Lattice Radiant 2.2 Tutorial with CrossLink-NX (LIFCL)

October 20, 2020

Copyright

Trademarks

Disclaimers

Type Conventions Used in This Document

Convention	Meaning or Use
Bold	Items in the user interface that you select or click. Text that you type into the user interface.
<*Italic*>	Variables in commands, code syntax, and path names.
Ctrl+L	Press the two keys at the same time.
`Courier`	Code examples. Messages, reports, and prompts from the software.
`...`	Omitted material in a line of code.
. . .	Omitted lines in code and report examples.
`[ ]`	Optional items in syntax descriptions. In bus specifications, the brackets are required.
`( )`	Grouped items in syntax descriptions.
`{ }`	Repeatable items in syntax descriptions.
`|`	A choice between items in syntax descriptions.

Contents

Lattice Radiant 2.2 Tutorial with CrossLink-NX (LIFCL)

The Lattice Radiant® software is a complete toolset for designing for Lattice Semiconductor's FPGAs. This tutorial leads you through all the basic steps of designing, implementing, and debugging designs targeted to the Lattice CrossLink-NX™ (LIFCL) device family.

Note

Some of the screen captures in this tutorial may have been taken from a version of the Radiant software that differs from the one you are using. There may be slight differences in the graphical user interface (GUI), but the software functions the same.

About the Tutorial

When you have completed this tutorial, you should be able to do the following:

- Create a new Radiant software project.
- Customize IP using IP Catalog.
- Verify functionality with simulation.
- Set timing and location constraints.
- Process the design.
- Analyze power consumption.
- Analyze static timing.
- Create on-chip debug logic.

Time to Complete About 2 hours.

You can stop at the end of any task and restart at the beginning of the next task. See "Close the Radiant Project" on page 42. When you restart the

Radiant software, it shows a Recent Project List. Just click the name of your project.

System Requirements You need:

- Radiant software, version 2.2

Online Help You can find additional information on any tool used in the tutorial at any time by choosing **Help > Lattice Radiant Software Help** or **Help > <*tool name*>**. The Help also provides easy access to many other information sources.

Sample Design This tutorial comes with a Verilog design that counts up and down. There are also some additional modules so you can fully exercise the Radiant software's on-chip debugging abilities: a dual-port RAM module and a module that uses the MIPI D-PHY interface built into CrossLink-NX. The tutorial includes a simple testbench to run the simulator on. This is not intended to be a complete design, so some of the modules are not completely connected.

Task 1: Create a New Radiant Project

A "project" is a collection of all the files and settings needed to create your design, test and analyze its behavior, and process it into a programming file for a Lattice FPGA.

Setting up a new project is done through the New Project wizard. The New Project wizard guides you through the steps of specifying a project name and location, selecting a target device, and adding existing source files to the new project. We will walk through each page of the wizard one by one. At the end, we'll introduce the Radiant main window and its parts.

Opening the New Project Wizard

Open the Radiant software and open the New Project wizard.

To open the New Project wizard:

1. If you haven't already, start the Radiant software by doing one of the following:

   - On Windows, go to the Start menu and choose **Lattice Radiant Software > Radiant Software**.

   - On Linux, enter the following on a command line:

     <*Radiant_install_path*>/bin/lin64/**radiant**

   The main window of the Radiant software opens along with an Update dialog box. This takes a moment.

2. If the Update dialog box says "No update found," click **Close**. Otherwise, install the update and restart the Radiant software.

Now you have a clear view of the Start Page. With the Start Page you can easily open a new project, open a recent project, and access information.

3. Click the **New Project** button.

   The New Project wizard opens.

4. Click **Next**.

   The Project Name page opens.

Setting the Project Name and Location

Specify a name and location for the project files and for a design "implementation."

An implementation is one version of your design. You can have more than one implementation, so that you can experiment with different design approaches. A project starts with one implementation. You can add more later.

To fill out the Project Name page:

1. Specify the project name: **CLNXtutorial.**
2. Browse to where you want to store the project's files. This tutorial uses C:/my_radiant_tutorial. But you can use any location.
3. Make sure the **Create subdirectory** option is selected.

   The wizard automatically adds a folder for your project, which is shown immediately below the Location box.

4. Specify an implementation name. We'll use the default: **impl_1**.

   The directory for the implementation is displayed in the Location box.

5. Click **Next**.

   The Add Source dialog box appears.

Adding Source Files

Since the tutorial comes with source files, you can add them now. Source files can be added at any time or created with the Radiant software.

To add existing source files:

1. Click **Add Source**.

   The Import File dialog box appears.

2. Browse to: <*Radiant_install_path*>/docs/tutorial/**crosslink_nx_tutorial**.

3. Select the following files (**Control+A** will do it.):
   - count32.v
   - testbench.v
   - top.v
   - top_test1.v
   - topcount.v
4. Click **Open**.
5. Confirm that the New Project wizard is showing all of the files.

   If any files are missing, click **Add Source** again.

   If any extra files are showing, select the files and click **Remove Source**.
6. Make sure that the **Copy source to implementation source directory** option is selected.

   This makes copies of the files in your implementation instead of referring to the original files.

   The **Create empty constraint files** option is not needed for this tutorial.
7. Click **Next**.

   The Select Device dialog box appears.

Selecting a Device

Specify exactly which Lattice FPGA you plan to use. This selection can be changed at any time if you find a need to.

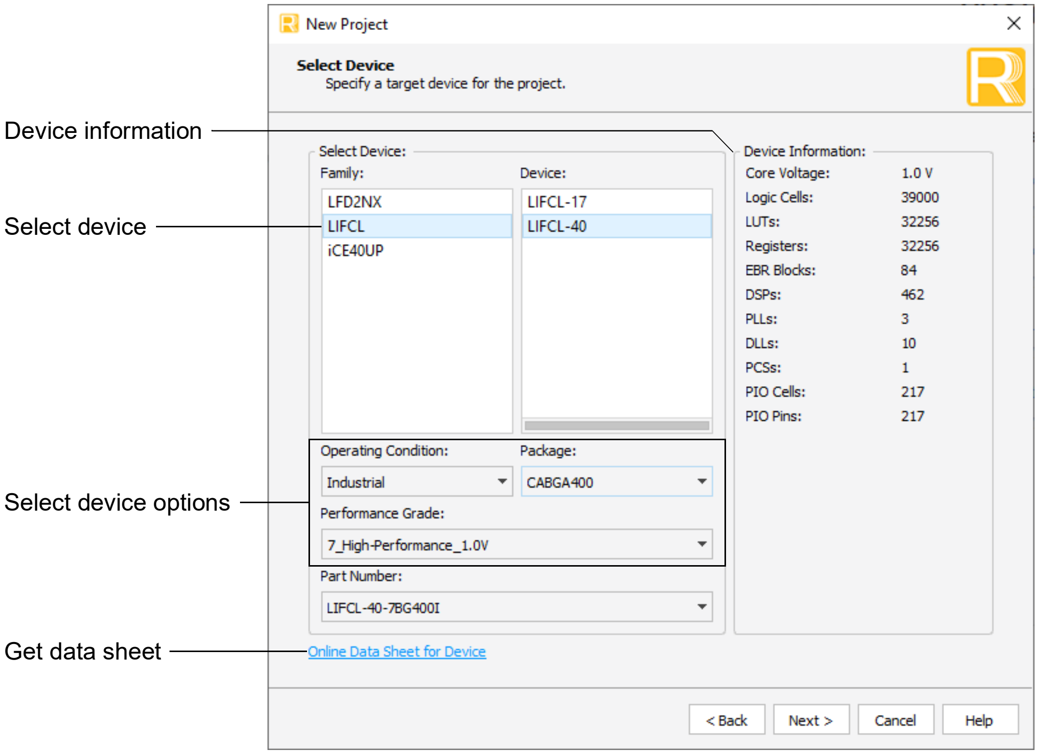
To select a device:

1. Select the device family: **LIFCL** (which is the part number code for the CrossLink-NX family).
2. Select the specific device within the family: **LIFCL-40**.
3. Select the following device options:
   - Operating Condition: **Industrial**
   - Package: **CABGA400**
   - Performance Grade: **7_High-Performance_1.0V**

   The Part Number, at the bottom, changes as you make selections.

   The dialog box should resemble Figure 1. At the bottom is a link to get a data sheet for the device. At the right is Device Information, including a list of resources in the device such as the number of LUTs (look-up tables), registers, and PIO (programmable I/O) pins.

Figure 1: New Project Wizard's Select Device Page



4. Click **Next.**

   The Select Synthesis Tool dialog box opens.

Finishing the Project Setup

Finish by selecting a synthesis tool and confirming all the choices that you made in the New Project wizard. Then you'll see how a project looks in the Radiant main window.

