My First FPGA Design Tutorial
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How to Contact Altera</td>
<td>v</td>
</tr>
<tr>
<td>Typographic Conventions</td>
<td>v</td>
</tr>
<tr>
<td>Introduction</td>
<td>1–1</td>
</tr>
<tr>
<td>Design Flow</td>
<td>1–1</td>
</tr>
<tr>
<td>Before You Begin</td>
<td>1–2</td>
</tr>
<tr>
<td>What You Will Learn</td>
<td>1–2</td>
</tr>
<tr>
<td>Get Started</td>
<td>1–3</td>
</tr>
<tr>
<td>Assign the Device</td>
<td>1–6</td>
</tr>
<tr>
<td>Design Entry</td>
<td>1–7</td>
</tr>
<tr>
<td>Add a PLL Megafuction</td>
<td>1–14</td>
</tr>
<tr>
<td>Add a Multiplexer</td>
<td>1–28</td>
</tr>
<tr>
<td>Assign the Pins</td>
<td>1–35</td>
</tr>
<tr>
<td>Create a Default TimeQuest SDC File</td>
<td>1–38</td>
</tr>
<tr>
<td>Compile Your Project</td>
<td>39</td>
</tr>
<tr>
<td>Program the Device</td>
<td>1–41</td>
</tr>
<tr>
<td>Verify in Hardware</td>
<td>1–45</td>
</tr>
<tr>
<td>Next Steps</td>
<td>1–45</td>
</tr>
</tbody>
</table>
About this Tutorial

This tutorial provides comprehensive information that will help you understand how to create an Altera® FPGA design and run it on your development board.

How to Contact Altera

For the most up-to-date information about Altera products, refer to the following table.

<table>
<thead>
<tr>
<th>Information Type</th>
<th>Contact (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical support</td>
<td><a href="http://www.altera.com/mysupport/">www.altera.com/mysupport/</a></td>
</tr>
</tbody>
</table>
| Technical training                | www.altera.com/training/
custrain@altera.com                |
| Product literature                | www.altera.com/literature/ |
| Altera literature services        | literature@altera.com   |
| FTP site                          | ftp.altera.com          |

Note to table:
(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown below.

<table>
<thead>
<tr>
<th>Visual Cue</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold Type with Initial Capital Letters</td>
<td>Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: <strong>Save As</strong> dialog box.</td>
</tr>
<tr>
<td><strong>bold type</strong></td>
<td>External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $f_{MAX}$, \texttt{\textbackslash qdesigns} directory, \texttt{d:} drive, \texttt{chiptrip.gdf} file.</td>
</tr>
<tr>
<td><em>Italic Type with Initial Capital Letters</em></td>
<td>Document titles are shown in italic type with initial capital letters. Example: <em>AN 75: High-Speed Board Design.</em></td>
</tr>
<tr>
<td><em>Italic type</em></td>
<td>Internal timing parameters and variables are shown in italic type. Examples: $t_{PIA}$, $n + 1$.</td>
</tr>
<tr>
<td></td>
<td>Variable names are enclosed in angle brackets (&lt; &gt;) and shown in italic type. Example: &lt;file name&gt;, &lt;project name&gt;.pot file.</td>
</tr>
<tr>
<td>Visual Cue</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>Initial Capital Letters</td>
<td>Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.</td>
</tr>
<tr>
<td>“Subheading Title”</td>
<td>References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: “Typographic Conventions.”</td>
</tr>
<tr>
<td>Courier type</td>
<td>Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. Active-low signals are denoted by suffix n, e.g., resetn.</td>
</tr>
<tr>
<td></td>
<td>Anything that must be typed exactly as it appears is shown in Courier type. For example: c:\qdesigns\tutorial\chiptrip.gdf. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword SUBDESIGN), as well as logic function names (e.g., TRI) are shown in Courier.</td>
</tr>
<tr>
<td>1., 2., 3., and a., b., c., etc.</td>
<td>Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.</td>
</tr>
<tr>
<td>■ ● □</td>
<td>Bullets are used in a list of items when the sequence of the items is not important.</td>
</tr>
<tr>
<td>✓</td>
<td>The checkmark indicates a procedure that consists of one step only.</td>
</tr>
<tr>
<td>☰</td>
<td>The hand points to information that requires special attention.</td>
</tr>
<tr>
<td>☳ CAUTION</td>
<td>The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.</td>
</tr>
<tr>
<td>☳ WARNING</td>
<td>The warning indicates information that should be read prior to starting or continuing the procedure or processes.</td>
</tr>
<tr>
<td>←</td>
<td>The angled arrow indicates you should press the Enter key.</td>
</tr>
<tr>
<td>↑</td>
<td>The feet direct you to more information on a particular topic.</td>
</tr>
</tbody>
</table>
1. My First FPGA Design

