UP1X printed on the board, some early UP2 production boards had the designation UP1X printed on the board. The UP1X is electronically equivalent to a UP2 board and contains the same FPGA, so follow the instructions for a UP2 board.

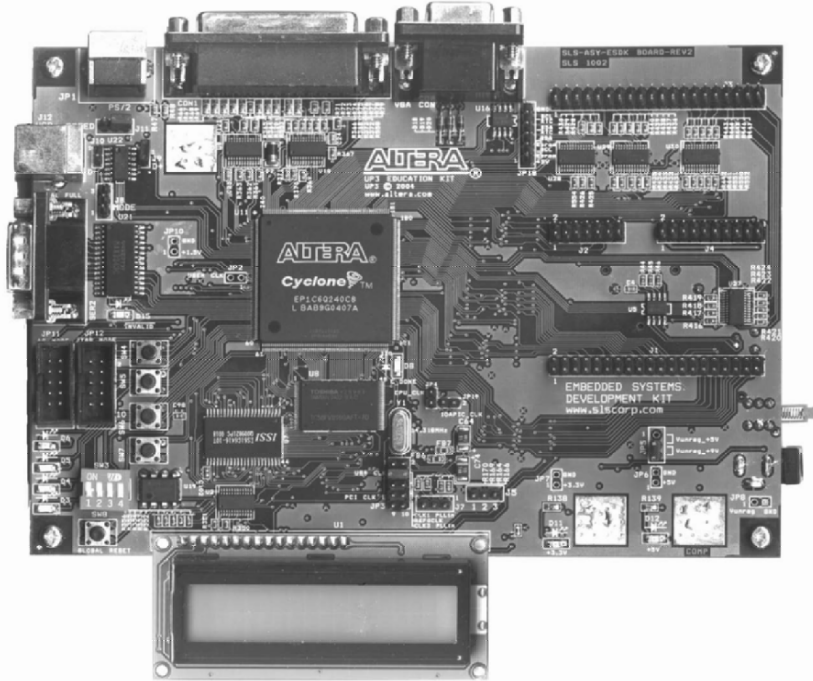
If you have a UP3 board, the UP3 board comes in two versions and you need to determine which version you have. The 1C12 version contains a larger EP1C12 FPGA instead of the EP1C6 FPGA. Check the part number on the FPGA chip on the left center of the board.



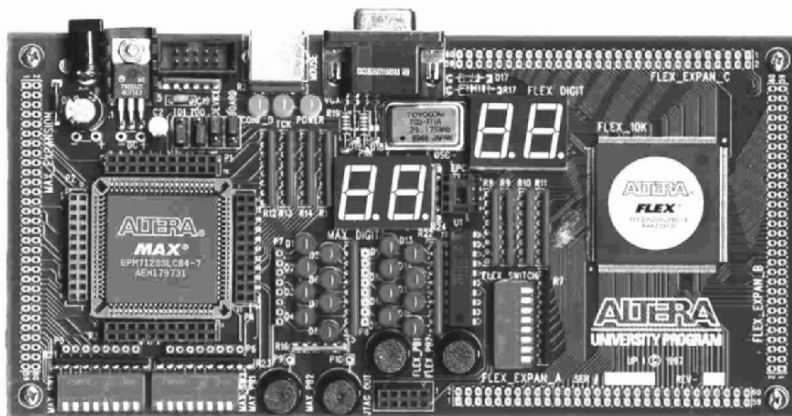
**Figure 1.1** The Altera DE1 FPGA Development board.



**Figure 1.2** The Altera DE2 FPGA Development board.



**Figure 1.3** The Altera UP3 FPGA Development board. The 1C12 version has a larger EP1C12Q240 FPGA. (Check the part number on the large square FPGA chip in middle of board).



**Figure 1.4** The Altera UP2 FPGA development board. UP1 boards appear very similar and can also be used, but instead of a FLEX EPF10K70 they use a smaller EPF10K20 FPGA that contains fewer logic elements. (Check the part number on the large square FPGA chip on right side of your board)

In this tutorial, a simple OR logic function will be demonstrated to provide an introduction to the Altera Quartus II CAD tools. After simulation, the design will then be used to program a field programmable gate array (FPGA) on an FPGA development board. The design will then be tested running on real hardware.

ALTERA'S NEWEST BOARDS ARE THE DE1 AND DE2. IF YOU HAVE ONE OF THE OTHER FPGA BOARDS AS SEEN IN FIGURES 1.1 TO 1.4, YOU SHOULD ALWAYS FOLLOW THE INSTRUCTIONS AND PROCEDURES OUTLINED IN THE TEXT AND ON THE DVD FOR YOUR BOARD. THE DVD CONTAINS ADDITIONAL INFORMATION AND FILES FOR THE OLDER FPGA BOARDS.

The inputs to the OR logic will be two pushbuttons and the output will be displayed using a light emitting diode (LED). Both the pushbuttons and the LED are part of the development board, and no external wiring is required.

Of course, any actual design will be more complex, but the objective here is to quickly understand the capabilities and flow of the design tools with minimal effort and time.

More complex digital designs including computers and color video graphics will be introduced later in this text after you have become familiar with the development tools and hardware description languages (Hals) used in digital designs.

Granted, all this may not be accomplished in just 15 minutes; however, the skills acquired from this demonstration tutorial will enable the first-time user to duplicate similar designs in less time than that!

INSTALL THE QUARTUS II WEB VERSION SOFTWARE USING THE DVD AND OBTAIN A WEB LICENSE FILE FROM ALTERA. CHECK FOR ALTERA QUARTUS II WEB VERSION SOFTWARE UPDATES AT [WWW.ALTERA.COM](http://WWW.ALTERA.COM). THE BOOKS DESIGNS WERE ALL TESTED WITH VERSION 7.1 SP1. AS NEWER VERSIONS OF THE ALTERA SOFTWARE APPEAR, MINOR CHANGES MAY BE NEEDED. CHECK THE BOOK'S WEBSITE FOR ANY UPDATES.

With the standard FPGA computer aided design tools, designs can be entered via schematic capture or by using a Hardware Description Language (HDL) such as VHDL or Verilog. It is also possible to combine blocks with different entry methods into a single design. As seen in Figure 1.5, tools can then be used to simulate, calculate timing delays, synthesize logic, and program a hardware implementation of the design on an FPGA.

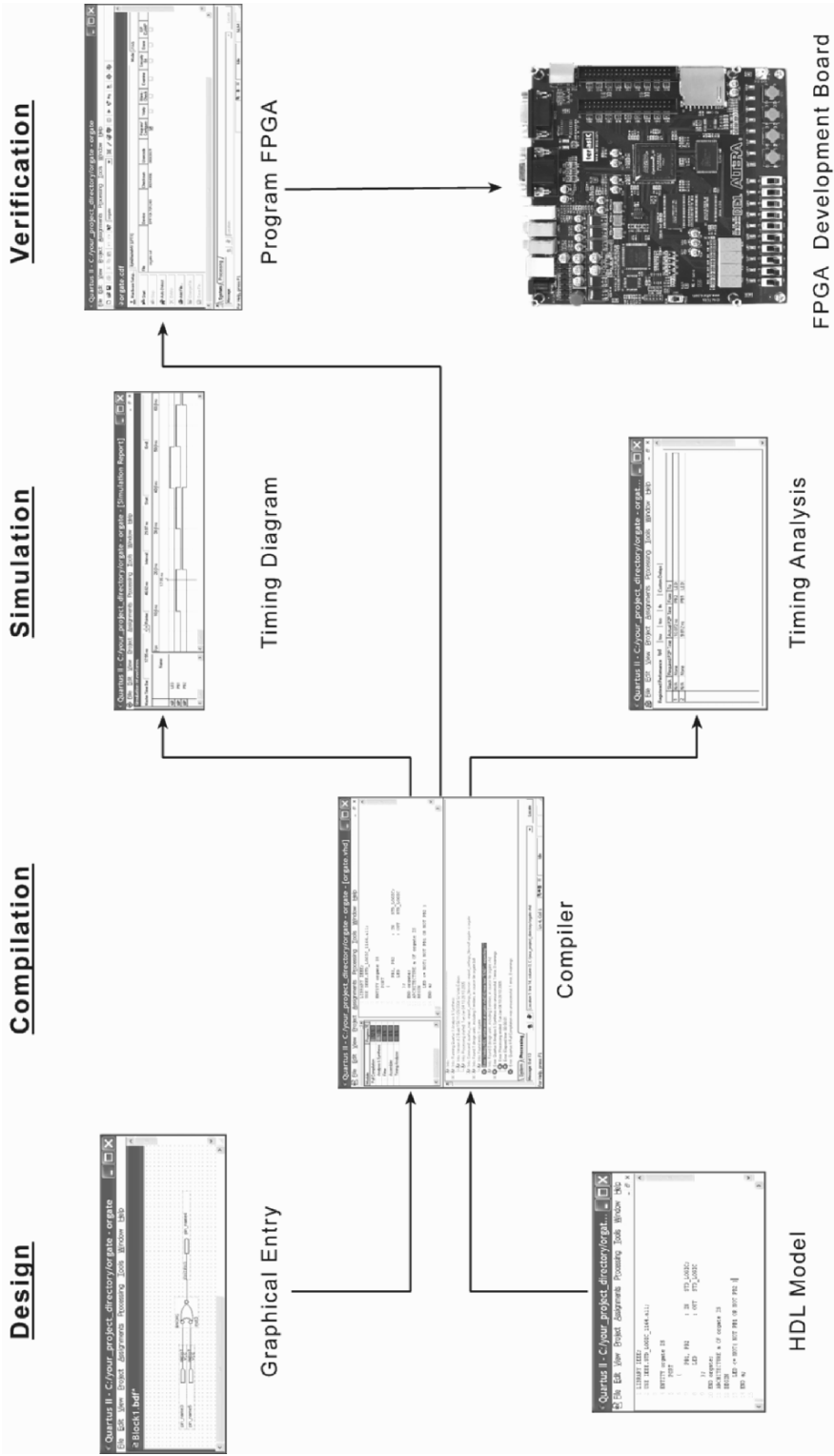


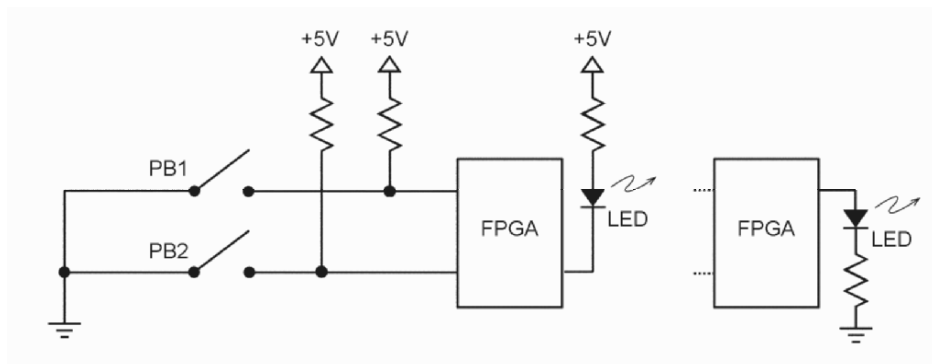
Figure 1.5 Design process for schematic or HDL entry.

## The Board

The default boards that will be used are the newer DE1 and DE2 boards. Although the following tutorial can be done with any of the DE1, DE2, UP3, UP2 or UP1 boards. Some minor modifications (i.e., the FPGA device number and pin number assignments) will be needed for the other boards. In this tutorial, complete instructions will be provided for all of the different FPGA boards.

## The Pushbuttons

On the FPGA board, two pushbutton switch inputs, PB1 and PB2, are connected to FPGA input pins. Each pushbutton input is tied “High” with a pull-up resistor and pulled “Low” when the respective pushbutton is pressed. One needs to remember that when using the on-board pushbuttons, this “active low” condition ties zero volts to the input when the button is pressed and the  $V_{cc}$  high supply to the input when not pressed. See Figure 1.6.  $V_{cc}$  is 3.3V on newer boards and 5V on the older UP2 and 1 boards. On the FPGA board shown on the left in Figure 1.6, a logic “0” at the output turns on the LED.



**Figure 1.6** FPGA I/O connections to Pushbuttons (PBx) and LED: Right of center, active LOW LED output (i.e., UP1 and UP2 boards) or on far right active HIGH LED output (i.e., DE1, DE2, and UP3 boards). Note that a depressed pushbutton input will be LOW.

## The LED Outputs

On many of the newer FPGA development boards including the DE1 and DE2, the LED is connected with active HIGH outputs as in the right LED output illustration in Figure 1.6. This allows a HIGH signal on the output pin of the FPGA to turn the LED on and a LOW signal to turn it off.