To finish setting up the project:

1. Select a synthesis tool. This tutorial requires **Lattice LSE** (Lattice Synthesis Engine).

   Note

   If you choose to use Synopsys® Synplify Pro® for Lattice, some of the Radiant tools, such as Timing Constraint Editor and Netlist Analyzer, will not be available. Synplify Pro may have similar tools but they are not covered in this tutorial.

2. Click **Next.**

   The Project Information dialog box appears. This dialog box summarizes the choices you made in the wizard. If you want to change any of them, click **Back**.

3. Click **Finish**.

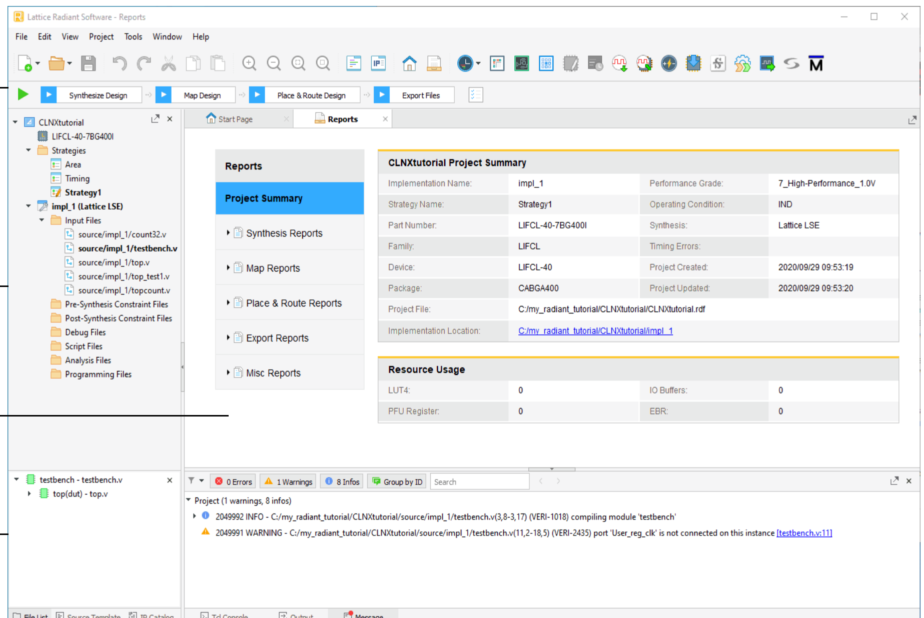
Several views are added to the Radiant window to give you easy access to files, tools, and messages from the software. Figure 2 identifies the views in the default arrangement. On the left is the File List view showing the files and other components of the project that you just created. On the right is the Reports view showing a summary of other information about the project.

4. In the File List view, right-click **testbench.v** and choose **Include for** > **Simulation**.

   By default all input files are marked for both synthesis and simulation. But you do not want the testbench when you synthesize the design.

   You will see activity in the Output view, at the bottom of the window, as the Radiant software re-analyzes the design hierarchy. In the File List view, top.v is bolded to show that it holds the top module. The Hierarchy view, which is underneath the File List view, also changes.

Figure 2: Radiant Main Window



About the File List View

The File List view gives easy access to the components of the project including:

- The device.
- Strategies, which are collections of option settings for how the design is processed. Start with Strategy1, a balanced approach. If you are having trouble fitting a design into a device, try the Area strategy. If you are

having trouble with timing, try the Timing strategy. You can create your own strategy by cloning one of these.

- Implementations, which are all the source files for a version of a design. A project can have several implementations so that you can experiment with different design approaches.
- Input Files, which are the design files.
- A variety of other files that may be created in the project.

Bold Text Notice that some of the items, such as Strategy1 and impl_1, are written with bold text. You can have multiple components of a given type, but usually only one can be active. So impl_1 is the active implementation and Strategy1 is the active strategy for impl_1.

An exception to this rule is in the Input Files, which are the HDL design files. These are all active. In Input Files, bold text indicates a file with a top module. The Radiant software automatically analyzes the Input Files for the design hierarchy, which can be seen in the Hierarchy view. So top.v holds the top module in impl_1.

Commands Right-click an item to see the available commands for that item. The commands vary depending on the item. There are commands for changing properties, adding files, changing the active file, and more.

Task 2: Add HDL Code

The Radiant software has a few tools to help you create HDL code:

- Source Editor is a text editor optimized for HDL code. Source Editor color codes different parts of HDL code, tracks parenthesis pairs, and can collapse blocks for easier reading.
- Source Template provides templates for common functions and structures to help you build Verilog, VHDL, and constraint files. The templates can be simply dragged and dropped into Source Editor and filled in there.
- IP Catalog provides a collection of pre-built modules that you customize through a dialog box. The Radiant software comes with many commonly used functions such as I/O, arithmetic, and memory. Many more-specialized functions can be downloaded.
- PMI (Parameterized Module Interface) provides a collection of modules similar to those that come with IP Catalog. But with PMI, you customize by changing parameters in the instantiation code, which is available in Source Template. IP Catalog tends to provide more ways to customize its modules. But PMI may be easier when you need several similar, but not identical, instances of a module.

Of course, you can also create code outside of the Radiant software and import the files into your project.

In this task you will use all these tools to add a few modules to finish the design.

Generating a Module from IP Catalog

In this section, you will customize and generate a phase-locked loop (PLL) module to add to the design.

To customize and generate a PLL module:

1. Click the IP Catalog tab (lower-left corner, under the File List view).

   IP Catalog replaces the File List view.

   IP Catalog comes with a large variety of architecture, arithmetic, and memory modules. These are under the IP on Local tab. Click the IP on Server tab to see more-specialized modules that you can download. Take this opportunity to expand the folders and see what's available to you.

2. On the IP on Local tab, expand Module > Architecture_Modules and hover over **PLL**.

   To the right, a blue circle with a question mark ? appears. You may need to scroll to the right to see it.

3. Click the blue circle.

   A brief description of the module appears in the tool area. To get more information about this module, click **User Guide** in the description. This will download a PDF file to your browser.

4. Double-click **PLL**.

   The Module/IP Block Wizard opens.

5. For Component name, enter **my_pll**. Use the default for the "Create in" location.

6. Click **Next**.

   The wizard changes to a block diagram of the module and a table of properties and values.

   As you can see, there are several ways that you can customize this module. Each tab provides more options.

   Some of the properties are grayed out because they are read-only, such as a value calculated from the option settings. But usually, a grayed out property becomes available to change depending on other option settings. For example, if you change Configuration Mode to Divider, the CLKI: Divider Value option becomes available.

7. In the General tab, set the following values:

   - Configuration Mode: **Frequency**
   - CLKI: Frequency: **125**
   - CLKFB: Feedback Mode: **INTCLKOP** (Feed back CLKOP, the primary output clock, internally.)
   - CLKOP: Frequency Desired Value: **200** (Scroll down to the Primary Clock Output section.)

8. Click the Optional Ports tab. This is where the reset and lock pins come from.

9. Clear the **Provide PLL Lock Signal** option.

   The diagram changes to remove the lock_o pin.

10. Click **Calculate**.

    A box opens with messages. This may take a moment. Check for error messages.

    Note: Most IP do not have a Calculate button.

11. Click **Generate**.

    The Check Generated Result page appears. This may take a moment.

12. Ensure that **Insert to project**, in the lower-left corner, is selected and click **Finish.**

13. Go back to the File List view to see that my_pll/my_pll.ipx has been added to the list of Input Files. The module comes with a few associated files. In the Hierarchy view, a my_pll module appears.

Instantiating the Module

When IP Catalog generates a module, it also creates templates for instantiating the module. You just copy the Verilog or VHDL code, paste it into your design, and fill in the blanks: instance name and I/O signals.

To instantiate the PLL module:

1. In the File List view, double-click **source/impl_1/top.v**.

   The file opens in Source Editor.

2. Scroll down to a comment that says: `//*** Add my_pll here. ***`

3. In the File List view, right-click **my_pll.ipx** and choose **Copy Verilog Instantiation**.

4. Go to Source Editor and paste the code below the comment.

   Note

   For VHDL, follow a similar process using the Copy VHDL Component and Copy VHDL Instantiation commands.

5. You need to fill in a name for the instance and signal names for the ports. See below for the finished instantiation command. Bold is the text that you enter.

   ```
   //*** Add my_pll here. ***
   my_pll pll_inst(.clki_i(oclk),
            .rstn_i(rstn),
            .clkop_o(pclk));
   ```

6. Click the Save button in the toolbar.

   In the Hierarchy view, the my_pll module moves to be under the top module.

7. Close Source Editor and IP Information by clicking the X in their tabs.

Adding RAM with PMI

Suppose you want to add a few RAM blocks to your design. There are several types available in IP Catalog. But, for each RAM block, you would have to open IP Catalog, select options, and generate the module from the beginning.