Welcome to Altera and the world of programmable logic! This tutorial will teach you how to create a simple FPGA design and run it on your development board. The tutorial takes less than an hour to complete. The following sections provide a quick overview of the design flow, explain what you need to get started, and describe what you will learn.

Design Flow

The standard FPGA design flow starts with design entry using schematics or a hardware description language (HDL), such as Verilog HDL or VHDL. In this step, you create the digital circuit that is implemented inside the FPGA. The flow then proceeds through compilation, simulation, programming, and verification in the FPGA hardware (see Figure 1–1).

Figure 1–1. Design Flow

This tutorial guides you through all of the steps except for simulation. Although it is not covered in this document, simulation is very important to learn, and there are entire applications devoted to simulating hardware designs. There are two types of simulation, RTL and timing. RTL (or functional) simulation allows you to verify that your code is manipulating the inputs and outputs appropriately. Timing (or post place-and-route) simulation verifies that the design meets timing and functions appropriately in the device.

See “Next Steps” on page 1–45 for links to further information about simulation.
Before You Begin

This tutorial assumes the following prerequisites:

- You generally know what an FPGA is. This tutorial does not explain the basic concepts of programmable logic.
- You are somewhat familiar with digital circuit design and electronic design automation (EDA) tools.
- You have installed the Altera® Quartus® II software on your computer. If you do not have the Quartus II software, you can download it from the Altera web site at www.altera.com/download.
- You have an Altera Cyclone® III, Stratix® III, or Arria™ GX Development Board (or equivalent) on which you will test your project. Using a development board helps you to verify whether your design is really working.
- You have gone through the quick start guide and/or the getting started user guide for your development kit. These documents ensure that you have:
  - Installed the required software.
  - Determined that the development board functions properly and is connected to your computer.
  - Installed the USB-Blaster™ driver, which allows you to program the FPGA on the development board with your own design.

What You Will Learn

In this tutorial, you will perform the following tasks:

- **Create a design that causes LEDs on the development board to blink at a speed that is controlled by an input button**—This design is easy to create and gives you visual feedback that the design works. Of course, you can use your development board to run other designs as well. For the LED design, you will write Verilog HDL code for a simple 32-bit counter, add a phase-locked loop (PLL) megafunction as the clock source, and add a 2-input multiplexer megafunction. When the design is running on the board, you can press an input switch to multiplex the counter bits that drive the output LEDs.
Get Started

You begin this tutorial by creating a new Quartus II project. A project is a set of files that maintain information about your FPGA design. The Quartus II Settings File (.qsf) and Quartus II Project File (.qpf) files are the primary files in a Quartus II project. To compile a design or make pin assignments, you must first create a project.

1. In the Quartus II software, select File > New Project Wizard. The Introduction page opens. See Figure 1–2.
2. Click Next.

3. Enter the following information about your project:

   a. **What is the working directory for this project?** Enter a directory in which you will store your Quartus II project files for this design, for example, `c:\altera\my_first_fpga`.  

      File names, project names, and directories in the Quartus II software cannot contain spaces.

   b. **What is the name of this project?** Type `my_first_fpga`.

   c. **What is the name of the top-level design entity for this project?** Type `my_first_fpga_top`. See Figure 1–3.
Figure 1–3. Project Information

The wizard has several other pages after this one; however, for this tutorial you do not need to make changes to these pages. For more information on the options available in these pages, refer to the Quartus II Handbook.