Historically, most digital logic output pins were designed to sink more current than they could source. In such cases, it makes sense to let the output pin of the FPGA tie the cathode (-) of the LED to ground to allow for a brighter LED (i.e., more current flowing); however, this configuration has the side effect of requiring a LOW signal to turn the LED on, but is generally the more common

configuration. This active LOW output is actually the arrangement used on all seven segment LED displays on all of the FPGA development boards.

### The Problem Definition

To illustrate the capabilities of the software in the simplest terms, we will build a circuit that turns off the LED when one OR the other pushbutton is pushed. In a simple logic equation, one could write:

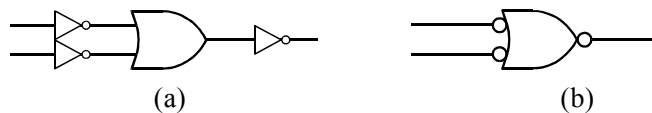
$$\text{LED\_OFF} = \text{PB1\_HIT} + \text{PB2\_HIT}$$

At first, this may seem too simple; however, the active low inputs and outputs add just enough complication to illustrate some of the more common errors, and it provides an opportunity to compare some of the different syntax features of VHDL and Verilog. (Students needing an exercise in DeMorgan's Law will also find these exercises particularly enlightening.)

We will first build this circuit with the graphical editor and then implement it in VHDL and Verilog. As you work through the tutorial, note how the design entry method is relatively independent of the compile, timing, and simulation steps, and which FPGA board is used for the hardware implementation.

### Resolving the Active Low Signals

Since the pushbuttons generate inverted signals and the LED requires an inverted or low level logic signal to turn off, we could build an OR logic circuit using the layout in Figure 1.7a. Recalling that a bubble on a gate input or output indicates inversion, careful examination shows that the two circuits in Figure 1.7 are functionally equivalent; however, the circuit in Figure 1.7a uses more gates and would take a bit longer to enter in the schematic editor. We will therefore use the single gate circuit illustrated in Figure 1.7b.



**Figure 1.7a and 1.7b.** Equivalent circuits for ORing active low inputs and outputs.

This form of the OR function is known as a "negative-logic OR." If you are confused, try writing a truth table to show this Boolean equality. (In Exercise 1 at the end of the chapter, this circuit will be compared with its DeMorgan's equivalent, the "positive-logic AND.")

ON THE UP2 AND UP1 BOARDS, THE LED'S OUTPUT STATE WILL APPEAR INVERTED SINCE ITS LED OUTPUT CIRCUIT IS INVERTED, SO PUSHING ONE OF THE PUSHBUTTONS WILL TURN ON THE LED.



## 1.1 Design Entry using the Graphic Editor

Examine the CAD tool overview diagram in Figure 1.5. The initial path in this section will be from schematic capture (Graphical Entry) to downloading the design to the FPGA board. On the way, we will pass through some of the nuances of the Compiler along with setting up and controlling a simulation. Later, after having actually tested the design, we will examine the Timing Analysis information of the design. Although relatively short, each step is carefully illustrated and explained. Install the Altera Quartus II software on your PC using the book's DVD, if it is not already installed.



**Figure 1.8** Creating a new Quartus II Project.

### New Project Creation

Start the Quartus II program. In Quartus II, the New Project wizard is used to create a new project. Choose **File ⇒ New Project Wizard**. Click next in the Introduction window, if it appears to continue. A second dialog box will appear asking for the working directory for your new project. Enter an appropriate directory. For the project name and top-level design entity boxes, enter **orgate**. Click **Next**. If you need to create a new project directory with that name, click **Yes**. An Add Files dialog box then appears. This page is used to enter all of the

design files (other than the top-level file). Since this simple project will only use a single top-level design file, click **Next**.

### Select the FPGA Device to be Used

The next dialog box is used to select the FPGA device type as seen in Figure 1.9. The detailed instructions for this step will vary depending on your board type.

A summary of the different FPGA devices found on each board is shown in Table 1.1. Find your board's FPGA family and device part number in Table 1.1 and follow the specific instructions below for your board.

**New Project Wizard: Family & Device Settings**

Select the family and device you want to target for compilation.

Family: **Cyclone II**

Target device:

- ☐ Auto device selected by the Fitter
- ☒ Specific device selected in 'Available devices' list

Show in 'Available device' list:

Package: **Any**

Pin count: **Any**

Speed grade: **7**

☒ Show advanced devices

☐ HardCopy compatible only

Available devices:

Name	Core v...	LEs	User I/...	Memor...	Embed...	PLL
EP2C15AF256C7	1.2V	14448	152	239616	52	4
EP2C15AF484C7	1.2V	14448	315	239616	52	4
EP2C20F256C7	1.2V	18752	152	239616	52	4
EP2C20F484C7	1.2V	18752	315	239616	52	4
EP2C35F484C7	1.2V	33216	322	483840	70	4
EP2C35F672C7	1.2V	33216	475	483840	70	4
EP2C35U484C7	1.2V	33216	322	483840	70	4
EP2C50F484C7	1.2V	50528	294	594432	172	4

Companion device:

HardCopy II: **HardCopy II:**

☒ Limit DSP & RAM to HardCopy II device resources

< Back   **Next >**   Finish   Cancel

**Figure 1.9** Setting the FPGA Device Type. Settings shown are for the DE1 board.

Table 1.1 FPGA Devices used on the various Altera Educational FPGA boards.

	<b>DE1</b>	<b>DE2</b>	<b>UP3</b>	<b>UP2 &amp; UP1</b>
<b>FPGA Family</b>	Cyclone II	Cyclone II	Cyclone	FLEX10K
<b>FPGA Device</b>	EP2C20F484C7	EP2C35F672C6	EP1C6Q240C8 or 1C12 version EP1C12Q240C8	EPF10K70RC240-4 or on older UP1s EPF10K20RV240-4

**If you are using a DE1 or DE2 board:**

Select the **Cyclone II** Family FPGA. The DE1 uses an **EP2C20F484C7** device and the DE2 uses a **EP2C35F672C6** device.

**If you are using the UP3 board:**

Select **Cyclone** family. You will then need to select the specific FPGA device number on your board. The UP3 is available with two different sizes of Cyclone FPGAs: an **EP1C6Q240C8** (UP3 1C6 board) or the larger **EP1C12Q240C8** (UP3 1C12 board). Check the large square chip in the middle of the board to verify the exact FPGA part number.

**If you are using a UP2 or UP1:**

Select **FLEX10K** family. On the UP2, it will be a **EPF10K70RC240-X** (-X is the speed grade of the chip). Check the large square chip in the middle on the right side of the board to verify the FPGA part number. If you see **EPF10K20RC240** on the FPGA, you have a UP1 board. The original UP1 boards can also be used. The UP1 looks very similar to a UP2 and they use the same pin assignments and have the same I/O features as the UP2, but they contain a EPF10K20RC240 FPGA with fewer logic elements. UP1 users should always follow the instructions for the UP2, but specify the EPF10K20RC240 device type for each project.

**All Boards:**

The last digit in the FPGA part number is the speed grade. The correct speed grade is needed for accurate delays in timing simulations. You may need to change the setting of the Speed Grade on Pin Count dialog box to **Any** to display your specific device. Always choose the correct speed grade to match your board's FPGA so that the correct delay times are used in the logic simulation tools.


If you choose the wrong device type, you will have errors when you attempt to download your design to the FPGA.

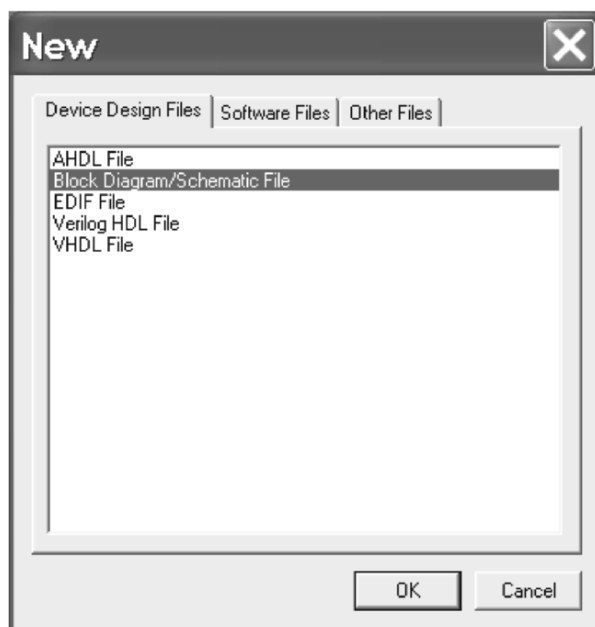
After selecting the correct FPGA part number, click **Next**, then on the third-party EDA tools settings box also click **Next** since we will not be using any third-party EDA tools – only Quartus II. Double check the information summary page that appears and click **Finish**. In case of problems, use the back option to make changes.

## Establishing Graphics (Schematic) as the Input Format


You have now named your project and setup which FPGA will be used, now choose **File** ⇒ **New**, and a popup menu will appear. Select **Block Diagram/Schematic File**, then click **OK**. This will create a blank schematic worksheet – a graphics display file (\*.gdf file). Note that the toolbar options in Quartus II are context sensitive and change as different tools are selected. An empty schematic window with grids will appear named Block1.bdf.

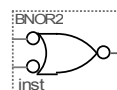
## Enter and Place the OR Symbol in Your Schematic

Click on the AND gate icon  on the left-side toolbar. This selects the symbol tool. In the upper left box under Libraries, expand the library path to see the options. Find the library named primitives and click on it to expand it. Then click to expand the logic library. Scroll down the list of logic symbols and select BNOR2. An OR gate with inverted inputs and outputs should appear in the symbol window. (The naming convention is B-bubbled NOR with 2 inputs. Although considered to be a NOR with active low inputs, it is fundamentally an OR gate with active low inputs and output.) Click OK at the bottom of the Symbol window.



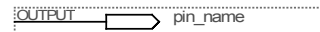
**Figure 1.10** Creating the top-level project schematic design file.

Select the Block1.bdf window and the BNOR2 symbol will appear in the schematic. Drag the symbol to the middle of the window and **left click** to place it. **Click on the arrow icon**  on the left side toolbar or hit escape to stop inserting the symbol.

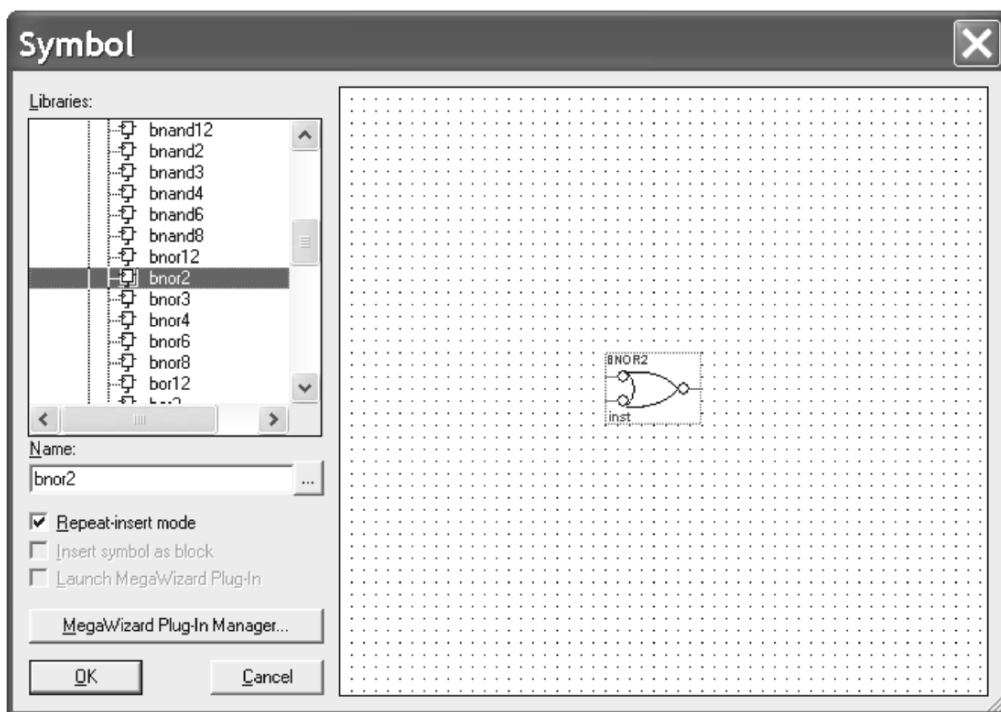


TO USE THE ONLINE HELP SYSTEM, CLICK **HELP** ON THE TOP MENU, SELECT SEARCH AND THEN ENTER **BNOR**. AT ANY POINT IN THE TUTORIAL, EXTENSIVE ONLINE HELP IS ALWAYS AVAILABLE. TO SEARCH BY TOPIC OR KEYWORD SELECT THE **HELP** MENU AND FOLLOW THE INSTRUCTIONS THERE.