If the RAM blocks are similar, it might be easier to use PMI from Source Template. You can set up one block and then make copies of it to modify.

In this section, you will create two RAM DQ blocks. The first RAM will be 32-bits wide, 64-words deep, and with a 9-bit address. It will include a register on the output with a synchronous reset. The second RAM will be smaller, only 32 words, and without the registered output.

To add RAM with PMI:

1. In the File List view, double-click **source/impl_1/top_test1.v**.

   The file opens in Source Editor.

2. Scroll down to a comment that says: `//*** Add RAM here. ***`

3. Click the Source Template tab (under File List).

   Source Template replaces the File List view.

   Source Template offers a large variety of Verilog, VHDL, and constraint templates. Take this opportunity to expand the folders and see what is available to you.

4. Expand Verilog > PMI Templates > LIFCL and LFD2NX PMI and click **ram_dq** in the list of modules.

   The instantiation template for the pmi_ram_dq module appears in the lower space of Source Template.

   To get more information about this module, get *Memory Modules User Guide* at:

   www.latticesemi.com/view_document?document_id=52685

5. Drag ram_dq from the list to the space below the “Add RAM here.” comment.

   The template is added to the file.

6. You need to fill in the parameter values.

   To the right of the parameters are comments showing what kind of values can be used. Many of these are text inside quotes. The straight line, |, means OR. When quoted text is available, you can copy it, including the quotes, and paste it in the parenthesis. Any parameter left without a value uses a default value.

   - pmi_addr_depth: **64**
   - pmi_addr_width: USER_REG_ADDRWIDTH (This parameter is defined a little higher up in the file.)
   - pmi_data_width: USER_REG_DATAWIDTH (This parameter is defined a little higher up in the file.)

- pmi_regmode: **"reg"**
- pmi_gsr: **"enable"**
- pmi_resetmode: **"sync"**
- pmi_family: **"LIFCL"**

7. Change `<your_inst_label>`, which is near the middle of the template to the instance name, **ram1**. Delete the angle brackets.
8. You can copy the port names from the wire definitions just above the “Add RAM here.” comment.

   The result in Source Editor should look like Figure 3 (but with color). Bold is the text that you enter.

Figure 3: First Finished PMI RAM

```
// *** Add RAM here. ***
pmi_ram_dq
#(
  .pmi_addr_depth       (64), // integer
  .pmi_addr_width       (USER_REG_ADDRWIDTH), // integer
  .pmi_data_width       (USER_REG_DATAWIDTH), // integer
  .pmi_regmode          ("reg"), // "reg"|"noreg"
  .pmi_gsr              ("enable"), // "enable"|"disable"
  .pmi_resetmode        ("sync"), // "async"|"sync"
  .pmi_init_file        ( ), // string
  .pmi_init_file_format ( ), // "binary"|"hex"
  .pmi_family           ("LIFCL")  // "LIFCL"|"LFD2NX"|"common"
) ram1 (
  .Data       (User_reg_wdata1),  // I:
  .Address    (User_reg_addr1),  // I:
  .Clock      (User_reg_clk),  // I:
  .ClockEn    (User_reg_en),  // I:
  .WE         (User_reg_wr_rdn1),  // I:
  .Reset      (reset),  // I:
  .Q          (User_reg_rdata1)   // O:
);
```

Adding More RAM with PMI

Now make the second RAM block.This is probably easier and faster than going through IP Catalog twice. You will have two different RAM blocks and could easily create more.

To add more RAM with PMI:

1. Select all of the code of the ram_dq template that you just finished. (Clicking a line number selects the whole line.) Copy this code and paste it below the first instance.
2. In the copy, change the following parameters:
   - pmi_addr_depth: **32**
   - pmi_regmode: **"noreg"**

3. Change the instance name to **ram2**.
4. Similarly, change the 1 to a 2 in the ports:
   - Data: User_reg_wdata**2**
   - Address: User_reg_addr**2**
   - WE: User_reg_wr_rdn**2**
   - Q: User_reg_rdata**2**

   The result in Source Editor should look like Figure 3. Bold is the text that you change.

Figure 4: Second Finished PMI RAM

```
pmi_ram_dq
#(
  .pmi_addr_depth        (32), // integer
  .pmi_addr_width        (USER_REG_ADDRWIDTH), // integer
  .pmi_data_width        (USER_REG_DATAWIDTH), // integer
  .pmi_regmode           ("noreg"), // "reg"|"noreg"
  .pmi_gsr               ("enable"), // "enable"|"disable"
  .pmi_resetmode         ("sync"), // "async"|"sync"
  .pmi_init_file         ( ), // string
  .pmi_init_file_format  ( ), // "binary"|"hex"
  .pmi_family            ("LIFCL")  // "LIFCL"|"LFD2NX"|"common"
) ram2 (
  .Data       (User_reg_wdata2),  // I:
  .Address    (User_reg_addr2),  // I:
  .Clock      (User_reg_clk),  // I:
  .ClockEn    (User_reg_en),  // I:
  .WE         (User_reg_wr_rdn2),  // I:
  .Reset      (reset),  // I:
  .Q          (User_reg_rdata2)   // O:
);
```

5. Click the Save button in the toolbar.
6. Close Source Editor.

Task 3: Verify Functionality with Simulation

Now that the design is finished, you can simulate it to test the logic. With the Radiant software, you can run a simulation at different stages of the development process:

- Before synthesis (RTL)
- Post-synthesis
- Post-route, gate-level
- Post-route, gate-level and timing

In this tutorial we will just do the RTL simulation. For the other stages, the process is similar.

For a simulator, this tutorial uses the Mentor® ModelSim® Lattice FPGA Edition simulator that comes with the Radiant software on Windows.

If you are not using an HDL simulator that is integrated with the Radiant software, you can skip this task. "Integrated" means that you can run the simulator from the Radiant software. What is available depends on your operating system. You can use other simulators outside of the Radiant software.

If you are not using the ModelSim that comes with the Radiant software, you need to compile the primitive library. For instructions, open the Radiant Help and see User Guides > Simulating the Design > Third-Party Simulators.

This tutorial comes with a simple testbench. You will probably create your own testbenches using your simulator. Simulators usually include tools for creating testbenches.

Starting a Simulation Run

While you can start your simulator directly, it's good to create a simulator project that allows you to run the simulator from the Radiant software.

To start simulating the design:

1. Choose **Tools** > **Simulation Wizard**.

   The Simulation Wizard dialog box appears.

2. Click **Next.**

   The Simulator Project Name page appears.

3. Enter the Project name: **sim_test**.

   Leave the other settings at their defaults.

4. Click **Next**.

5. If you left the default for the project location, a dialog box opens saying, "sim_test does not exist. Do you want to create it?" Click **Yes**. This creates a sim_test folder.

   The Add and Reorder Source page appears.

6. Make sure all source files are present in the Source Files list. You can modify this list but that is usually not needed. Instead, leave the **Automatically set simulation compilation file order** option selected. Click **Next**.

   The "Parse HDL files for simulation" page appears.

7. Verify that the simulation top module is "testbench." This is shown at the bottom of the dialog box. Click **Next.**

   The Summary dialog box appears.

8. Make sure that the **Run simulator**, **Add top-level signals to waveform display**, and **Run simulation** options are all selected.

9. Click **Finish**.

   The selected simulator launches and the simulation starts automatically. After completing the simulation, the waveform appears. This takes several moments. Wait for the waveform to appear.

   If you see the Welcome to ModelSim dialog box, select **Don't show this again**, at the bottom of the dialog box, and click **Close**. Do not click Jumpstart.

10. Look at the File List view in the Radiant window. Under Script Files, you see sim_test/sim_test.spf.

You can rerun the simulation by double-clicking the .spf file. The Simulation Wizard will open with a Skip to End button. Click it to jump to the last page of the wizard. Then click **Finish** to start the simulation running.

Checking the Simulation Results

Note

If you are not using ModelSim Lattice FPGA Edition, you can skip the rest of the simulator task.

Now that you've run the simulation, you can see what happened on the top-level signals of the testbench as shown in Figure 5. ModelSim stopped automatically after the first microsecond of simulation time. The testbench is set to run longer, over 5 μs, but this is enough to see the startup.