4. When prompted, choose Yes to create the my_first_fpga project directory.

Congratulations! You just created your first Quartus II FPGA project. See Figure 1–4.
Assign the Device

In this section, you will assign a specific FPGA device to the design and make pin assignments. To assign the device, perform the following steps.

1. Choose Assignments > Device.

2. Under Family, choose the device family that corresponds to the device on the development board you are using.

3. Under Available devices, choose the device given in Table 1–1.

Table 1–1. Available Device Settings

<table>
<thead>
<tr>
<th>Development Board</th>
<th>Device Family</th>
<th>Package</th>
<th>Pin Count</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arria GX Development Board</td>
<td>Arria GX</td>
<td>FBGA</td>
<td>780</td>
<td>EP1AGX60DF780C6</td>
</tr>
<tr>
<td>Stratix III Development Board</td>
<td>Stratix III</td>
<td>FBGA</td>
<td>1,152</td>
<td>EP3SL150F1152C3</td>
</tr>
<tr>
<td>Cyclone III Starter Board</td>
<td>Cyclone III</td>
<td>FBGA</td>
<td>324</td>
<td>EP3C25F324C8</td>
</tr>
<tr>
<td>Cyclone III Development Board</td>
<td>Cyclone III</td>
<td>FBGA</td>
<td>780</td>
<td>EP3C120F780C7</td>
</tr>
</tbody>
</table>
4. Click OK.

**Design Entry**

In the design entry phase, you use RTL or schematic entry to create the logic to be implemented in the device. You also make pin assignments, including pin placement information, and timing constraints that might be necessary for building a functioning design.

In the design entry step you create a schematic or Block Design File (.bdf) that is the top-level design. You will add library of parameterized modules (LPM) functions and use Verilog HDL code to add a logic block. When creating your own designs, you can choose any of these methods or a combination of them.

1. Choose File > New > Block Diagram/Schematic File (see Figure 1–6) to create a new file, Block1.bdf, which you will save as the top-level design.

2. Click OK.
3. Choose File > Save As and enter the following information (see Figure 1-7).

- **File name**: my_first_fpga_top
- **Save as type**: Block Diagram/Schematic File (*.bdf)
4. Click **Save**. The new design file appears in the Block Editor (see **Figure 1–8**).
5. Add HDL code to the blank block diagram by choosing File > New > Verilog HDL File.

6. Click OK to create a new file Verilog1.v, which you will save as simple_counter.v.

7. Select File > Save As and enter the following information (see Figure 1–9).
   - File name: simple_counter.v
   - Save as type: Verilog HDL File (*.v, *.vlg, *.verilog)
The resulting empty file is ready for you to enter the Verilog HDL code.

8. Type the following Verilog HDL code into the blank `simple_counter.v` file (see Figure 1–10).

```
// This is an example of a simple 32 bit up-counter called simple_counter.v
// It has a single clock input and a 32-bit output port
module simple_counter (input clock , output reg [31:0] counter_out);
always @ (posedge clock)// on positive clock edge
begin
  counter_out <= #1 counter_out + 1;// increment counter_out + 1;
end
endmodule// end of module counter
```
9. Save the file by choosing **File > Save**, pressing Ctrl + s, or by clicking the floppy disk icon.

10. Choose **File > Create/Update > Create Symbol Files for Current File** to convert the `simple_counter.v` file to a Symbol File (.sym). You use this Symbol File to add the HDL code to your BDF schematic.

   The Quartus II software creates a Symbol File and displays a message (see **Figure 1–11**).

**Figure 1–11. Symbol File Created**

11. Click **OK**.

12. To add the `simple_counter.v` symbol to the top-level design, click the **my_first_fpga_top.bdf** tab.
13. Choose **Edit > Insert Symbol**.

14. Double-click the **Project** directory to expand it.

15. Select the newly created **simple_counter symbol** by clicking its icon.
   
   ![Symbol dialog box](image)
   
   You can also double-click in a blank area of the BDF to open the **Symbol** dialog box

**Figure 1–12. Adding the Symbol to the BDF**

16. Click **OK**.

17. Move the cursor to the BDF grid; the symbol image moves with the cursor. Click to place the **simple_counter symbol** onto the BDF. You can move the block after placing it by simply clicking and dragging it to where you want it and releasing the mouse button to place it. See **Figure 1–13**.
18. Press the Esc key or click an empty place on the schematic grid to cancel placing further instances of this symbol.