### Assigning the Output Pin for Your Schematic

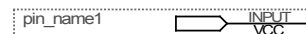


Select the **AND gate symbol** again on the left side toolbar, expand the **pin library**, select **output**, and click **OK**. Using the mouse and the left mouse button, drag the output symbol to the right of the **BNOR2** symbol leaving space between them – they will be connected later.



**Figure 1.11** Selecting a new symbol with the Symbol Tool.

### Assigning the Input Pins for Your Schematic



Find and place two pin **input** symbols to the left of the **BNOR2** symbol in the same way that you just selected and placed the **output** symbol. (Another hint: Once selected, a symbol can be copied with **Right Click** ⇒ **Copy** and pasted multiple times using the **Right Click** ⇒ **Paste** function.). Hit the arrow symbol on the left tool bar and deselect the new symbol by moving the cursor away and clicking the left mouse button a second time.

Connecting the Signal Lines

When you move the mouse cursor near a wire, the cursor changes into a crosshair. Move to one end of a wire you need to add and push and hold down the left mouse button. Hold down the left mouse button and drag the end of the wire to the other point that you want to connect. Release the left button to connect the wire. If you need to delete a wire, click on it – the wire should turn blue when selected. Hit the delete key to remove it. You can also use the **Right Click ⇒ Delete** function. Connect the other wires using the same process so that the diagram looks something like Figure 1.12

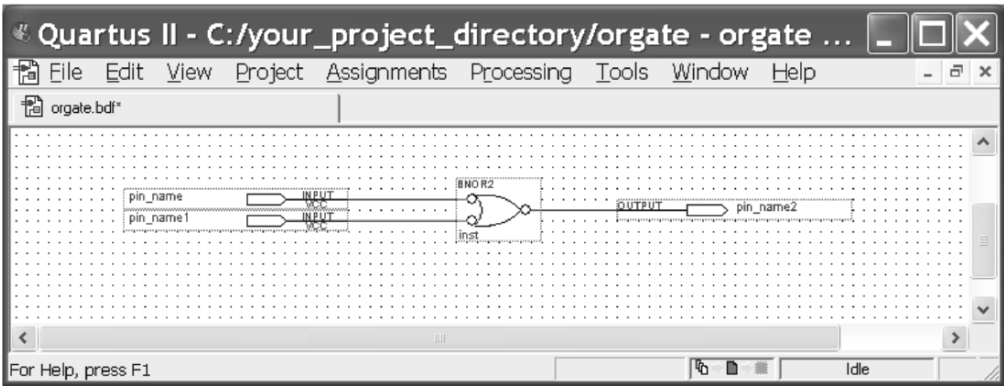


Figure 1.12 Active low OR-gate schematic example with I/O pins connected.

Enter the PIN Names

Right click on the first **INPUT** symbol. It will be outlined in blue and a menu will appear. Select **Properties**, type **PB1** for the pin name and click **OK**. Name the other input pin **PB2** and the output pin for the **LED** in a similar fashion.

Assign the PIN Numbers to Connect the Pushbuttons and the LED

Since the FPGA chip is already prewired to the pushbuttons and the LED on the printed circuit board (PCB), you need to look up the correct pin numbers and designate them in your design.

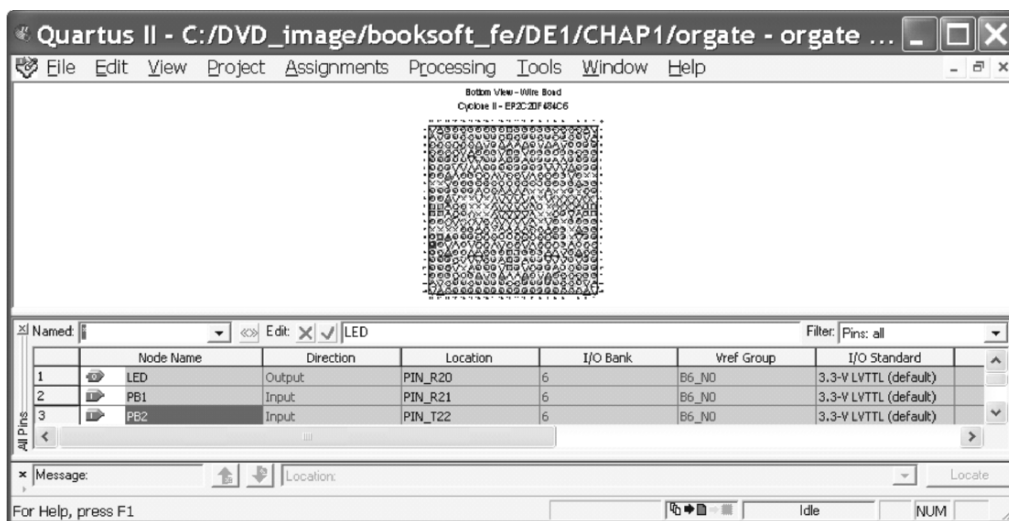
Table 1.2 Hardwired I/O connections on the various FPGA boards in the design example.

I/O Device	DE1 Pin	DE2 Pin	UP3 Pin	UP2 & 1 Pin
<b>PB1</b>	R21 (Key1)	N23 (Key1)	62 (SW7)	28 (FLEX PB1)
<b>PB2</b>	T22 (Key2)	P23 (Key2)	48 (SW4)	29 (FLEX PB2)
<b>LED</b>	R20(LED R0)	AE23(LED R0)	56 (D3)	14 (7Seg Dec. pt.)

The information in Table 1.2 is from the documentation on the pinouts for the various FPGA board user’s manuals. (Table 2.4 lists additional I/O pins) In the

main menu, select **Assignments** ⇒ **Pin**. (If the option to select the pin is unavailable, you need to go back and select **Assignments** ⇒ **Device**, and make sure that your device is selected correctly.) You may need to adjust the Window sizes to find the pin assignment area or select the pulldown, **View** ⇒ **All Pins List**; Figure 1.13 shows the pin list area with pin information entered. In the “Node Name” column, type the name of the new pin, **PB1**. In the “Location” column, double click, scroll down, and **select the pin number for PB1 on your board** or type in the pin number in the blank space provided (**NOTE:** pin numbers will be different on the various FPGA boards refer to Table 1.2). The software adds PIN\_ to the pin number. Repeat this process assigning **PB2** and **LED** to the correct pins for your board. After assigning all three pins and verifying your entries, click **File** ⇒ **Save Project** to save. Device and pin information is stored in the project’s \*.qsf file. Pin names are also case sensitive.

**CAUTION:** BE SURE TO USE UNIQUE NAMES FOR DIFFERENT PINS. PINS AND WIRES WITH THE SAME NAME ARE AUTOMATICALLY CONNECTED EVEN WITHOUT A VISIBLE WIRE SHOWING UP ON THE SCHEMATIC DIAGRAM.



**Figure 1.13** Assigning Pins with the Assignment Editor.

## Saving Your Schematic

If you haven’t saved your file yet, Select **File** ⇒ **Save As** and save your project using the filename **ORGATE**. Throughout the remainder of this tutorial you should always refer to the same project directory path when saving or opening files. A number of other files are automatically created by the Quartus II tools and maintained in your project directory.

### Set Unused Pins as Inputs

The memory chips on the development board could all be turned on at the same time by unused pins on the FPGA, causing their tri-state output drivers to try to force output data bus bits to different states. This causes high currents, which can overheat and damage devices after several minutes. To eliminate the possibility of any damage to the board, the following option should always be set in a new project. On the menu bar, select **Assignments ⇒ Device** then click the **Device and Pin Options** button. Click on the **Unused Pins** tab and check the **As inputs, tri-stated** option. Click **OK** and then **OK** in the first window. This setting is saved in the projects \*.qsf file. Any time you create a new project repeat this step.

## 1.2 Compiling the Design

Compiling your design checks for syntax errors, synthesizes the logic design, produces timing information for simulation, fits the design on the selected FPGA, and generates the file required to download the program. After any changes are made to the design files or pin assignments, the project will always need to be re-compiled prior to simulation or downloading.

### Compiling your Project

Compile by selecting **Processing ⇒ Start Compilation**. The compilation report window will appear in the Quartus II screen and can be used to monitor the compilation process, view warnings, and errors.

### Checking for Compile Warnings and Errors

The project should compile with **0 Errors**. If a popup window appears that states, "Full Compilation was Successful," then you have not made an error. Info messages will appear in green in the message window. Warnings appear in blue in the message window and Errors will be red. Errors must be corrected. If you forget to assign pins, the compiler will select pins based on the best performance for internal timing and routing. Since the pins for the pushbuttons and the LED are pre-wired on the FPGA boards, their assignment cannot be left up to the compiler and the user must always specify them.

### Examining the Report File

After compilation, the compiler window shows a summary of the compiled design including the FPGA logic and memory resources used by the design.

Select the **orgate.bdf** schematic window. Use **View ⇒ Show Location Assignments** and check the schematic's I/O pins to verify the correct pin numbers have been assigned. If a pin is not assigned you may have a typo somewhere in one of the pin names or you did not save your pin assignments earlier. You will need to recompile whenever you change pin assignments.

You can also check all of the FPGA's pins by going to the compiler report window with **Processing ⇒ Compilation Report**, expanding the **Fitter file folder**, and clicking on the **Pin-out file**.



### 1.3 Simulation of the Design

For complex designs, the project is normally simulated prior to downloading to a FPGA. Although the OR example is straightforward, we will take you through the steps to illustrate the simulation of the circuit.

#### Set Up the Simulation Traces

Choose **File** ⇒ **New**, select the **Other Files** tab, and then from the popup window select **Vector Waveform File** and click **OK**. A blank waveform window should be displayed. Right click on the **Name** column on the left side. Select **Insert Nodes or Bus**. Click on the **Node Finder** and then the **LIST** button. PB1, PB2 and LED should appear as trace values in the window. Then click on the center >> button and click **OK** and **OK** again. The signals should appear in the waveform window.

#### Generate Test Vectors for Simulation

A simulation requires external input data or "stimulus" data to test the circuit. Since the PB1 and PB2 input signals have not been set to a value, the simulator sets them to a default of zero. The 'X' on the LED trace indicates that the simulator has not yet been run. (If the simulator *has* been run and you still get an 'X,' then the simulator was unable to determine the output condition.)

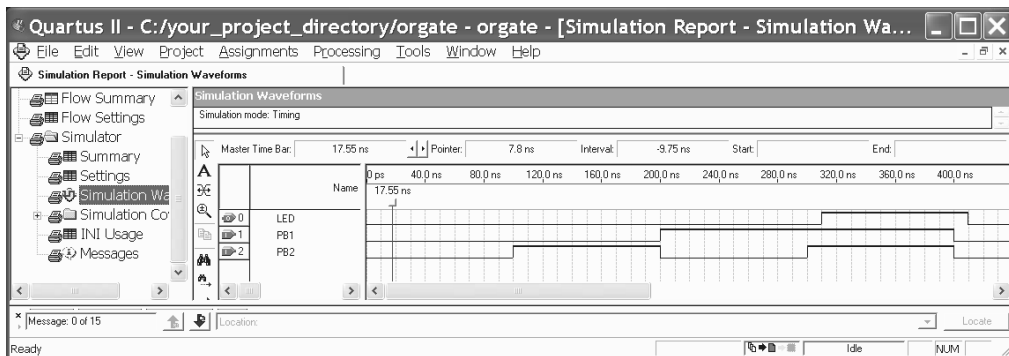
Right click on **PB1**; the PB1 trace will be highlighted. Select **Value** ⇒ **Count Value**, then click on the **Timing** tab and change the entry in the field **Multiplied By** from **1** to **5** and click **OK**. An alternating pattern of Highs and Lows should appear in the PB1 trace. (Use **View** ⇒ **Zoom Out**, if you cannot see the pattern.) Repeat the procedure for **PB2** but this time change the entry in **Multiplied By** from **1** to **10**. **PB2** should now be an alternating pattern of ones and zeros but at twice the frequency of PB1.

(Other useful options in the **Value** menu will generate a clock and set a signal High or Low. It is also possible to highlight a portion of a signal trace with the mouse and set the level manually.)

When you need a longer simulation time in a waveform, you can change the default simulation end time using **Edit** ⇒ **End Time**.

#### Performing the Simulation with Your Timing Diagram

Select **File** ⇒ **Save** and click the **Save** button to save your project's vector waveform file. Select **Processing** ⇒ **Start Simulation** and click **OK** on the window that appears. The simulation should run and the output waveform for LED should now appear in the Simulation Report window. You may want to right click on the timing display and use the Zoom options to adjust the time scale as seen in Figure 1.14.