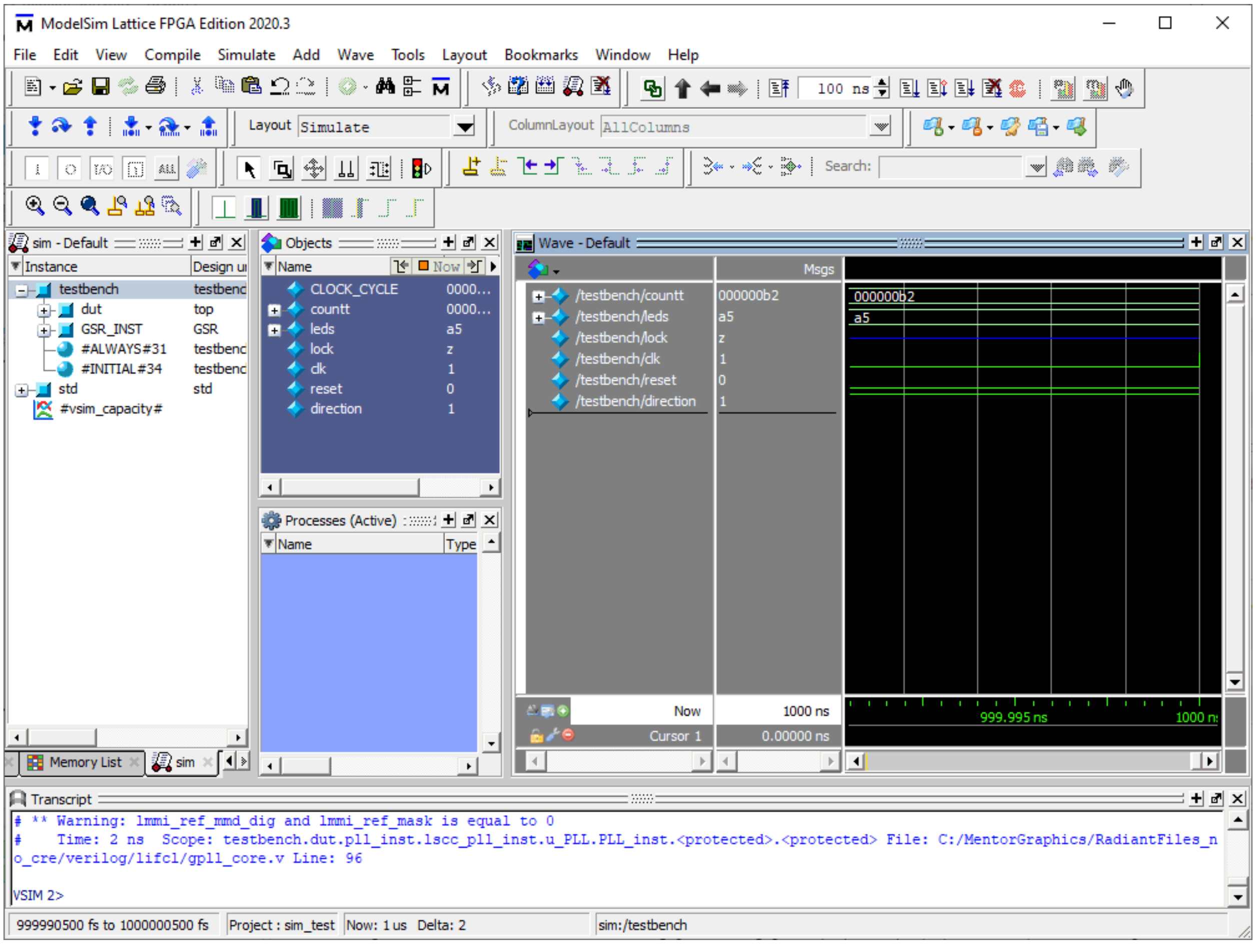
To check the simulation results:

1. You probably want to expand the Wave view. Do one of the following:
   - Expand the ModelSim window.
   - Undock the Wave view. Click the Dock/Undock button that is in the upper-right corner of the Wave view. Then expand the Wave window.
2. To make other adjustments to the Wave view, choose **Simulate > Runtime Options**.

   The Runtime Options dialog box opens showing a variety of options that you can set.
3. Make the following changes in the Defaults tab:
   - For Default Radix, select **Hexadecimal**. This is how the values of signals are normally displayed.
   - For Default Run, enter **100ns**. This is the amount of time that the Run command simulates.
4. Click **OK**.

   The values shown in the Objects and Wave views change to hexadecimal.

Figure 5: Simulated Waveform



5. Choose **Wave > Zoom > Zoom Full** or click the Zoom Full button in the toolbar to see the whole waveform. The Zoom toolbar looks like this:

   In the Wave view, you see the reset signal activated by the testbench. This drives the leds value to zero. After reset is released, countt starts counting.

6. The values of countt may not be visible. Click the Zoom In button in the toolbar until you can see the values.

7. Choose **Simulate > Run > Run 100** or click the Run button to see more of the simulation. The Run toolbar looks like this:

   Another 100 ns is added to the waveforms. This is the time you set in the Runtime Options dialog box. You can change this amount in the box next to the Run button.

8. Click anywhere to see what the values are at that moment.

The nearest cursor (a vertical yellow line) jumps to where you clicked. The value column shows all the values at that moment.

You can click on the cursor and drag it to other positions on the timeline. You can also return to the cursor after scrolling away by clicking the Zoom In on Active Cursor button.

Rerunning the Simulation

In ModelSim you can make changes in the simulation and rerun it. For example, you can add more signals.

To add a signal and rerun the simulation:

1. In the List view, click the sim tab (also know as the Structure view), and expand:

   testbench > dut > top_test1_inst > counter > counter

2. Click on **#ALWAYS#12**. This is the ball icon underneath counter.

   The Objects view changes.

3. Drag **countai** from the Objects view to the Wave view.

4. Rerun the simulation to see what is happening with the countai register. Choose **Simulate > Restart** or click the Restart button.

   The Restart dialog box opens with a variety of features that you might have changed. You can leave them all selected.

5. Click **OK**.

   The waveforms in the Wave view disappear.

6. Then choose **Simulate > Run > Run -All** or click the Run -All button.

   The Finish Vsim dialog box opens. It asks if you want to finish.

7. Click **No**.

   Warning

   Do *not* click Yes. If you do, the $finish statement in the testbench causes ModelSim to exit.

   If this happens, go to the File List view in the Radiant window and look under the Script Files folder. Double-click **sim_test/sim_test.spf** to restart ModelSim.

   ModelSim's source editor opens with the testbench.v file.

8. Close testbench.v and go back to the Wave view. Now you see the full 5 μs.

9. You can take this opportunity to explore ModelSim more.

   There's a lot more that you can do with ModelSim. For more information, see the **Help** menu in the ModelSim window.

10. When you are done exploring ModelSim, choose **File > Quit** to close ModelSim.

The Quit Vsim dialog box opens.

11. Click **Yes**.

Task 4: Set Location Assignments

You will use the Device Constraint Editor to assign signals to the pins of the FPGA. There are a few ways to do this:

- Drag the port from the Editor's list view to the Package View, which is a graphic layout of the FPGA's pins.
- Right-click the port in the spreadsheet to open the Assign Ports dialog box, which presents a list of all appropriate pins.
- Type the pin number in the spreadsheet.

Since we have a list of the pin numbers, typing is probably the easiest way.

To assign pins:

1. Choose **Tools >** **Device Constraint Editor**.

   The Device Constraint Editor appears.

2. If you see a yellow bar with a message saying the "Design database in memory is outdated," click **Reset Database**, which is to the right of the message.
3. Click the **Port** tab, in the lower-left.
4. In the spreadsheet, scroll to the bottom and find the rstn port.
5. Click in the Pin cell and enter **G19**.

   In the Device View, G19 shows a green dot, indicating an input port.

6. In the spreadsheet, right-click on **Name** and choose **Filter > Enable Filter**.

   A button for a drop-down menu appears on each column title.

7. Click the drop-down button in the Name column.

   A filter list appears.

8. In the Search box, type **leds**.

   The filter list is reduced to the leds ports.

9. Click **OK**.

   The spreadsheet is reduced to the leds ports.

10. Fill in the Pin cells of the leds ports with the following pins. Start at the top of the list. After typing the pin number, press the down arrow key to get to the next cell.

    - leds[0]: **E17**
    - leds[1]: **F13**
    - leds[2]: **G13**

- leds[3]: **F14**
- leds[4]: **L16**
- leds[5]: **L15**
- leds[6]: **L20**
- leds[7]: **L19**

As you enter values, the matching spots in the diagram are filled in with blue, indicating output ports.

11. Click the Constraint Preview button.

    The Preview dialog box opens showing the constraint commands. See Figure 6.

Figure 6: Device Constraints

```
ldc_set_location -site {G19} [get_ports rstn]
ldc_set_location -site {E17} [get_ports {leds[0]}]
ldc_set_location -site {F13} [get_ports {leds[1]}]
ldc_set_location -site {G13} [get_ports {leds[2]}]
ldc_set_location -site {F14} [get_ports {leds[3]}]
ldc_set_location -site {L16} [get_ports {leds[4]}]
ldc_set_location -site {L15} [get_ports {leds[5]}]
ldc_set_location -site {L20} [get_ports {leds[6]}]
ldc_set_location -site {L19} [get_ports {leds[7]}]
```

12. Click the Save button in the toolbar.

    The Save dialog box opens.