Save your project regularly.

**Add a PLL Megafuction**

Megafuctions, such as the ones available in the LPM, are pre-designed modules that you can use in FPGA designs. These Altera-provided megafuctions are optimized for speed, area, and device family. You can increase efficiency by using a megafuction instead of writing the function yourself. Altera also provides more complex functions, called MegaCore® functions, which you can evaluate for free but require a license file for use in production designs.

This tutorial design uses a PLL clock source to drive a simple counter. A PLL uses the on-board oscillator (which is different for different development boards) to create a constant clock frequency as the input to the counter. To create the clock source, you will add a pre-built LPM megafunction named ALTPLL.

1. Choose **Edit > Insert Symbol** or click Add Symbol on the toolbar ( ).
2. Click Megawizard Plug-in Manager. The MegaWizard® Plug-In Manager appears (see Figure 1–14).

Figure 1–14. MegaWizard Plug-In Manager

3. Click Next.

4. In MegaWizard Plug-In Manager [page 2a], specify the following selections (see Figure 1–15):
   
   a. Choose I/O > ALTPLL.

   b. Under Which device family will you be using?, choose the device family the corresponds to the device on your development kit board.

   c. Under Which type of output file do you want to create?, choose Verilog HDL.

   d. Under What name do you want for the output file?, type \texttt{pll} at the end of the already created directory name.

   e. Click Next.
5. In the MegaWizard Plug-In Manager [page 3 of 14] window, make the following selections (see Figure 1–16).

   a. Confirm that the **Currently selected device family** option shows the device family that corresponds to the development board you are using.

   b. Under **Which device speed grade will you be using?**, type the value given in Table 1–2.

<table>
<thead>
<tr>
<th>Table 1–2. Speed Grade Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Development Board</strong></td>
</tr>
<tr>
<td>Arria GX Development Board</td>
</tr>
<tr>
<td>Stratix III Development Board</td>
</tr>
<tr>
<td>Cyclone III Starter Board</td>
</tr>
<tr>
<td>Cyclone III Development Board</td>
</tr>
</tbody>
</table>
c. Under **What is the frequency of the inclock0 input?**, type the value given in Table 1–3.

<table>
<thead>
<tr>
<th>Development Board</th>
<th>Setting (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arria GX Development Board</td>
<td>100</td>
</tr>
<tr>
<td>Stratix III Development Board</td>
<td>50</td>
</tr>
<tr>
<td>Cyclone III Starter Board</td>
<td>50</td>
</tr>
<tr>
<td>Cyclone III Development Board</td>
<td>50</td>
</tr>
</tbody>
</table>

d. Ensure that the units are **MHz** (default).

e. Click **Next**.
6. Turn off all options on MegaWizard page 4. As you turn them off, pins disappear from the PLL block’s graphical preview. See Figure 1–17 for an example.
7. Click Next.

8. At the top of the wizard, click the tab 3. **Output Clocks** to jump to the **Output Clocks > clk c0 page.**
a. Under **Clock division factor**, use the up/down arrows or enter the value given in **Table 1–4**.

<table>
<thead>
<tr>
<th>Development Board</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arria GX Development Board</td>
<td>20</td>
</tr>
<tr>
<td>Stratix III Development Board</td>
<td>10</td>
</tr>
<tr>
<td>Cyclone III Starter Board</td>
<td>10</td>
</tr>
<tr>
<td>Cyclone III Development Board</td>
<td>10</td>
</tr>
</tbody>
</table>

See **Figure 1–18** for an example that uses 10 as the division factor.
9. Click Finish.

10. The wizard displays a summary of the files it creates (see Figure 1–19). Click Finish again.
The Symbol window opens, showing the newly created PLL megafunction. See Figure 1–20.
11. Click **OK** and place the `pll` symbol onto the BDF to the left of the `simple_counter` symbol. You can move the symbols around by holding down the left mouse button, helping you ensure that they line up properly. See Figure 1–21.
12. Move the mouse so that the cursor (also called the selection tool) is over the pll symbol’s c0 output pin. The orthogonal node tool (cross-hair) icon appears.