**Figure 1.14** Active low OR-gate timing simulation with device time delays.

Note that the simulation includes the signal propagation timing delays through the FPGA and that it takes almost 10 ns ( $\text{ns} = 10^{-9} \text{ sec.}$ ) for an input change to be reflected in the delayed output. Taking this LED output time delay into account, examine the Simulation Waveform to verify that the LED output is Low only when either PB1 OR PB2 inputs are Low.

## 1.4 Testing Your Design on an FPGA Board

The next step is to download the design to a board and test it on real hardware. At this point, the instructions vary depending on which type of board you are using for the tutorial. If you do not know your board type, refer back to Figures 1.1 to 1.4 to identify it. You will need to skip ahead to the appropriate section for each board as listed below:

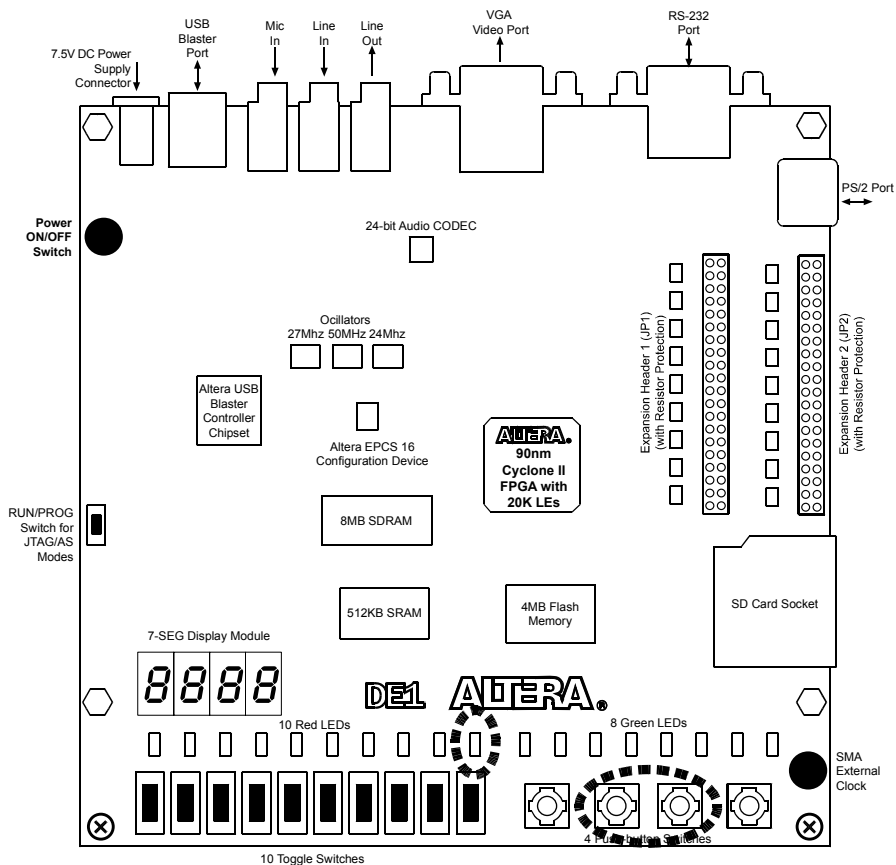
- DE1 users go to Section 1.5 (next Section),
- DE2 users skip ahead to Section 1.6,
- UP3 users skip ahead to Section 1.7,
- UP2 & UP1 users skip ahead to Section 1.8.

## 1.5 Downloading Your Design to the DE1 Board

### Hooking Up the DE1 Board to the Computer

If you have a DE2, UP2, or UP3 board refer to the specific download Section on those boards and skip this section. To try your design on a DE1 board as seen in Figure 1.15, plug the USB download cable into the DE1 board's USB connector (top left corner of the board) and attach the other end to an open USB port on the PC.

The DE1 board is normally powered using only the USB download cable's 5V power. The 7.5V AC to DC wall transformer supplied with the board attaches to the DC power connector located on the upper left corner of the DE2 board and can be used to supply power to the board when it is not attached to the PC (i.e., standalone operation).



**Figure 1.15** The Altera DE1 board showing Pushbutton and LED locations used in the design (enclosed in dashed ellipses seen at bottom of board).

After attaching the USB cable, press in the DE1's red power switch located on the upper left edge of the board below the USB connector to turn on the board. When properly powered, the blue power LED on the DE1 board near the power switch should light up and the other LEDs will all flash, if it is still setup to run the default demo design shipped from the factory.

### Preparing for Downloading

Make sure that you have assigned the correct **Device Name** for the DE1. The DE1 contains a **EP2C20F484C7** Cyclone II FPGA. When you change the device type you will also need to redo the pin assignments. Make sure that you have also assigned the correct pin numbers for the DE1 board. PB1 is pin **R22**, PB2 is pin **R21** and the LED is pin **U22**. (refer to section 1.1 for help) Whenever you change the Device or pin assignments, it is necessary to recompile before downloading.

After checking to make sure that the cables are hooked up properly, you are ready to download the compiled circuit to the DE1 board. Select **Tools ⇒ Programmer**. Click on **Hardware Setup**, select the proper hardware, a **USB-Blaster**. (If a window comes up that displays, "**No Hardware**" to the right of the Hardware Setup button, use the Hardware Setup button to change currently selected hardware from "**No Hardware**" to "**USB-Blaster**". If a red JTAG error message appears or the start button is not working, close down the Programmer window and reopen it. If this still doesn't correct the problem, then there is something else wrong with the setup or cable connection. Go back to the beginning of this section and check each step and connection carefully.)

### Final Steps to Download

The filename `orgate.sof` should be displayed in the programmer window. The \*.sof file contains the FPGA's configuration (programming) data for your design. To the right of the filename in the Program/Configure column, check the **Program/Configure** box. To start downloading your design to the board, click on the **Start** button. Just a few seconds are required to download. If download is successful, a green info message displays in the system window notifying you the programming was successful.

### Testing Your Design

On the DE1 board, the right two pushbuttons are used in the design and one of the LEDs to the right just above them. The locations of PB1 (Key0), PB2 (Key1), and the LED (LEDG0) are in the lower right corner of the board as seen in Figure 1.13. After downloading your program to the DE1 board, the LED in the lower right corner should turn off whenever a pushbutton is hit. Since the output of the OR gate is driving the LED signal, it should be on when no pushbuttons are hit. Since the buttons are active low, and the **BNOR2** gate also has active low inputs and output, hitting either button should turn off the LED.

Congratulations! You have just entered, compiled, simulated, downloaded a design to a FPGA device, and verified its operation. Since you are using a DE1 board, you can skip the next three sections on the DE2, UP3 or UP2 board and go directly to Section 1.9.



COMPLETED TUTORIAL FILES ARE AVAILABLE ON THE TEXT'S DVD.

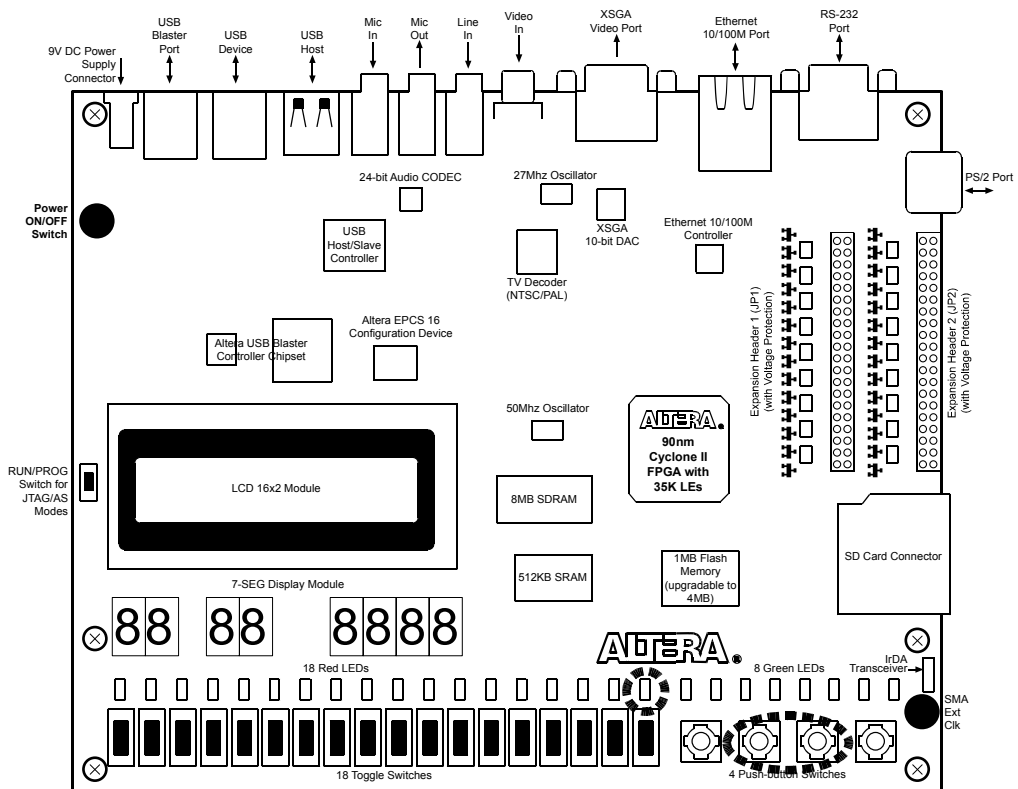
IN THE BOOK'S DESIGN EXAMPLES, ADDITIONAL DE1 RELATED MATERIALS  
CAN BE FOUND IN THE **BOOKSOFT\_FE\DE1\CHAPX** DIRECTORIES.

## 1.6 Downloading Your Design to the DE2 Board

### Hooking Up the DE2 Board to the Computer

If you have a DE1, UP2 or UP3 board refer to the download Sections on those boards. To test your design on a DE2 board as seen in Figure 1.16, plug the USB download cable into the DE2 board's USB connector (leftmost of the three USB connectors on the top left side of the board) and attach the other end to an open USB port on the PC. Using the 9V AC to DC wall transformer attach power to the DC power connector located on the upper left corner of the DE2 board.

Press the power switch located on the upper left edge of the board below the power connector. When properly powered, the blue power LED on the DE2 board next to the power switch should light up and the other LEDs will flash, if it is still setup to run the default demo design shipped from the factory.



**Figure 1.16** Altera DE2 board showing the Pushbutton and LED locations used in design (enclosed in dashed ellipses seen in bottom right).

## Preparing for Downloading

Make sure that you have assigned the correct **Device Name** for the DE2. The DE2 contains a **EP2C35F672C6** Cyclone II FPGA. When you change the device type you will also need to redo the pin assignments. Make sure that you have also assigned the correct pin numbers for the DE2 board. PB1 is pin **N23**, PB2 is pin **P23** and the LED is pin **AE23**. (refer to section 1.1 for help) Whenever you change the Device or pin assignments, it is necessary to recompile before downloading.

After checking to make sure that the cables and jumpers are hooked up properly, you are ready to download the compiled circuit to the DE2 board. Select **Tools ⇒ Programmer**. Click on **Hardware Setup**, select the proper hardware, a **USB-Blaster**. (If a window comes up that displays, "**No Hardware**" to the right of the Hardware Setup button, use the Hardware Setup button to change currently selected hardware from "**No Hardware**" to "**USB-Blaster**". If a red JTAG error message appears or the start button is not working, close down the Programmer window and reopen it. If this still doesn't correct the problem, then there is something else wrong with the setup or cable connection. Go back to the beginning of this section and check each step and connection carefully.)

## Final Steps to Download

The filename `orgate.sof` should be displayed in the programmer window. The \*.sof file contains the FPGA's configuration (programming) data for your design. To the right of the filename in the Program/Configure column, check the **Program/Configure** box. To start downloading your design to the board, click on the **Start** button. Just a few seconds are required to download. If download is successful, a green info message displays in the system window notifying you the programming was successful.

## Testing Your Design

On the DE2 board, the middle two pushbuttons are used in the design and one of the LEDs to the left just above them. The locations of PB1, PB2, and the decimal LED are in the lower right corner of the board as seen in Figure 1.16. After downloading your program to the DE2 board, the LED in the lower right corner should turn off whenever a pushbutton is hit. Since the output of the OR gate is driving the LED signal, it should be on when no pushbuttons are hit. Since the buttons are active low, and the **BNOR2** gate also has active low inputs and output, hitting either button should turn off the LED.

Congratulations! You have just entered, compiled, simulated, downloaded a design to a FPGA device, and verified its operation. Since you are using a DE2 board, you can skip the next two sections on the UP3 or UP2/1 board and go directly to Section 1.9.



COMPLETED TUTORIAL FILES ARE AVAILABLE ON THE TEXT'S DVD.

IN THE BOOK'S DESIGN EXAMPLES, ADDITIONAL DE2 RELATED MATERIALS  
AND DESIGN FILES CAN BE FOUND IN THE **BOOKSOFT\_FE\DE2\CHAPx**  
DIRECTORIES.