13. Name the file **eval_board** and click **Save**.

    In the File List view, eval_board.pdc appears under the Post-Synthesis Constraint Files folder. Device constraints are not used in synthesis.

14. Close the Device Constraint Editor.

Task 5: Process the Design

Processing a design involves a few steps that convert the high-level Verilog and VHDL description into code that can actually program a specific FPGA:

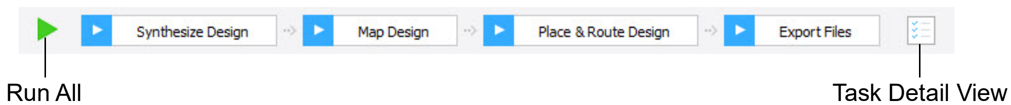
1. Synthesize converts HDL into a gate-level netlist that is optimized for the FPGA.
2. Map converts the netlist into a network of device-specific components, such as PFU (programmable function units) and I/O buffers.
3. Place and route converts the mapped network into specific components and signal routes within the device.
4. Export converts the place-and-route specifications into code to program the FPGA.

Each step also produces a set of reports that describe how the process was run and the results. If a process fails, its reports are the place to start troubleshooting.

About the Process Toolbar

Use the Process Toolbar (shown below) to run the processes.

Figure 7: Process Toolbar



With a single click you can run any individual process including any preceding processes that have not been run yet. Click the Run All button to run the whole sequence. Right-click a process button to get a menu of options for running the process.

Click the Task Detail View button to select other files to generate while running the processes. Timing analysis and simulation files are available.

While a process is running, the Run All button changes to the Stop button. Click the Stop button to stop the processing.

When a process completes, its button shows its success or failure with a green check mark or a red X.

Processing the Design

In this task, you will step through the processes one-by-one and check the reports after each. However, in normal practice, you would probably run the whole sequence and then check the results.

To process the design:

1. In the Process Toolbar, click **Synthesize Design**.

   Task Detail View opens and tracks completion of the processes.

2. In the Reports view, click **Synthesis Reports**.

   These reports give details of how synthesis ran. They also give detailed information about use of device resources and timing. Hover over the Contents button in the top-right corner to get links to different sections of a report.

3. When you finish looking at the synthesis reports, click **Map Design**.
4. In the Reports view, click **Map Reports** and examine the available reports.
5. When you finish looking at the map reports, click **Place & Route Design**.

6. In the Reports view, click **Place & Route Reports** and examine the available reports.
7. When you finish looking at the place and route reports, click **Export Files**.
8. In the Reports view, click **Export Reports** and examine the available reports.

   At the end of the Bitstream report is the pathname of the bitstream file: <*project_path*>/impl_1/**CLNXtutorial_impl_1.bit**.

Task 6: Examine the Layout

After place-and-route, you can see a display of the layout using Physical Designer and cross-probing between different views.

To see the layout:

1. Choose **Tools > Physical Designer**.

   Physical Designer shows a large-component layout of your design.
2. To the left of the diagram are lists of instances and IOs. Expand the Instances list and choose one of the primitives, such as Instances > top_test1_inst > **leds_i8.ff_inst**.

   The display zooms to the component.
3. Right-click on the component and choose **Physical Designer Routing Mode**.

   The Routing Mode opens with the display zoomed to the same component. The Routing Mode provides a detailed layout of your design that includes switch boxes and physical wire connections.
4. In the toolbar of Physical Designer, click the arrow of the Routing button and choose **IO Mode**.

   Physical Designer changes to show the I/O of the device.
5. In the list, expand Instances, scroll down to the bottom, and click **rstn_pad.bb_inst**.

   Physical Designer zooms in to the I/O for rstn: G19, which you set in the constraint file. The padlock symbol shows the pads that have constraints on them.

   You can do this for any of the instances labeled with “_pad” and for any of the items in the IOs list.
6. Close Physical Designer.

Task 7: Analyze Power Consumption

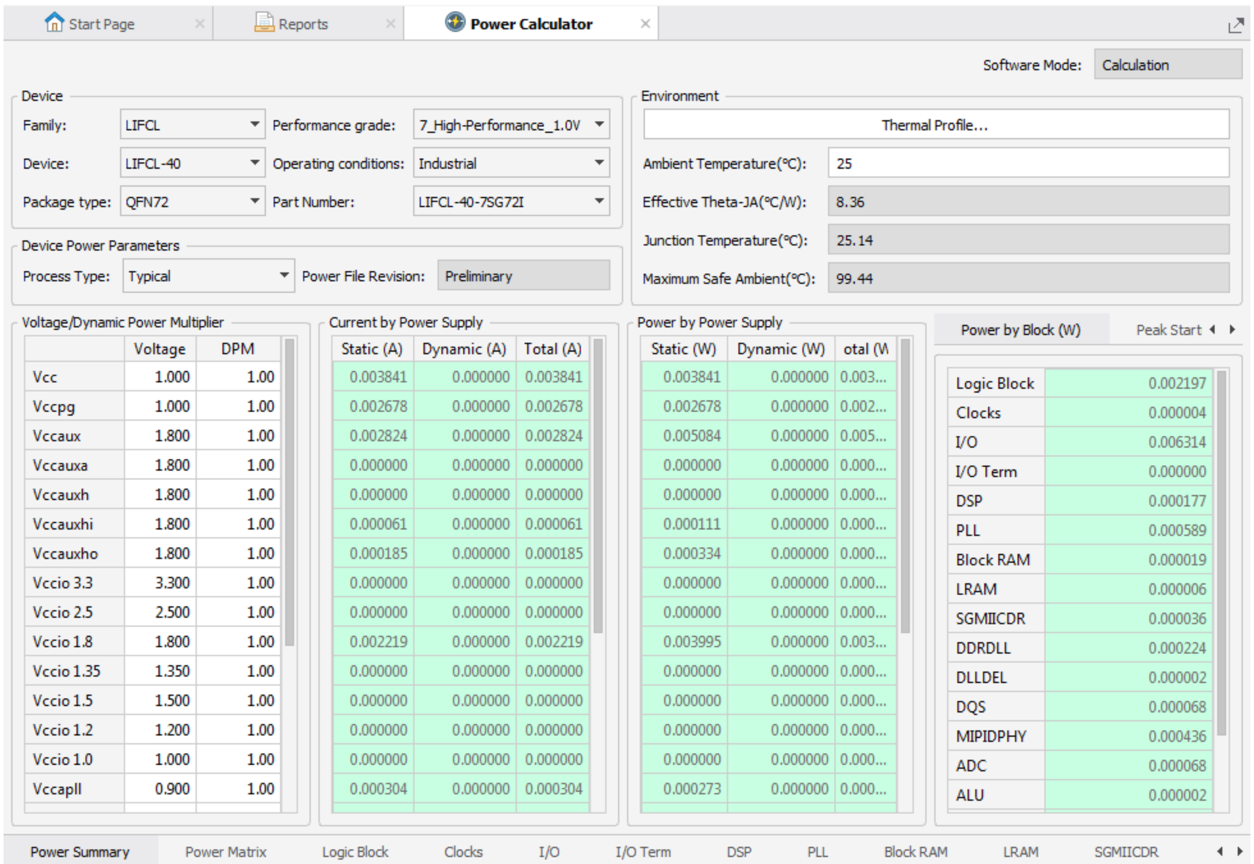
Power Calculator estimates the power dissipation for a given design. Power Calculator uses parameters such as voltage, temperature, process variations, air flow, heat sink, resource utilization, activity, and frequency to calculate the device's static and dynamic power consumption.

To analyze power consumption:

1. Choose **Tools >** **Power Calculator**.

   Power Calculator opens in Calculation mode as shown in Figure 8.

Figure 8: Power Calculator



Power Calculator provides two modes for reporting power consumption:

- Estimation mode:

  In estimation mode, Power Calculator provides estimates of power consumption based on the device resources or template that you provide. This mode enables you to estimate the power consumption for your design before the design is complete or even started.

- Calculation mode:

  In calculation mode, Power Calculator calculates power consumption on the basis of device resources taken from a design's .udb file or from an external file such as a value change dump (.vcd) file, after placement and routing. This mode is intended for accurate calculation of power consumption, because it is based on the actual device utilization.

Editing data in white cells, such as voltage, frequency, activity factor, and thermal data, does not change the mode. Editing data in yellow cells, such as design data, will change calculation mode to estimation mode.

2. For Process Type in the Device Power Parameters section, choose **Worst**.
3. Click **Thermal Profile** in the Environment section.

   The Power Calculator – Thermal Profile dialog box appears.
4. For Board Selection, choose **Small Board**.
5. Click **OK**.

   After a moment, the new temperature results become available in the Environment section.
6. Close Power Calculator.

   A Confirm dialog box appears.
7. Click **Yes**.
8. Give the file a name, such as **eval_board**, and click **Save**.