13. Click and drag a bus line from the c0 output to the simple_counter clock input. This action ties the pll output to the simple_counter input (see Figure 1–22).
14. Add an input pin and an output bus with the following steps:

   a. Choose **Edit > Insert Symbol**.

   b. Under **Libraries**, select **quartus/libraries > primitives > pin > input**. See **Figure 1–23**.

   c. Click **OK**.

   If you need more room to place symbols, you can use the vertical and horizontal scroll bars at the edges of the BDF window to view more drawing space.

   ![Figure 1–23. Input Pin Symbol](image)

   d. Place the new pin onto the BDF so that it is touching the input to the **pll** symbol.

   e. Use the mouse to click and drag the new input pin to the left; notice that the ports remain connected as shown in **Figure 1–24**.
f. Change the pin name by double-clicking `pin_name` and typing `osc_clk` (see Figure 1–25). This name correlates to the oscillator clock that is connected to the FPGA.

g. Using the Orthogonal Bus tool ( ), draw a bus line connected on one side to the `simple_counter` output port, and leave the other end unconnected at about 6 to 8 grid spaces to the right of the `simple_counter`.

h. Right-click the new output bus line and choose Properties.
i. Type `counter[31..0]` as the bus name (see Figure 1–26). The notation [X..Y] is the Quartus II method for specifying the bus width in BDF schematics, where X is the most significant bit (MSB) and Y is the least significant bit (LSB).

Figure 1–26. Change Output Bus Name

![Bus Properties dialog box]

j. Click OK. Figure 1–27 shows the BDF.
Add a Multiplexer

This design uses a multiplexer to route the `simple_counter` output to the LED pins on the development board. You will use the MegaWizard Plug-In Manager to add the multiplexer, `lpm_mux`. The design multiplexes two variations of the counter bus to four LEDs on the development board.

1. Choose `Edit > Insert Symbol`.
2. Click `Megawizard Plug-in Manager`.
3. Click `Next`.
5. Choose the device family that corresponds to the device on the development board you are using, choose `Verilog HDL` as the output file type, and name the output file `counter_bus_mux` (see Figure 1–28).
6. Click Next.

**Figure 1–28. Selecting lpm_mux**

![MegaWizard Plug-In Manager](image)

7. Under **How many ‘data’ inputs do you want?**, select 2 inputs (default).

8. Under **How ‘wide’ should the data input and result output be?**, select 4 (see Figure 1–29).
9. Click **Next**.

10. Click **Finish** twice. The Symbol window appears (see Figure 1–30 for an example).
11. Click OK.

12. Place the `counter_bus_mux` symbol below the existing symbols on the BDF. See Figure 1–31.
13. Add input buses and output pins to the `counter_bus_mux` symbol as follows:

   a. Using the Orthogonal Bus tool, draw bus lines from the `data1x[3..0]` and `data0x[3..0]` input ports to about 6 to 8 grid spaces to the left of `counter_bus_mux`.

   b. Draw a bus line from the `result[3..0]` output port to about 6 to 8 grid spaces to the right of `counter_bus_mux`.

   c. Right-click the bus line connected to `data1x[3..0]` and choose **Properties**.

   d. Name the bus `counter[26..23]`, which selects only those counter output bits to connect to the four bits of the `data1x` input.

Because the input busses to `counter_bus_mux` have the same names as the output bus from `simple_counter`, (`counter[x..y]`) the Quartus II software knows to connect these busses.
e. Click OK.

f. Right-click the bus line connected to data0x[3..0] and choose Properties.

g. Name the bus counter[24..21], which selects only those counter output bits to connect to the four bits of the data1x input.

h. Click OK. Figure 1–32 shows the renamed buses.

If you have not done so already, save your project file before continuing.


15. Under Libraries, double-click quartus/libraries/ > primitives > pin > output (see Figure 1–33).

16. Click OK.
17. Place this output pin so that it connects to the `counter_bus_mux` `result[3..0]` bus output line.

18. Rename the output pin as `led[3..0]` as described in steps 13 c and d. (see Figure 1–34).

**Figure 1–34. Rename the Output Pin**

19. Attach an input pin to the multiplexer select line using an input pin:
   a. Choose **Edit > Insert Symbol**.
   c. Click **OK**.