## 1.7 Downloading Your Design to the UP3 Board

### Hooking Up the UP3 Board to the Computer

If you have a UP2 board skip ahead to Section 1.8. To try your design on a UP3 board, plug the Byteblaster™ II cable into the UP3 board's JTAG connector (innermost of the two connectors on the left side of the board) and attach the other end to the parallel port on the PC (USB port if you are using a USB Blaster). If you have not done so already, make sure that the PC's BIOS setting for the printer port is ECP or EPP mode. Using the 6V AC to DC wall transformer attach power to the DC power connector (DC\_IN) located on the lower right side of the UP3 board. Press in the power switch located on the right edge of the board above the power connector. When properly powered, two LEDs on the bottom of the UP3 board near the power connector should light up. A USB ByteBlaster can also be used, if one is available.

### Preparing for Downloading

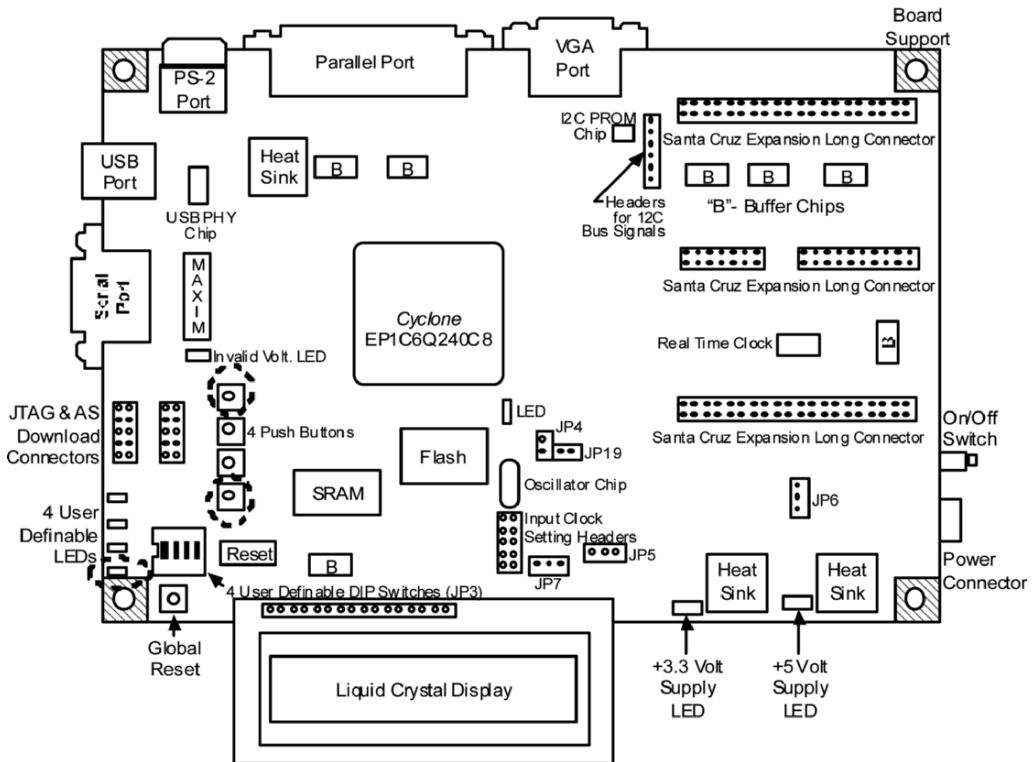
After checking to make sure that the cables are hooked up properly, you are ready to download the compiled circuit to the UP3 board. Select **Tools ⇒ Programmer**. Click on **Hardware Setup**, select the proper hardware, a **ByteBlasterII on LPT1**. (If a window comes up that displays, **No Hardware** to the right of the Hardware Setup button, use the Hardware Setup button to change currently selected hardware from **No Hardware** to **ByteblasterII**. If a red JTAG error message appears or the start button is not working, close down the Programmer window and reopen it. If this still doesn't correct the problem, then there is something else wrong with the setup or cable connection. Go back to the beginning of this section and check each step and connection carefully.) The new USB ByteBlaster can also be used.

### Final Steps to Download

The filename `orgate.sof` should be displayed in the programmer window. The \*.sof file contains the FPGA's configuration (programming) data for your design. To the right of the filename in the Program/Configure column, check the **Program/Configure** box. To start downloading your design to the board, click on the **Start** button. Just a few seconds are required to download. If download is successful, a green info message displays in the system window notifying you the programming was successful.

### Testing Your Design

The locations of PB1, PB2, and the decimal LED are indicated in Figure 1.17. After downloading your program to the UP3 board, the LED in the lower left corner should turn off whenever a pushbutton is hit. Since the output of the OR gate is driving the LED signal, it should be on when no pushbuttons are hit. Since the buttons are active low, and the **BNOR2** gate also has active low inputs and output, hitting either button should turn off the LED.



**Figure 1.17** ALTERA UP3 board. Pushbuttons and LED locations used in the tutorial design are seen in the lower left corner (enclosed in dashed ellipses). Silk-screening found on the board identifies each switch and LED.

Congratulations! You have just entered, compiled, simulated, downloaded a design to a FPGA device, and verified its operation. Since you are using a UP3 board, you can skip the next section on the UP2 board and go directly to Section 1.9.



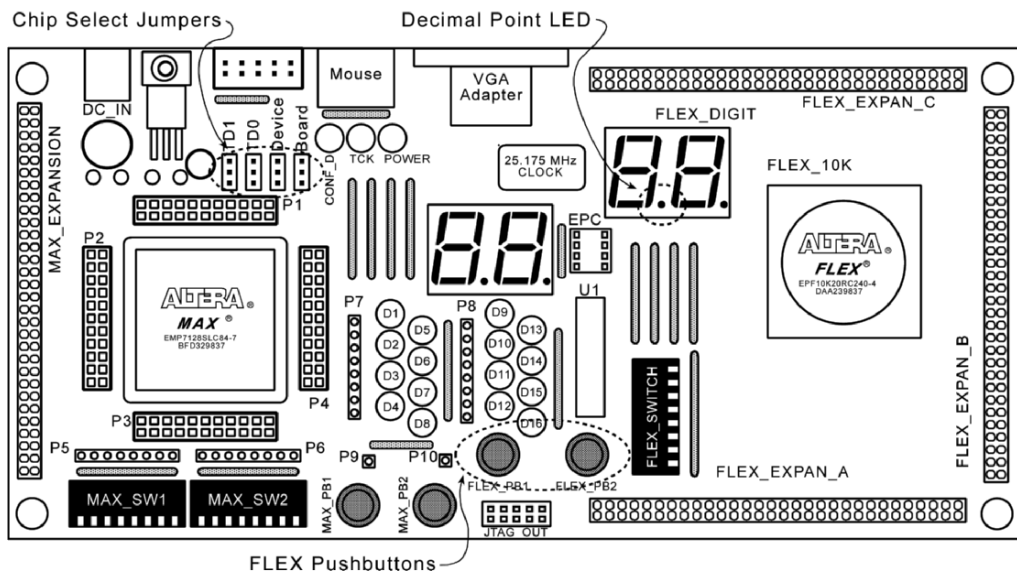
COMPLETED TUTORIAL FILES ARE AVAILABLE ON THE TEXT'S DVD.

IN THE BOOK'S DESIGN EXAMPLES, ADDITIONAL UP3 RELATED MATERIALS AND DESIGN FILES CAN BE FOUND IN THE **BOOKSOFT\_FE\UP3\CHAPX** DIRECTORIES. THE **\UP3\1C12\CHAPX** DIRECTORIES CONTAINS DESIGNS FOR THE LARGER FPGA USED ON THE 1C12 VERSION OF THE UP3

## 1.8 Downloading Your Design to the UP2 or UP1 Board

### Hooking Up the UP1 or UP 2 Board to the Computer

To try your design on a UP1 or UP2 board, plug the ByteBlaster cable into the UP board (top left) and attach the other end to the parallel port on a PC. If you have not done so already, make sure that the PC's BIOS setting for the printer port is ECP or EPP mode. Using a 9V AC to DC wall transformer or another 7 to 9V DC power source, attach power to the DC power connector (DC\_IN) located on the upper left-hand corner of the UP3 board. When properly powered, one of the green LEDs on the board should light up.



**Figure 1.18** ALTERA UP2 board with jumper settings and PB1, PB2, and LED locations.

Verify that the device jumpers are set for the FLEX chip as shown in Table 1.2. The locations of the pushbuttons, PB1 and PB2, and the LED decimal point are also highlighted in Figure 1.18. (Note that for the MAX EPM7128 chip, the jumper pins are all set to the top position as indicated in Table 1.2.)

Table 1.2 Jumper settings for downloading to the MAX and FLEX devices.

MAX	FLEX

### Preparing for Downloading

After checking to make sure that the cables and jumpers are hooked up properly, you are ready to download the compiled circuit to the UP2 board. Select **Tools** ⇒ **Programmer**. Click on **Hardware Setup**, select the proper hardware, a **ByteBlasterII** on **LPT1**. (If a window comes up that displays, "**No Hardware**" to the right of the Hardware Setup button, use the Hardware Setup button to change currently selected hardware from "**No Hardware**" to "**ByteBlasterII**". If a red JTAG error message appears or the start button is not working, close down the Programmer window and reopen it. If this still doesn't correct the problem, then there is something else wrong with the setup or cable connection. Go back to the beginning of this section and check each step and connection carefully.) The new USB ByteBlaster can also be used.

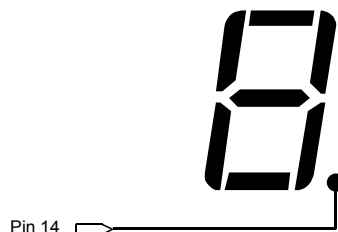
### Final Steps to Download

Make sure that the **Device Name** has changed to **EP10K20** or **EPF10K20RC240** for the UP1 or **EPF10K70RC240** for the UP2 (depending on the UP board and Quartus II version that you are running). Make sure you have also assigned the pin numbers for a UP2 board (see Table 1.1). If it does not display the correct device, then return to your schematic, assign the correct device first and then the pin numbers (See section 1.1.), recompile, and try again. Next, check the **Program/Configure** box.

The **Start** button in the programming window should now be highlighted. Click on the **Start** button to download to the UP2 board. Just a few seconds are required to download. If download is successful, a green info successful programmer operation message displays in the system window. (If the **Start** button is not highlighted, click **Hardware Setup** from the programmer window. Confirm the port settings and click **OK**. If you still have problems confirm that the printer port BIOS settings use ECP or EPP mode.)

### Testing Your Design

The locations of PB1, PB2, and the decimal LED are indicated in Figure 1.19. On the UP2, one of the seven-segment LEDs decimal points is used for monitoring the LED output.



**Figure 1.19** UP2's FLEX FPGA pin connection to seven-segment display decimal point.

All of these LEDs are pre-wired to the FPGA chip with a pull-up resistor as illustrated earlier in Figure 1.6. This configuration allows the external resistor

to control the amount of current through the LED; however, it also requires the FPGA chip to output a Low signal to turn on the LED. (Students regularly forget this point and have a fully working project with an inverted pattern on the LEDs.).  $V_{cc}$  is 5V on the UP2.

Figure 1.19 shows the UP2's Flex FPGA pin number 14 hard wired to the seven-segment LEDs decimal point. On the UP2, in this tutorial, only the decimal point will be used for output.

After downloading your program to the UP2 board, locate the two rightmost seven-segment displays. Since the output of the BNOR2 gate is driving the decimal LED signal on the left digit of the two seven-segment displays, it should be off (i.e., LED state is inverted on UP1 and UP2). Since the buttons are active low, and the **BNOR2** gate also has active low inputs and output, hitting either button should turn on the LED.

Congratulations! You have just entered, compiled, simulated, and downloaded a design to a FPGA device, and verified its operation.



COMPLETED TUTORIAL FILES ARE AVAILABLE ON THE TEXT'S DVD.

IN THE BOOK'S DESIGN EXAMPLES, ADDITIONAL UP2 RELATED MATERIALS CAN BE FOUND IN THE **BOOKSOFT\_FE\UP2\CHAPX** DIRECTORIES.

ADDITIONAL UP1 RELATED MATERIALS CAN BE FOUND IN THE **\UP1\CHAPX** DIRECTORIES.

## 1.9 The 10 Minute VHDL Entry Tutorial

As an alternative to schematic capture, a hardware description language such as VHDL or Verilog can be used. In large designs, these languages greatly increase productivity and reduce design cycle time. Logic minimization and synthesis to a netlist are automatically performed by the compiler and synthesis tools. (A netlist is a textual representation of a schematic.) As an example, to perform addition, the VHDL statement:

$$A \leq B + C;$$

will automatically generate an addition logic circuit with the correct number of bits to generate the new value of A. Using the OR-gate design from the Schematic Entry Tutorial, we will now create the same circuit using VHDL.