   In the File List view, a .pcf file appears under Analysis Files.

Task 8: Add an On-Chip Debug Module

Many times you will want to see what is happening inside the FPGA while it is running. After you have your design in an FPGA on a prototype circuit board, you may find problems that did not show up in simulation. The Radiant software allows you to see what's happening inside the FPGA and to even change register values while your system is running.

The Radiant software does this by helping you create a "debug module" and adding it to your design. The module is a combination of two types of "cores:"

- Logic Analyzer monitors selected signals for events that you define. When these events happen, the values of these and other signals are uploaded to the Radiant software. You can see the values in a waveform viewer or save them for other software.
- Controller gives ongoing access to selected signals and registers. A Controller core has virtual switches and LEDs to monitor signals, read and write access to user-defined memory, and read and write access to the control registers of "hard IP." Hard IP are modules such as I2CFIFO, PLL, and DPHY that use features built into the FPGA.

The debug module can have up to 15 cores.

The Radiant software has two tools for on-chip debugging:

- Reveal Inserter, which you use to create a debug module and add it to your design.
- Reveal Analyzer/Controller, which you use to control the debug module and to view test results. Reveal Analyzer/Controller is used after programming the FPGA with your combined design and debug module.

In this task, you will create a debug module with both Logic Analyzer and Controller cores.

About the Logic Analyzer Core

The Radiant software has a flexible system that lets you specify the signals you want to see and when you want to see them. The events that trigger sampling the signals can range from very simple to very complex. The Logic Analyzer core has several features that build up to a powerful logic analyzer:

- Trace signals are the signals that you want to analyze.
- Sample clock is a clock from your design. Trace signals are sampled on the rising edge of the sample clock.
- Trigger units are the signals that you want to monitor and logic to monitor them for certain values.
- Trigger expressions are logical or sequential combinations of the trigger units.
- Trigger events are logical or sequential combinations of the trigger expressions. Trigger events trigger uploading the trace samples to Reveal Analyzer/Controller.

You use Reveal Inserter to specify the signals that the Logic Analyzer core will use and to set up the trigger units and trigger expressions. But these are only initial settings. They can be modified in Reveal Analyzer/Controller without taking the time to process the design and program the FPGA again. Think of Reveal Inserter as creating capabilities and capacities that you can use with Reveal Analyzer/Controller.

In your own on-chip debugging, think about all the signals and all the trigger events that you might want to see, and build as much of that as possible into the debug module. The limitation, of course, is the FPGA resources, especially EBR (embedded block RAM) and distributed RAM, that you have left after installing your design.

Setting Up Trace Signals

Start by opening Reveal Inserter and adding a Logic Analyzer core. You'll add signals with a simple drag-and-drop action. Then set several options.

To set up trace signals for a Logic Analyzer core:

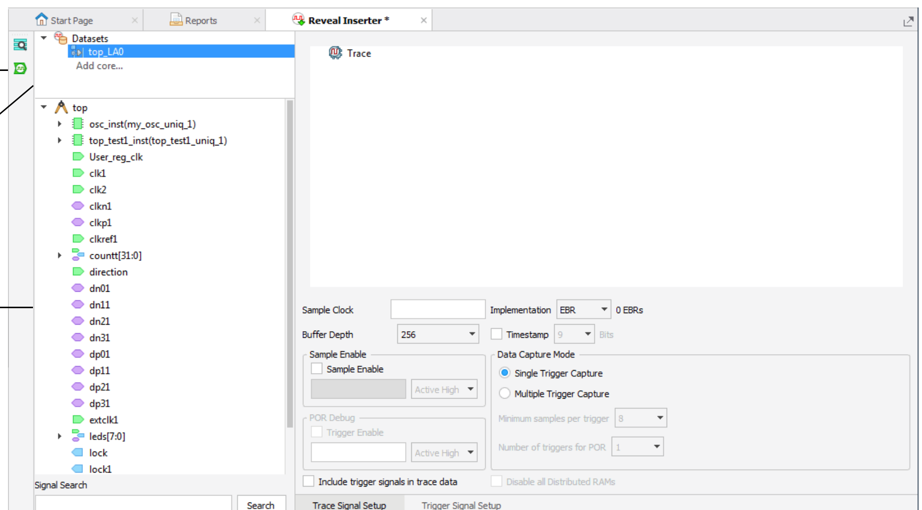
1. Choose **Tools >** **Reveal Inserter**.

   Reveal Inserter starts with a largely blank screen.

2. Choose **Debug > Add New Core > Add Logic Analyzer**.

   The Trace Signal Setup tab appears. The Dataset view expands to include a core named top_LA0. See Figure 9.

Figure 9: Trace Signal Setup



Reveal Inserter Toolbar
With Design Rule Check and Insert Debug buttons.

Dataset
Provides access to the cores in the debug module.

Design Tree
Provides access to all signals in the design.

Signal Search
Finds signals in the Design Tree view.

Core Setup Area
Shows setup tabs of the selected core.

3. Click on the **Trace Signal Setup** tab, if it is not already selected.

4. In the Signal Search box, enter **countai**.

   The Data Tree view expands to show countai[31:0] selected.

5. Drag the **countai[31:0]** bus to the Trace pane on the right.

   The name of the bus now appears in bold font in the Design Tree pane. The name is also labeled with "@Tc" to show that it is a trace signal.

6. Right-click the bus in the Trace pane and choose **Rename Trace Bus**. Name the bus **countai**.

7. Select the **Include trigger signals in trace data** option. This is in the lower-left of the window.

   With this option, all signals used to create triggers will also be in the trace list. This allows you to check how the triggers happen. You will set up the triggers later.

At the top, “Trigger Signals” is added to the trace list.

8. In the Signal Search box, enter **clk**.

   This time the Search Result dialog box opens with choices.

9. Select **top_test1_inst/counter/clk** and click **OK**.

   The signal is selected in the Design Tree view.

10. Drag the selected **clk** from the Design Tree view to the Sample Clock box.

    The name of the signal now appears in bold font in the Design Tree pane. The name is also labeled with “@C” to show that it is the sample clock signal.

Setting Up Trace Options

Besides selecting the trace signals, there are several options that you need to consider.

To set up trace options:

1. For Implementation, choose **EBR**.

   The implementation specifies what kind of RAM to use for the Logic Analyzer core. Normally EBR (embedded block RAM) would be selected, but distributed RAM can be used if you are short of EBR.

   The number next to the Implementation menu shows how many EBR or slices are needed.

2. For Buffer Depth, choose **64**.

   Choose an amount at least as big as the number of samples multiplied by the number of trigger events. In this case, we plan for 16 samples for 1 trigger event. But it’s good to build in some extra capacity if your FPGA has the resources.

3. Select **Timestamp** and choose **10** bits.

   Timestamp provides a count of sample clock cycles from the beginning of a test run. The timestamp will show how long the test ran before triggering. The timestamp can also help associate triggers with external events.

   The number of bits is the size of the timestamps. Choose the smallest value that can hold the desired count.

   Note that the number of EBRs went up when you selected Timestamp.

4. Leave Sample Enable cleared.

   This option specifies a signal that can turn data capture on and off. Use sample enable to reduce the size of the trace buffer when there are stretches of data of no interest that are associated with a single signal.

5. For Data Capture Mode, select **Multiple Trigger Capture**.

   This option allows for multiple trigger events. The actual number of events will be set in Reveal Analyzer.

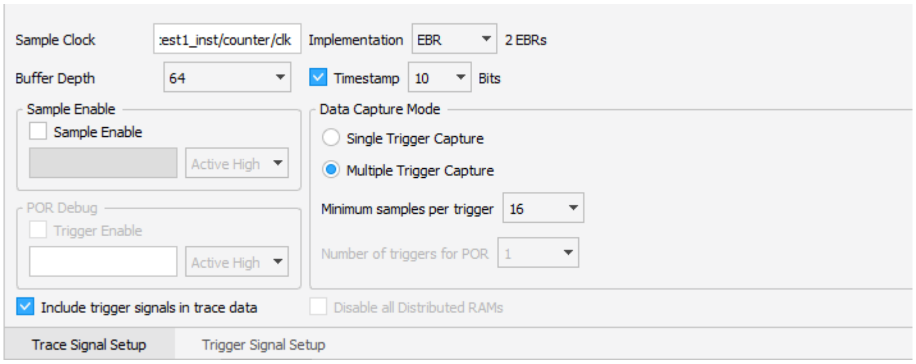
6. For “Minimum samples per trigger,” choose 16.

   This is the minimum number of samples for each trigger event. The maximum is set in Buffer Depth.

7. Ignore POR Debug and Disable all Distributed RAMS. These options are not currently available.

The lower part of the Trace Signal Setup tab should now resemble Figure 10.