20. Place this input pin below `counter_bus_mux`.

21. Connect the input pin to the `counter_bus_mux sel` pin.

22. Rename the input pin as `button[0]` (see Figure 1–35).
You have finished adding symbols to your design.

You can add notes or information to the project as text using the Text tool on the toolbar (indicated with the A symbol). For example, you can add the label “OFF = SLOW, ON = FAST” to the button[0] input pin and add a project description, such as “My First FPGA Project.”

Assign the Pins

In this section, you will make pin assignments. Before making pin assignments, perform the following steps:


2. Click OK in the message window that appears after analysis and elaboration completes.
To make pin assignments that correlate to the button[0] and osc_clk input pins and led[3..0] output pin, perform the following steps:

1. Choose Assignments > Pins, which opens the Pin Planner, a spreadsheet-like table of specific pin assignments. The Pin Planner shows the design’s six pins. See Figure 1–36.

**Figure 1–36. Pin Planner Example**
2. In the **Location** column next to each of the six node names, add the coordinates (pin numbers) as shown in Table 1–5 for the actual values to use with your board.

### Table 1–5. Pin Information Settings

<table>
<thead>
<tr>
<th>Pin</th>
<th>Arria GX Development Board</th>
<th>Stratix III Development Board</th>
<th>Cyclone III Starter</th>
<th>Cyclone III Development Board</th>
</tr>
</thead>
<tbody>
<tr>
<td>button[0]</td>
<td>A19</td>
<td>B17</td>
<td>F1</td>
<td>AD7</td>
</tr>
<tr>
<td>led[2]</td>
<td>C13</td>
<td>B23</td>
<td>N12</td>
<td>AF18</td>
</tr>
<tr>
<td>led[0]</td>
<td>A12</td>
<td>F21</td>
<td>P13</td>
<td>AD15</td>
</tr>
<tr>
<td>osc_clk</td>
<td>U28</td>
<td>T33</td>
<td>V9</td>
<td>AH15</td>
</tr>
</tbody>
</table>

Double-click in the **Location** column for any of the six pins to open a drop-down list and type the location shown in the table. Alternatively, you can select the pin from a drop-down list. For example, if you type `F1` and press the Enter key, the Quartus II software fills in the full `PIN_F1` location name for you. The software also keeps track of corresponding FPGA data such as the I/O bank and Vref group. Each bank has a distinct color, which corresponds to the top-view wire bond drawing in the upper right window. See Figure 1–37.

> To provide more detailed information about the development boards, Altera provides board reference manuals and schematics for all development boards. These documents provide complete information about the pinouts for the devices on the boards.
Create a Default TimeQuest SDC File

Timing settings are critically important for a successful design. For this tutorial you will create a basic Synopsys Design Constraints File (.sdc) that the Quartus II TimeQuest Timing Analyzer uses during design compilation. For more complex designs, you will need to consider the timing requirements more carefully. To create an SDC, perform the following steps:

1. Open the TimeQuest Timing Analyzer by choosing **Tools > TimeQuest Timing Analyzer**.

2. Choose **File > New SDC file**. The SDC editor opens.

3. Type the following code into the editor:

```
create_clock -period 20.000 -name osc_clk osc_clk
derive_pll_clocks
derive_clock_uncertainty
```

4. Save this file as **my_first_fpga_top.sdc** (see Figure 1–38).

---

**Figure 1–37. Completed Pin Planning Example**

You are finished creating your Quartus II design!
Figure 1–38. Default SDC

Naming the SDC with the same name as the top-level file except for the .sdc extension causes the Quartus II software to use this timing analysis file automatically by default. If you used another name, you would need to add the SDC to the assignments file list.

For more advanced information about timing settings and the TimeQuest timing analyzer, refer to the Quartus II Verification and Simulation page on Altera’s website at www.altera.com/products/software/products/quartus2/verification/qts-verification.html.

Compile Your Project

After creating your design, you must compile it. Compilation converts the design into a bitstream that can be downloaded into the FPGA. The most important output of compilation is an SRAM Object File (.sof), which you use to program the device. The software also generates other report files that provide information about your code as it compiles.