PRIOR KNOWLEDGE OF VHDL IS NOT NEEDED TO COMPLETE THIS TUTORIAL.

### Using a Template to Begin the Entry Process

Choose **File** ⇒ **New**, select **VHDL File** and **OK**. Place the cursor within the text area, right click the mouse, and select **Insert Template**. Make sure **VHDL** is selected. (Note the different prewritten templates. These are provided to expedite the entry of VHDL.) Select **VHDL** ⇒ **Constructs** ⇒ **Design Units** ⇒ **Entity Declaration**. This template is the one you will generally start with since it also sets up the input and output declarations. The template for the ENTITY declaration appears in the Insert Template preview window. Click **Insert** and then **Close** to paste the template in your VHDL window. Since the editor knows that it is a VHDL source file, the text will appear in different context-sensitive colors. VHDL keywords appear in blue and strings in purple and comments are green. The color information should be used to detect minor syntax errors while still in the text editor.

### Saving the VHDL Source File

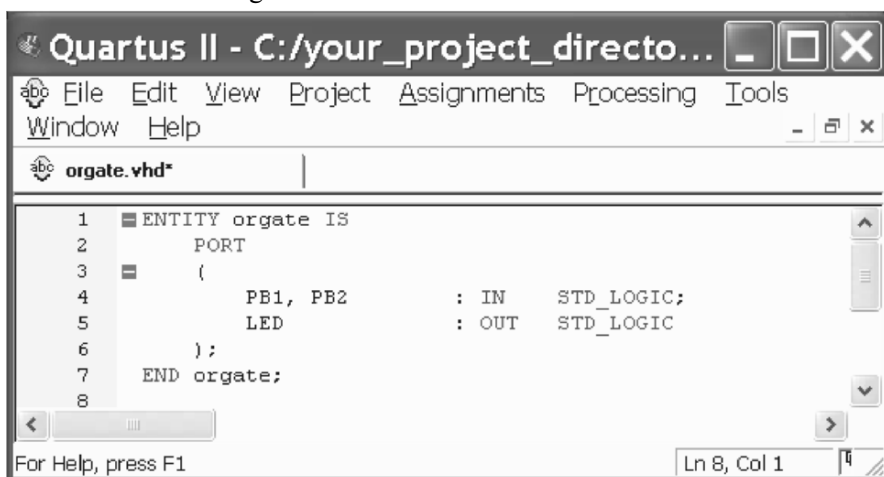
Select **File** ⇒ **Save As** and save the file as **orgate.vhd** – click **Save**.

### Replacing Comments in the VHDL Code

The entire string indicating the position of the entity name, **<entity\_name>**, should be set to the name used for the filename – in this case, **orgate**. There are two occurrences of **<entity\_name>** in the text. Find and change both accordingly.

### Declaring the I/O Pins

The input and output pins, PB1, PB2 and LED need to be specified in the PORT declaration. Since there are no input vectors, bi-directional I/O pins, or GENERIC declarations in this design, remove all of these lines. The source file should look like Figure 1.20.



**Figure 1.20** VHDL Entity declaration text.

### Setting up the Architecture Body

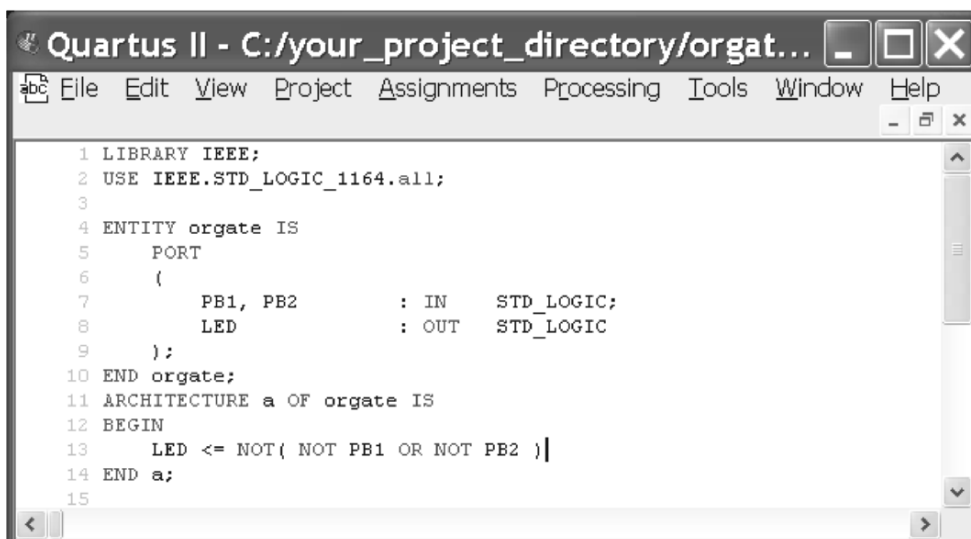
Click the mouse at the bottom of the text field. (We will be inserting another template here.) Following the earlier procedure for selecting a VHDL template (start with a right click), select **VHDL ⇒ Constructs ⇒ Design Units ⇒ Architecture Body**. (The **Architecture Body** specifies the internal logic of the design.) The syntax for the **Architecture Body** appears in the text window after the other text. (You can now see why the template is left highlighted – had you not placed your cursor first, text would have appeared at your last cursor position. If you do misplace the template, hitting the Edit Undo key removes the new text.)

### Editing the Architecture Body

Change the entity name in the **ARCHITECTURE** statement to **orgate**. Template lines with a "--" preceding a comment, need to be edited appropriately for each particular design. Delete the signal declaration line since this simple design does not require internal signals. Delete the remaining comment lines after BEGIN that start with "--", and insert **LED <= NOT ( NOT PB1 OR NOT PB2 )** as a single line. (This line contains a deliberate syntax error that will be detected and fixed later.) Insert the following two lines at the beginning of the text file to define the libraries for the STD\_LOGIC data type.

```
LIBRARY IEEE;  
USE IEEE.STD_LOGIC_1164.all;
```

This is the preferred data type for bits in VHDL. The file should now appear similar to Figure 1.21.



**Figure 1.21** VHDL OR-gate model (with syntax error).

## Before You Compile

Before you compile the VHDL code, the FPGA device type and pin numbers need to be assigned with **Assignments ⇒ Device** and **Assignments ⇒ Pin**. If your pins are already defined from the earlier Schematic Entry Tutorial, just confirm the pin assignments. If you did not do this step earlier in the tutorial see the device and pin assignment instructions at the end of section 1.1. At this point, VHDL code is generally ready to be compiled, simulated, and downloaded to the board using steps identical to those used earlier in the schematic entry method. Once pin assignments are made, they are stored in the project's \*.qsf file.

### 1.10 Compiling the VHDL Design

The Compile process checks for syntax errors, synthesizes the logic design, produces timing information for simulation, fits the design on the FPGA, and generates the file required to program the FPGA. After any changes are made to the design files or pin assignments, the project should always be re-compiled prior to simulation or programming.

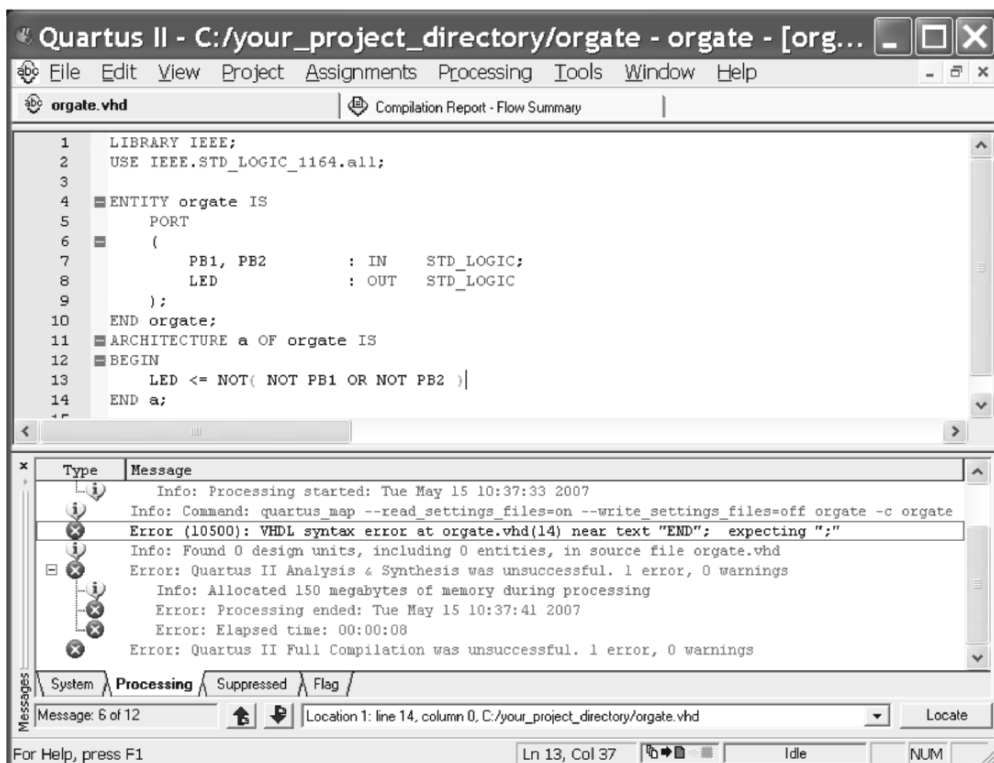


Figure 1.22 VHDL compilation with a syntax error.

Select **Project ⇒ ADD/Remove Files in Current Project**. Confirm that the new orgate.vhd file is now part of project and **remove** the tutorial's earlier



**orgate.bdf** file that the new VHDL file replaces from the project's file list, if it is present. Click **OK**. Start the compiler with **Processing ⇒ Start Compilation**.

### Checking for Compile Warnings and Errors

The project should compile with an error. After compiling the VHDL code, a window indicating an error should appear. The result should look something like Figure 1.22.

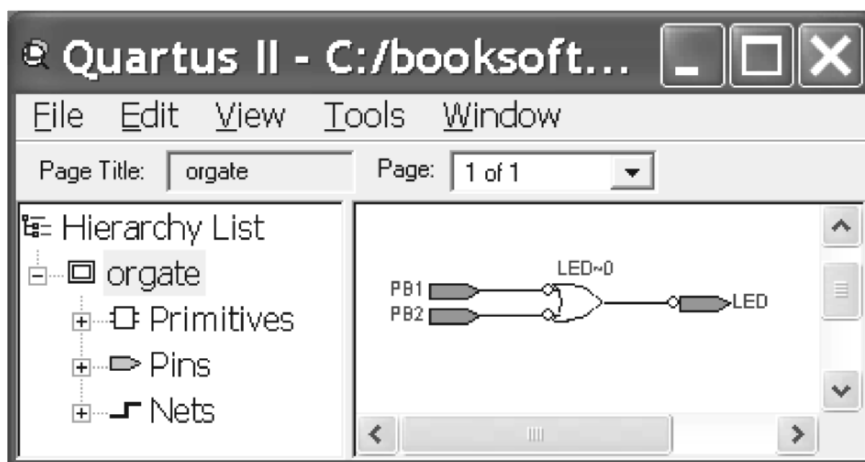
Double click on the first red error line and note that the cursor is placed in the editor either on or after the line missing the semicolon (;). VHDL statements should end with a semicolon. Add the semicolon to the end of the line so that it is now reads:

```
LED <= NOT ( NOT PB1 OR NOT PB2 );
```

Now, recompile, and you should have no errors. You can simulate your VHDL code using steps identical to the tutorial's earlier schematic version of the project.

### Viewing the Synthesized Logic in a Schematic

You can view the logic automatically generated by the VHDL synthesis tools using **Tools ⇒ Netlist Viewers ⇒ RTL Schematic**, after a successful compilation. A schematic of the logic synthesized by your VHDL code will be displayed as seen in Figure 1.23. This is a handy way to double check simple logic designs when first becoming familiar with VHDL or Verilog hardware description languages and their associated automatic logic synthesis tools.



**Figure 1.23** Logic schematic that was automatically synthesized from the VHDL code.

### 1.11 The 10 Minute Verilog Entry Tutorial

Verilog is another widely used hardware description language (HDL). Verilog and VHDL have roughly the same capabilities. VHDL is based on a PASCAL style syntax and Verilog is based on the C language. In large designs, HDLs greatly increase productivity and reduce design cycle time. Logic minimization and synthesis are automatically performed by the compiler and synthesis tools. Just like the previous VHDL example, to perform addition, the Verilog statement:

$$A = B + C;$$

will automatically generate an addition logic circuit with the correct number of bits to generate the new value of A.