Figure 10: Options in the Trace Signal Setup Tab



Setting Up Trigger Units

Here you will specify the signals and values that you want to watch for as part of the trigger. The values, in the Operator and Value cells, are just initial settings. They can be changed in Reveal Analyzer/Controller while running tests.

To set up the trigger units:

1. Click on the **Trigger Signal Setup** tab.

2. In the Trigger Unit section, at the top, click **Add**.

   A new row appears with default values.

3. Click in the Name cell and enter **tu_countai**.

4. Drag the **countai[31:0]@Tc** bus from the Design Tree pane to the Signals (MSB:LSB) cell in the Trigger Unit pane.

   In the Design Tree view, countai gains a Tg label to show that it is also a trigger signal.

5. Click in the Operator cell and choose **<=** from the drop-down menu.

   The operators are logical comparisons between the signal and a specified value. You can also choose rising or falling edges, or a series of values on a one-bit signal.

6. Click in the Radix cell and choose **Hex**.

   Radix is just the format used to show the value. Pick whichever radix is most convenient for you. If you are doing a lot of trigger units, you may want to choose a radix in the Default Trigger Radix menu (lower-right of the Trigger Unit area).

7. In the Value cell, enter **8**.

   This is the value that the trigger will look for.

8. Click **Add** to add a second trigger unit. Set up this trigger unit with the following values:

   - Name: **dir**
   - Signals: **direction** (This is top > direction. If you search for **dir***, it's "direction" in the results.)
   - Operator: **<=**
   - Radix: **Bin**
   - Value: **1**

Setting Up a Trigger Expression

Combine the trigger signals into a sequence that will trigger uploading the trace signals.

To set up the trigger expression:

1. In the Trigger Expression section, in the middle, click **Add**.

   A new row appears with default values.

2. In the Expression cell, select the tu_countai and dir trigger units by entering **dir THEN tu_countai**.

   This statement means: wait for dir to be true, then wait for tu_countai to be true. They do not have to be true at the same time.

   There are several logical and sequence operators available. These allow you to specify very specific trigger events. Operators include:

   - & - AND
   - | - OR
   - ^ - XOR
   - ! - NOT
   - () - Groups trigger units.
   - THEN - After the first unit is true, wait for the second one.
   - NEXT - Like THEN except the second unit must be true on the next clock cycle.
   - # - Adds a counter.
   - ## - Adds a counter. Events must be in consecutive clock cycles.

3. Set up the rest of this trigger expression with the following values:

   - RAM Type: **1 EBR**

     Choose the type of RAM to use for the expression. The menu also shows the amount needed.

   - Sequence Depth: **2**

     This cell shows the number of sequences, or units, in the expression. This cell is read-only.

   - Max Sequence Depth: **4**

     If you want to change the expression in Reveal Analyzer, this is the maximum number of sequences that will be possible.

   - Max Event Counter: **32**

     If you want to change the expression in Reveal Analyzer, this is the maximum number of counts that will be possible.

Setting Up Trigger Options

In addition to the trigger units and expressions, there are some options to consider.

To set up trigger options:

1. Select **Enable final trigger counter**.

   This option creates the ability to have a trigger event happen multiple times before capturing data.

2. For Event Counter Value, choose **4**.

   This is the maximum number of trigger events that will be possible before capturing data. You specify the actual number of such events in Reveal Analyzer. Choose the smallest number that will allow all the repetitions that you might want.

3. Leave Trigger Out clear.

   This option creates an output signal that pulses when the trigger event happens. This signal can be used by another Logic Analyzer core or go to an external I/O.

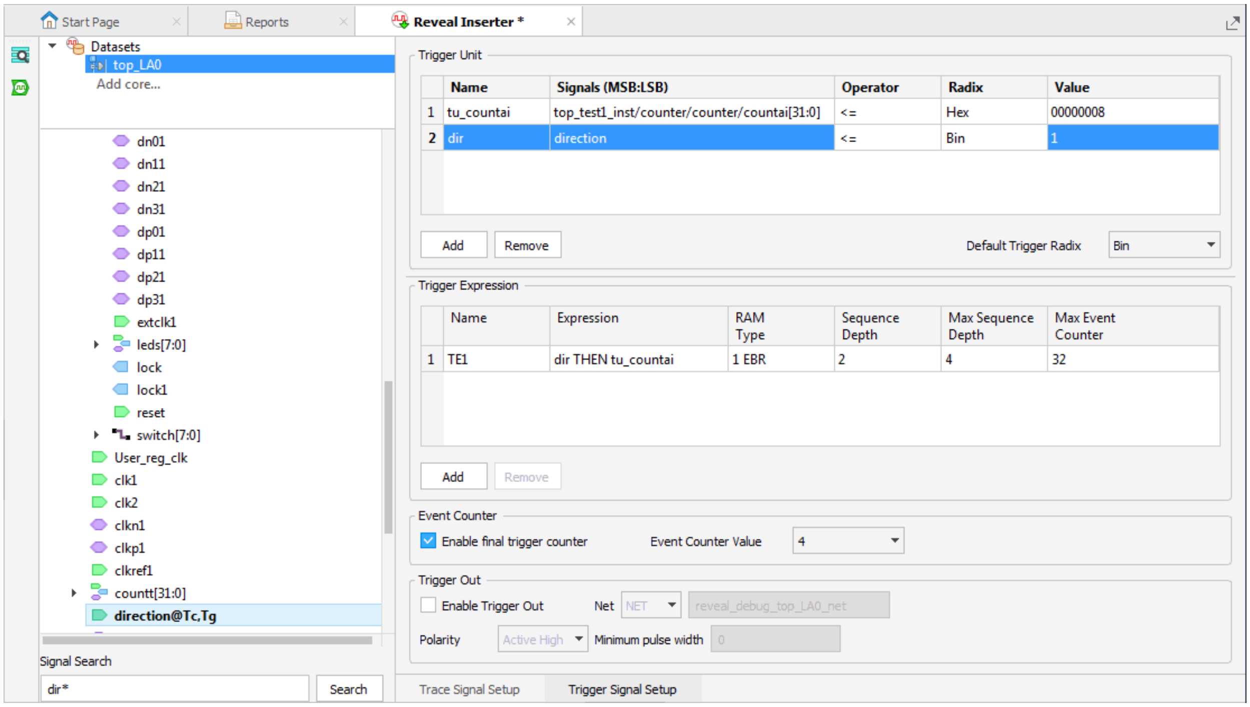
The Trigger Signal Setup tab should now resemble Figure 11.

Creating Virtual Switches and LEDs

In a Reveal Controller module, you can manually control and watch values inside the design by setting up virtual switches and LEDs.

The addresses that you see in the Reveal Controller core were assigned by the Radiant software while processing the design.

Figure 11: Trigger Signal Setup



To create virtual switches and LEDs:

1. Choose **Debug > Add New Core > Add Controller**.

   Most of the Reveal Inserter window changes to space to set up virtual switches and LEDs. There are also set-up tabs for accessing user registers and hard IP. The Dataset view expands to include top_Controller.

2. Click the **Virtual Switch & LED Setup** tab if it is not already showing.

3. Search for **switch** and select **top_test1_inst/switch[7:0]**.

   The Data Tree view expands to show switch[7:0] selected.

4. Drag **switch[7:0]** into the Signal column of the Switch List field.

   The field is filled with the individual switch signals. Above the Switch List field, the Width field changes to 8.

5. Search for **leds** and select **leds[7:0]**.

6. Drag **leds[7:0]** into the LED List field.

   The field is filled with the individual leds signals. Above the LED List field, the Width field changes to 8.

7. Make sure that the **Virtual Switch Setting** and **Virtual LED Setting** options, at the top of the Reveal Inserter window, are selected.

Creating User Register Access

You can set up read and write access to an internal register by simply specifying the register's control and data signals. You can access PMI, EBR, or distributed memory. In this tutorial, you are going to create read and write access of a pmi_ram_dq module.

To set up access to a register:

1. Click the User Register Setup tab.

   The tab shows a list of memory signal types.

2. In the Design Tree view, look through the modules under top_test1_inst, and find **ram1(pmi_ram_dq_uniq_1)**. Expand it.

3. Fill the User Register Setup tab by dragging the matching signals from the ram1 module:

   - Clock: **Clock**
   - Clock_enable: **ClockEn**
   - Wr_Rdn: **WE**
   - Address: **Address[7:0]**
   - WData: **Data[31:0]**
   - RData: **Q[31:0]**

4. Make sure that the **Enabled** check box, in the upper-right corner, is selected.

Creating Hard IP Access

Set up access to the control and status registers of the hard IP by simply selecting the IP you want. Hard IP are modules such as I2CFIFO, PLL, and DPHY that use features built into the FPGA.

To set up access to hard IP:

1. Click the Hard IP Setup tab.

   The tab shows a table with a list of all the hard IP in the design. In this tutorial, there's just the PLL.

2. In the Enabled column, select **PLL1**.

Inserting the Debug Logic

Now you will insert the debug logic into the design project.