If you want to store SOFs in memory devices (such as flash or EEPROMs), you must first convert the SOF to a file type specifically for the targeted memory device.

Now that you have created a complete Quartus II project and entered all assignments, you can compile the design.

In the Processing menu, choose Start Compilation or click the Play button on the toolbar ( ).

If you are asked to save changes to your BDF, click Yes.
While compiling your design, the Quartus II software provides useful information about the compilation (see Figure 1–39).

**Figure 1–39. Compilation Message Example**

![Compilation Message Example](image)

When compilation is complete, the Quartus II software displays a message. Click OK to close the message box.

The Quartus II Messages window displays many messages during compilation. It should not display any critical warnings; it may display a few warnings that indicate that the device timing information is preliminary or that some parameters on the I/O pins used for the LEDs were not set.

The software provides the compilation results in the **Compilation Report** tab as shown in Figure 1–40.
After compiling and verifying your design you are ready to program the FPGA on the development board. You download the SOF you just created into the FPGA using the USB-Blaster circuitry on the board.

Set up your hardware for programming using the following steps:

1. Connect the power supply cable to your board and to a power outlet.

2. For Cyclone III and Stratix III development boards, connect the USB cable to the board. For the Arria GX board, connect the USB-Blaster (included in your development kit) to J4 and the USB cable to the USB-Blaster. Connect the other end of the USB cable to the host computer.

   Refer to the getting started user guide for detailed instructions on how to connect the cables.

3. Turn the board on using the on/off switch (SW1).
For the Arria GX development board, before opening the device programmer, place the MAXII and HSMC slide switches in the bypass position. The default position (without an HSMC card installed) is HSMC in bypass mode and MAXII in chained mode.

Program the FPGA using the following steps.

1. Choose **Tools > Programmer**. The Programmer window opens. See Figure 1–41.

2. Click **Hardware Setup**.

3. If it is not already turned on, turn on the **USB-Blaster [USB-0]** option under **Currently selected hardware**. See Figure 1–42.
4. Click Close.

5. If the file name in the Programmer does not show my_first_fpga_top.sof, click Add File.

6. Select the my_first_fpga_top.sof file from the project directory (see Figure 1–43).
7. Click **Open**.

8. Turn on the **Program Configure** option that corresponds to the **my_first_fpga_top.sof** file.

9. Click **Start**. The file downloads to the development board.

   The progress bar shows the download status; the status is 100% when downloading completes. See Figure 1-44.

**Figure 1–44. Downloading Complete**

Congratulations, you have created, compiled, and programmed your first FPGA design! The compiled SRAM Object File (.sof) is loaded onto the FPGA on the development board and the design should be running.
Verify in Hardware

When you verify the design in hardware, you observe the runtime behavior of the FPGA hardware design and ensure that it is functioning appropriately.

Verify the design by performing the following steps:

1. Observe that the four development board LEDs appear to be advancing slowly in a binary count pattern, which is driven by the `simple_counter` bits [26..23].

   ❑ The LEDs are active low, therefore, when counting begins all LEDs are turned on (the 0000 state).

2. Press and hold Button 1 on the development board and observe that the LEDs advance more quickly. Pressing this button causes the design to multiplex using the faster advancing part of the counter (bits [24..21]).

Next Steps

Altera provides many tutorials and reference material that you can use to further your knowledge of FPGA design. The information on the following web pages will help you learn more about Altera tools and products.

- www.altera.com/education/univ/unv-index.html
- www.altera.com/literature/lit-qts.jsp
- www.altera.com/end-markets/refdesigns/ref-index.jsp
- www.altera.com/support/examples/exm-index.html
- www.altera.com/corporate/contact/con-index.html
- mysupport.altera.com/etraining/

The Altera etraining page provides tutorials for each of the steps covered in this document. Additionally, it provides simulation tutorials, such as:

- Using the Quartus II Software: Simulation
- Using the Quartus II Software: Timing Analysis
- Constraining and Analyzing Timing for Source Synchronous Circuits with TimeQuest
- Validating Performance with the TimeQuest Static Timing Analyzer