Using the OR-gate design from the Schematic Entry Tutorial and the VHDL Tutorial, we will now create the same circuit in Verilog.

PRIOR KNOWLEDGE OF VERILOG IS NOT NEEDED TO COMPLETE THIS TUTORIAL.

#### Using a Template to Begin the Entry Process

Choose **File** ⇒ **New**, select **Verilog HDL File** and **OK**. Place the cursor within the text area, right click the mouse, and select **Insert Template** and then select **Verilog HDL**. (Note the different prewritten templates. These are built to expedite the entry of Verilog.) Select **Constructs** ⇒ **Design Units** ⇒ **Module Declaration (style 2)** – this declaration is the one you will generally start with since it also sets up the input and output declarations. Click **Insert** then **Close** and the template for the module declaration appears in the Text editor. Since the editor knows that it is a Verilog source file, the text will appear in different context-sensitive colors. Verilog keywords appear in blue and strings in purple and comments in green. The color information should be used to detect minor syntax errors while still in the text editor.

#### Saving the new Verilog File

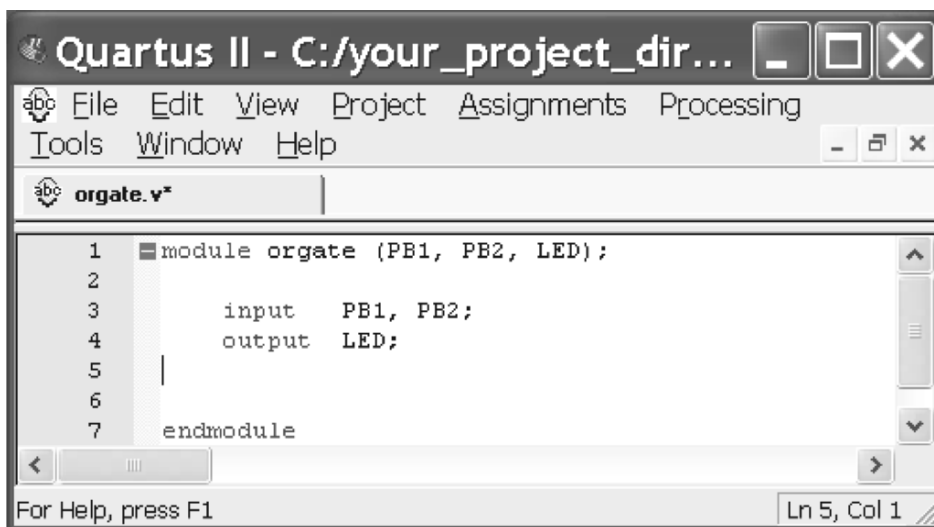
Select **File** ⇒ **Save As**. Note the automatic extension is **.v** (Verilog) and save the file as **orgate.v** – click **Save**.

#### Replacing Comments in the Verilog Code

The entire string indicating the position of the entity name, **<module\_name>**, should be set to the name used for the filename – in this case, **orgate**. There is one occurrence of **<module\_name>** in the text. Find and change it accordingly. Lines starting with **//** are comments and these will need to be replaced with the appropriate Verilog code.

## Declaring the I/O Pins

The input pins, PB1 and PB2, and the output pin, LED, need to be specified in the arguments of the Module statement and Port declaration. Since there are no 'inout pins', 'wire', 'reg', or 'integer' declarations, or Always statements in this design, remove all of these lines. The source file should now look similar to Figure 1.24.



**Figure 1.24** Verilog module declaration text.

## Setting up the Behavioral Code

Click the mouse to just before the line starting with "endmodule". (We will be inserting another template here.) Following the earlier procedure for selecting a Verilog template (start with a right click), select a **Constructs ⇒ Module Items ⇒ Continuous Assignment**. (A single assign statement will specify the internal logic of this design.) The syntax for a **Continuous Assignment Statement** appears in the text window after the other text. (You can now see why the template is left highlighted – had you not placed your cursor first, text would have appeared at your last cursor position. If you do misplace the template, hitting the delete key removes the highlighted text.)

## Editing the Continuous Assignment Statement

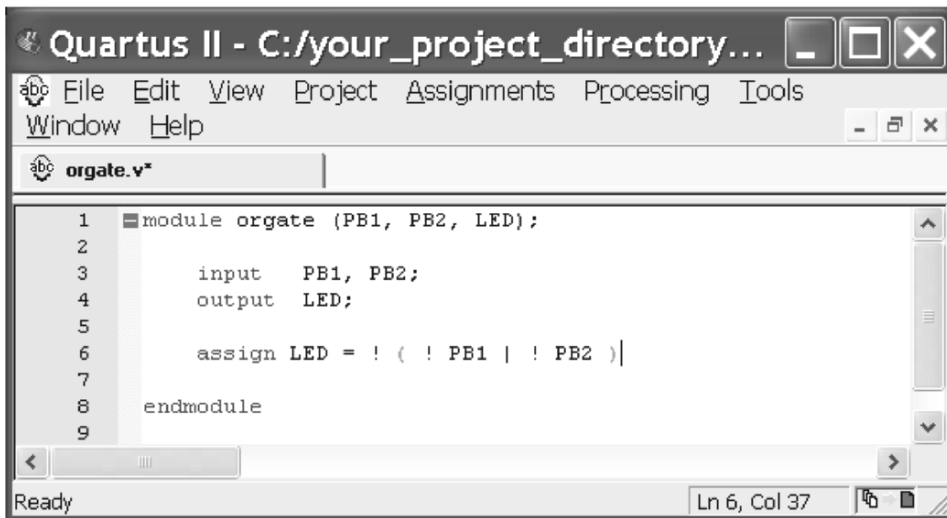
Change the <net lvalue> in the assign statement to "LED" and <value> to:

**! ( ! PB1 | ! PB2 );**

Verilog is based on C and "|" (vertical line) is the bit wise OR operator. The "!" (exclamation point) is the NOT operator. Delete the remaining comment lines that start with "/\*". Delete the ";" at the end of the assign LED statement (This causes a deliberate syntax error that will be detected and fixed later.) The file should now appear as in Figure 1.25.

## Before You Compile

Before you compile the Verilog code, the FPGA device type and pin numbers need to be assigned with **Assignments ⇒ Device** and **Assignments ⇒ Pin**. If your pins are already defined from the earlier Schematic Entry Tutorial, just confirm the pin assignments. If you did not do this step earlier in the tutorial see the device and pin assignment instructions at the end of Section 1.1.



**Figure 1.25** Verilog active low OR-gate model (with syntax error).

At this point, Verilog code is generally ready to be compiled, simulated, and downloaded to the board using steps identical to those used earlier in the schematic entry method. Once pin assignments are made, they are stored in the project's \*.qsf file.

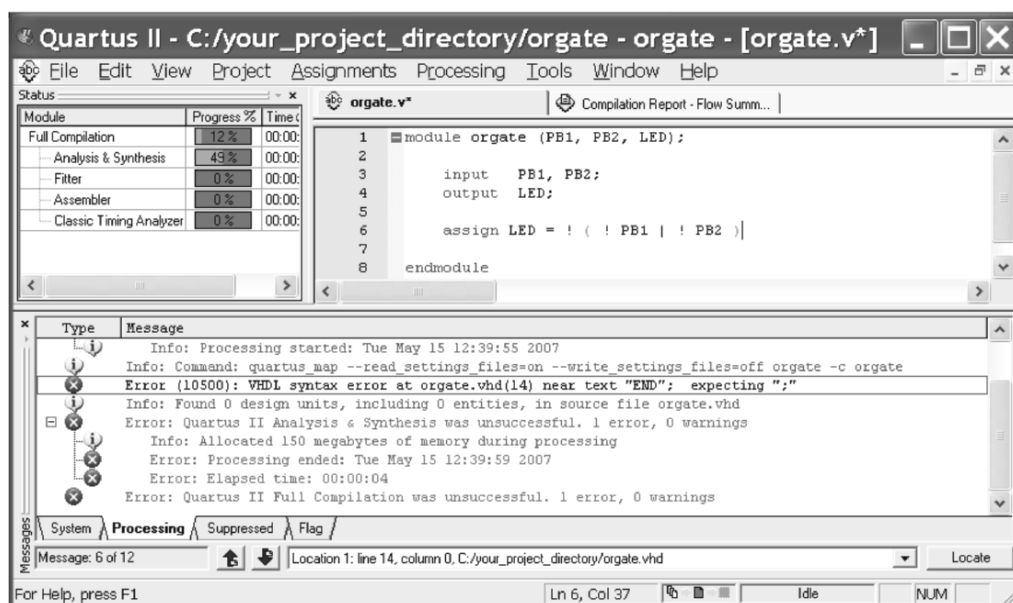
## 1.12 Compiling the Verilog Design

The Compile process checks for syntax errors, synthesizes the logic design, produces timing information for simulation, fits the design on the FPGA, and generates the file required to program the FPGA. After any changes are made to the design files or pin assignments, the project should always be re-compiled prior to simulation or programming.

Select **Project ⇒ ADD/Remove Files in Current Project**. Confirm that the new **orgate.v** file is now part of project and **remove** the tutorial's earlier **orgate.bdf** or **orgate.vhd** files that the new Verilog file replaces from the project if either file is present. Click **OK**. Start the compiler with **Processing ⇒ Start Compilation**.

## Checking for Compile Warnings and Errors

The project will compile with an error. After compiling the Verilog code, a window indicating an error should appear. (See Figure 1.26.)



**Figure 1.26** Verilog compilation with a syntax error.

Double click on the first red error line and note that the cursor is placed in the editor either on or after the line missing the semicolon (;). Verilog statements should end with a semicolon. Add the semicolon to the end of the line so that it is now reads:

```
assign LED = ! ( ! PB1 | ! PB2 );
```

Now, recompile, and you should have no errors. You can simulate your Verilog code using steps identical to the tutorial's earlier schematic version of the project. You can also view a schematic of the synthesized logic using **Tools ⇒ Netlist Viewers ⇒ RTL Schematic** similar to the steps previously outlined in the VHDL tutorial.

### 1.13 Timing Analysis

With every physical device, there are timing considerations. An FPGA's timing is affected by:

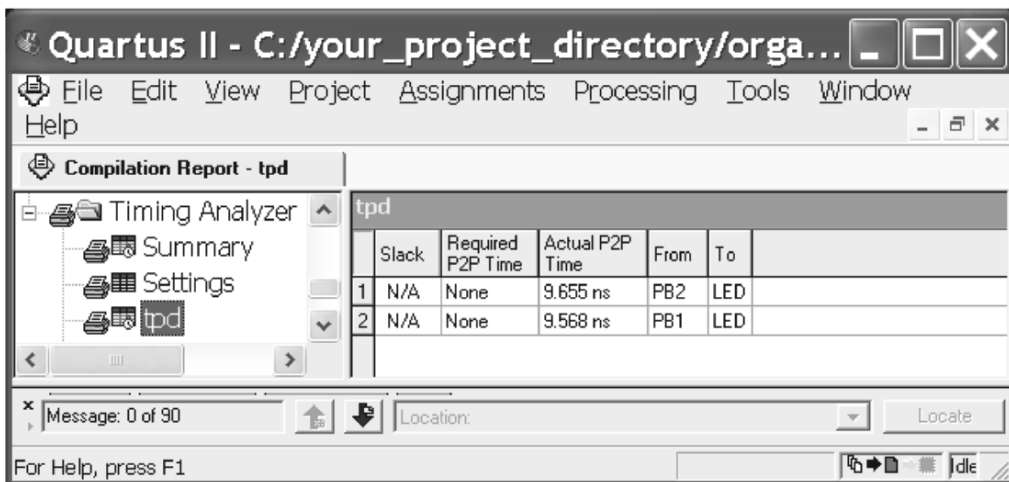
- Input buffer delays,
- Signal routing interconnect delays within the FPGA,
- The internal logic delays (in this case the OR), and
- Output buffer delays.

The timing analysis tool can be used to determine:

- The physical delay times and
- The maximum clock rates in your design.

#### Starting the Analyzer

At the top menu, select **Processing** ⇒ **Compilation Report** and then in the Compilation Report window, **expand Timing Analyzer** and **select tpd**. A matrix of input to output delay times for the project will be computed and displayed as seen in Figure 1.27.



**Figure 1.27** Timing analyzer showing input to output signal propagation timing delays.

Note that this is the same delay time seen in the simulator. These times include the input-to-output buffer delays at the pins and the interconnect delays inside the FPGA. The internal OR logic delay is only around a nanosecond relative to the rest of the device delay.

The exact time shown will vary with different versions of the Altera CAD tools, the different FPGA chips found on the various boards, and different FPGA chip speed grades. Other timing analysis options include setup times, hold times, and clock rates for sequential circuits.

### 1.14 The Floorplan Editor

A floorplan editor is a visual tool to assist expert users in manually placing and moving portions of logic circuits to different logic cells inside the FPGA. This is done in an attempt to achieve faster timing or better utilization of the FPGA.

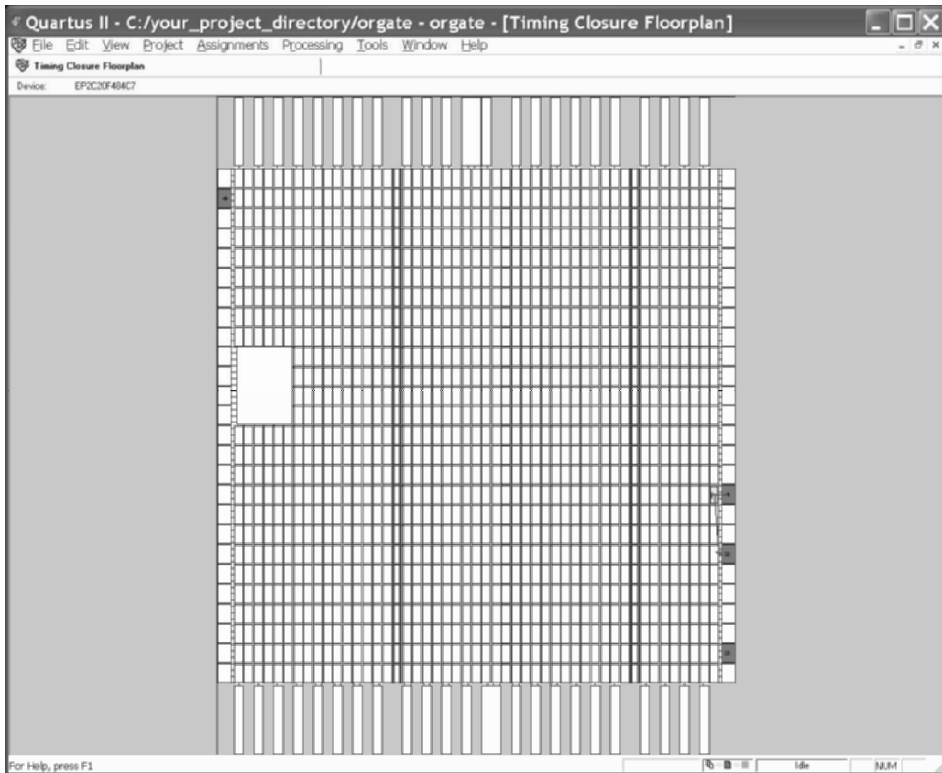
Floorplanning is typically used only on very large designs that contain subsections of hardware with critical high-speed timing. Since the interconnect delays are as large as the design's logic delays, logic element and I/O pin placement is very critical in high speed designs. Vertical and horizontal interconnect buses are used through the FPGA to connect Logic Elements.

For all but expert users, the compiler's automatic place-and-route tools should be used. Automatic place-and-route was already performed by the fitter in the compile process of the tutorial. Timing constraints for critical signals can also be specified in some FPGA place and routing tools to help the fitter meet the design's timing goals.

To see the fitter's automatic placement of the design inside the FPGA, select **Assignment ⇒ Back-Annotate Assignments** click **OK** and then **Assignments ⇒ Timing Closure Floorplan**. In the display that opens, zoom in and scroll around to find the colored logic element and gray shaded I/O pins used in your design. Find and select the colored Logic Element (LE), then **View ⇒ Routing ⇒ show fan in** and then **show fan out**, and a view like Figure 1.28 showing the design can be produced for the DE1.

There is a lot of empty space since the Cyclone II EP2C20 FPGA contains 18,752 Logic Elements (LEs) and 315 I/O pins. Only 1 LE and three I/O pins were used in this design. If you move the logic cell or I/O pins to other locations, it will make small changes to the circuit timing because of changes in the interconnect delays inside the FPGA.

Due to the vast number of possible combinations, FPGA CAD tools cannot explore every possible placement and routing option. The Quartus II Design Space Explorer tool can also be used to search and explore other design options in the design space. Large FPGA designs containing millions of gates can require several hours or even days of CPU time to examine many of the different place and route alternatives in the design space.



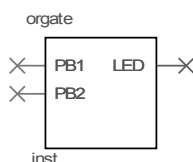
**Figure 1.28** Floorplan view showing the internal FPGA I/O and LE placements of an OR-gate on a DE1. User logic and pins used in the design are seen as shaded areas in the lower right corner.

### 1.15 Symbols and Hierarchy

The Symbol Editor is used to edit or create a symbol to represent a logic circuit. Symbols can be created for a design whenever a VHDL or Verilog file is compiled. Create a symbol for your VHDL design by opening the `orgate.vhd` file, and then select **File ⇒ Create/Update ⇒ Create Symbol Files for Current File**.

Select **File ⇒ Open**, change the file type setting for `*.bsf`, find and chose **orgate.bsf** to see the new symbol for your VHDL based design as shown in Figure 1.29. Inputs are typically shown on the left side of the symbol and outputs on the right side. Symbols are used for design hierarchy in more complex schematics. This new symbol can be used to add the circuit to a design with the graphic editor just like the BNOR2 symbol that was used earlier in the tutorial. Double clicking on a symbol in the graphic editor will open a pop-up requesting, “Select one design file.” Selecting the appropriate `*.vhd`, `*.v`, or `*.bdf` file takes the user to the underlying Verilog, VHDL, or graphic file respectively by opening the underlying file in a different screen tab.





**Figure 1.29** ORgate design symbol.

## 1.16 Functional Simulation

In large designs with long compile and simulation times, a faster *functional* simulation is commonly used initially. This type of simulation does not include device delay times and it is used solely to check for logic errors. Although a functional simulation is good for finding logic errors, a timing simulation is still necessary to check for any timing related errors as illustrated earlier in the tutorial.

### Performing a Functional Simulation

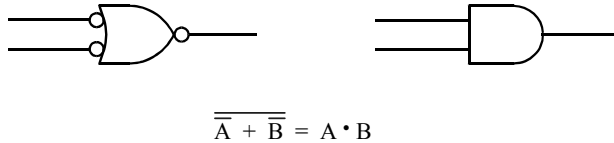
To perform a functional simulation, set the simulator for functional simulation with **Assignments ⇒ Settings**. Select **Simulator Settings** in the left column and then change the simulation mode from Timing to **Functional**. Run **Processing ⇒ Generate Functional Simulation Netlist**. Finally, select **Processing ⇒ Start Simulation**. Open the Simulation Report waveform and note that the output changes without any delay in response to an input, unlike the earlier timing simulation. To switch back to a timing-mode simulation, change the simulator setting back to timing, recompile, and restart the simulation.

This short tutorial has gone through the basics of a simple design using a common path through the design tools. As you continue to work with the tools, you will want to explore more of the menus, options and shortcuts. Chapter 4 contains a tutorial that will introduce a more complex design example. In Quartus II, **Help ⇒ Tutorial** also contains more tutorials. Quartus II video tutorials and reference manuals are also available online at Altera's website, [www.altera.com](http://www.altera.com).

A number of files such as the \*.q\* files are maintained in the project directory to support a design. Appendix B contains a list of different file extensions used by Quartus II. One of the more important files in a project is the \*.qsf file. It contains the device type and pin assignments along with a number of other project settings.

## 1.17 Laboratory Exercises

1. The tutorials ORed the active low signals from the pushbuttons and produced an output that was required to be low to turn off an LED. This was accomplished with the "negative-logic OR" gate illustrated to the left in Figure 1.30.



**Figure 1.30** Equivalent gates: A negative logic OR and a positive-logic AND.

We know from DeMorgan's Law that the equation in Figure 1.25 represents an equivalence. We should therefore be able to substitute a simple two-input AND gate as illustrated in Figure 1.29 and accomplish the same task as the single gate used in the tutorial. Substitute the AND2 gate for the BNOR2 gate in the schematic capture, then compile, simulate, and download the AND circuit. What can you conclude?

2. Substitute in the VHDL code:

```
LED <= PB1 AND PB2;
```

Or into the Verilog code:

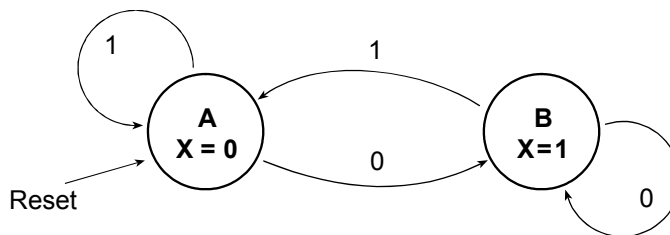
```
assign LED = PB1 & PB2;
```

Compile, simulate, and download and test the new circuit. What can you conclude about gate equivalence using DeMorgan's Theorem?

3. Design a logic circuit to turn on the LED when both pushbuttons are pressed. Compile, simulate, and download the new circuit.
4. Try a different logic function such as XOR. Start at the beginning or edit your existing schematic by deleting and replacing the BNOR2 symbol. Next repeat the tutorial steps to compile, simulate, download and test.
5. Repeat problem 2 for all of the basic gates including, OR, NOR, NAND, XOR, XNOR, and NOT. Try using different LEDs and output your results simultaneously. Look up the pin connections for the FPGA chip in Table 2.4 and be sure to give each pinout a different name.
6. Design, enter, simulate and implement a more complex logic gate network. One suggestion is a half adder. You will need two LED outputs.

- 
7. In the schematic editor, try building the design with some 74xx TTL parts from the others maxplus2 symbol library.
  8. Draw a schematic and develop a simulation to test the 2-to-1 Mux function in the others maxplus2 symbol library.
  9. View the orgate.rpt file and find the device utilization, the pin assignments, and the netlist. A substantial portion of the time delay in this simple logic design is the input and output buffer delays and the internal routing of this signal inside the FPGA. Find this delay time by removing the BNOR2 gate and one of the inputs in the schematic. Connect the input pin to the output pin, recompile and rerun the timing analyzer to estimate this time delay.
  10. Use the chip editor to move the logic cell used in the OR-gate design to another location inside the FPGA. For information of the chip editor, use the Quartus II Help function. Try moving the LE used several columns farther away from the pushbutton and LED pins. Not all locations of the logic cell will work and some trial and error will be required. Save the edited design, rerun the timing analyzer, and compare the resulting time delays with the original time delays. See if you are able to achieve a faster implementation than the automatic place-and-route tools.
  11. Remove the pin number constraints from the schematic and let the compiler assign the pin locations. Rerun the timing analyzer and compare the time delays. Are they faster or slower than having specified the input pins?
  12. If you are using a UP2 board, retarget the example design to the MAX chip. Pin numbers for the MAX decimal point LED can be found in the UP2 User manual. It will be necessary to connect jumper wires from the MAX header to the pushbuttons. Select pins near the pushbuttons. Pin numbers can be seen on the board's silk-screen. Compare the timing from the MAX implementation to the Flex implementation.
  13. If a storage oscilloscope or a fast logic analyzer is available, compare the predicted delay times from the simulation and timing analysis to the actual delays measured on the FPGA board. Force the pins to a header connector so that you can attach probes to the signal wires.
  14. Draw a schematic that uses the LPM\_ADD\_SUB megafunction to add two signed numbers on the Cyclone device. Use **Tools** ⇨ **Megawizard** to start the megawizard to help configure LPM symbols. Verify the proper operation using a simulation with two 4-bit numbers. Do not use pipelining, clock, or carry in. Vary the number of bits in the adder and find the maximum delay time using the timing analyzer. Plot delay time versus number of bits for adder sizes of 4, 8, 16, 32, and 64 bits. Using the LC percentages listed in the compiler's report file, estimate the hardware size in LEs. Plot LEs required versus number of bits.

15. Use the DFF part from the primitives storage library and enter the symbol in a schematic using the graphical editor. Develop a simulation that exercises all of the features of the D flip-flop. Use Help on DFF for more information on this primitive.
16. Use the DFFE part from the primitives storage library and enter the symbol in a schematic using the graphical editor. Develop a simulation that exercises all of the features of the D flip-flop with a clock enable. Use Help on DFFE for more information on this primitive.
17. Use gates and a DFF part from the primitives storage library with graphical entry to implement the state machine shown in the following state diagram. Verify correct operation with a simulation using the Altera CAD tools. The simulation should exercise all arcs in the state diagram. A and B are the two states, X is the output and Y is the input. Use the timing analyzer's **Processing** ⇌ **Classic Timing Analyzer Tool** ⇌ **Registered performance** option tab to determine the maximum clock frequency on the Cyclone device. Reset is asynchronous and the DFF Q output should be high for state B.



18. Repeat the previous problem but use one-hot encoding on the state machine. For one-hot encoding use two flip-flops with only one active for each state. For state A the flip-flop outputs would be "10" and for state B "01". One-hot encoding is common in FPGAs.