To insert the debug logic:

1. Choose **Debug >** **Insert Debug**.

   The Insert Debug to Design dialog box opens with the top_LA0 and top_Controller cores listed.

2. Make sure that both cores are selected and that the **Activate Reveal File in design project** option is selected.

3. Click **OK**.

   The Save Reveal Project dialog box opens.

4. Accept the default filename, **CLNXtutorial.rvl**.

5. Click **Save**.

   In the File List view, the CLNXtutorial.rvl file is added to the Debug Files folder. In the Process Toolbar, all the green check marks are turned back to blue arrows. The design has been changed and needs to be processed again.

6. Close the Reveal Inserter window.

7. In the Process Toolbar, click the Run All button.

   Note

   When place-and-route finishes, the Timing Check Error dialog box appears!

8. Click **No** to stop the export process.

9. Go to the Reports tab.

   The Project Summary report shows timing errors. The Place & Route report also shows errors. You will fix these in the next two tasks.

Task 9: Examine Timing Analysis Results

Static timing analysis can determine if your circuit design meets timing constraints. Rather than simulation, it employs conservative modeling of gate and interconnect delays that reflect specific operating conditions with a specific FPGA.

You can produce timing analysis reports as part of the synthesize, map, and place-and-route processes. Before running a process, click the Task Detail View in the Process Toolbar and select Timing Analysis for that process. Timing analysis is selected by default, so you already have all three reports.

The reports have similar information shown in the same format. But they are based on information from each process:

- Post-synthesis timing analysis is based on pre-synthesis constraints and estimates of delays.
- Map timing analysis is based on post-synthesis constraints, the actual types of components, and estimates of the routing delays.
- Place-and-route timing analysis is based on post-synthesis constraints, and the actual components and routing.

All the reports can be read in the Reports tab. The place-and-route timing analysis can also be viewed in the Timing Analyzer tool. Timing Analyzer gives you a spreadsheet view that you might find easier to read. Timing Analyzer also has a search function to help you find different data paths.

Reading the Timing Analysis Report

The timing analysis report has several sections to explore.

To examine the timing analysis report:

1. In the Reports tab, click **Place & Route Reports** and then click **Place & Route Timing Analysis**.

   The Timing Report appears.

   If the frame for the report is too small, you can enlarge it by clicking the Detach Tool button that is at the top-right corner of the Tools Area. This creates a separate window for Reports.

2. Hover over the **Contents** button, in the top-right corner of the report.

   A list of the report's section headings appears. You can use these links to jump to any section of the report. You can make the contents disappear by clicking anywhere in the report. You can also jump back to the top of the report by clicking the scroll-up button in the bottom-right corner of the report.

3. Click **1 DESIGN CHECKING**.

   "Design Checking" shows the constraints and operating conditions that guided the analysis. It also shows a list of combinational loops that could not be analyzed.

   Notice that you have one create_clock constraint (if you are using the LSE synthesis tool). This constraint was created by LSE to specify the high-frequency output of the FPGA oscillator.

4. Go to **2 CLOCK SUMMARY**. Either scroll down or click in the Contents.

   "Clock Summary" shows an analysis for each clock domain defined in the constraints. The "Clock Domain Crossing" section lists any other clocks that connect with the given domain. That is, a data path that has different clocks for its start and end points.

Notice that the target frequency for the oclk_N domain is 225 MHz, the oscillator's default speed. But the "Actual" frequency, which is the calculated maximum frequency, is much slower.

5. Go to **3 TIMING ANALYSIS SUMMARY**.

   "Timing Analysis Summary" shows a variety of data including lists of the ten worst paths for setup slack and for hold slack, unconstrained timing start and end points, unconstrained I/O ports, and registers without clocks.

6. Go to **4 DETAILED REPORT**.

   "Detailed Report" shows details of the ten worst paths for setup slack and for hold slack. Each path has a section that starts with information about the whole path. This is followed by a table calculating the delay step by step through the path, beginning with the clock at the start of the path and ending with the clock at the end of the path. Each step includes the name of the pin, the site within the FPGA, and the hierarchical name of the port.

   If you want to visualize the path, the report has links to other tools. Physical Designer Placement Mode shows the sites within the FPGA. Physical Designer Routing Mode shows the route within the FPGA. (Physical Designer is only available after place-and-route.) Netlist Analyzer shows a schematic view of the design. Unfortunately, Netlist Analyzer will often say that it "Can't show the schematic of this timing path."

7. Take some time to study the failing paths in section 4.1, "Setup Detailed Report."

   Note that the hierarchical names keep referring to "reveal_coretop." This should not be surprising because the timing errors only appeared after adding the Reveal cores.

It looks like parts of the Reveal cores are just too slow for the default oscillator speed.

Before leaving this task, take a look at the Timing Analyzer tool.

Using Timing Analyzer

Timing Analyzer is a different way to look at the place-and-route timing analysis that you might find easier to read. Timing Analyzer runs the timing analysis and presents the results on three spreadsheet tabs. Plus, there is a Query tab so you can search through the paths. The information in Timing Analyzer is very similar to that in the Place & Route Timing Analysis report but is presented differently.

Timing Analyzer can be run anytime after completing the place-and-route process. You do not need to select timing analysis in the Task Detail View of the Process Toolbar.

To use Timing Analyzer:

1. Choose **Tools >** **Timing Analyzer**.

A progress indicator opens, showing that the Radiant software is calculating the delays. This takes a moment. Then Timing Analyzer appears in the Tool Area. The General Information tab is just basic information about the FPGA and the option settings used in the analysis. Tabs for the actual analysis are along the bottom.

2. Click the **Critical Paths Summary** tab.

   This tab shows the same information as section 4, “Detailed Report,” of the text report. At first you just see introductory information for the paths.

3. Click on a row to see the rest of the information.

   The window splits into three parts. You might want to enlarge the view by detaching the tool as a separate window.

   The Path Detail part shows the same the introduction to the path seen in the text report. There are also some delay calculations for the destination and source clocks.

   The third part has two tabs for the table calculating the delay step by step through the path. Data Path shows the steps. Clock Paths shows the clocks at the start and end of the path.

   To link to Physical Designer Placement Mode or Routing Mode, right-click any row in the Data Path or Clock Paths tabs.

4. Click the **Critical Endpoint Summary** tab.

   This tab shows the same information as section 3.2, “Setup Summary Report,” and section 3.2, “Hold Summary Report,” of the text report. Click on a row to see the same path details as in the Critical Paths Summary tab.

5. Click the **Unconstrained Endpoint Summary** tab.

   This tab shows the same information as section 3.4, “Unconstrained Report,” of the text report.

6. Click the **Query** tab.

   This tab shows a query form to search for data paths. After each search, check the Output view to see if anything was found. Any paths found are shown in a spreadsheet view at the bottom of the form. Again, you might want to enlarge the view by detaching the tool as a separate window. Click on a row to see the same path details as in the Critical Paths Summary tab.

7. Close Timing Analyzer.

Task 10: Set Timing Constraints

The original design had no timing problems but with the addition of the Reveal Debug module there are problems. You need to redefine the design’s oscillator clock with a longer period.

Radiant uses standard Synopsys SDC timing constraints. These constraints can be created with Source Editor or Timing Constraints Editor. Timing Constraints Editor helps you find the correct signals and makes sure the syntax is correct. But Timing Constraints Editor can only be used with Lattice Synthesis Engine (LSE).

Note

If you are using Synplify Pro for Lattice, create the constraints using SCOPE or type them with Source Editor. See Figure 12 on page 42 for the finished timing constraints.

Timing Constraints Editor comes in pre- and post-synthesis versions. These work the same but produce different files: pre-synthesis produces an .ldc file that LSE reads and post-synthesis produces a .pdc file that the map process reads. In large designs, you might use post-synthesis to avoid rerunning synthesis.

Defining the Oscillator Clock

This “create_clock” constraint redefines the HFCLKOUT pin of the OSCA module as a clock with a period of 8 ns. The name is specified as oclk_N.

To define the oscillator clock:

1. Choose **Tools >** **Timing Constraints Editor > Post-Synthesis Timing Constraint Editor**.

   The Post-Synthesis Timing Constraint Editor appears. The top half is a spreadsheet with a row of tabs beneath it. The bottom half is a box where constraint text appears.

   Each of the tabs creates a different kind of constraint. But all the tabs work in the same way: fill in the cells along a row. For columns such as Clock or Port, double-click in the cell and launch the Object Edit dialog box. The dialog box helps you find objects. The Editor creates the constraint as you go.

2. Click the Clock tab.

   The Editor already has a create_clock constraint. This was created in synthesis. You can ignore it.

3. Double-click in the empty cell in the second row in the Object Clock column.

   Some text appears in the cell and, at the right side, three periods.

4. Ignore the text and click on the three periods.

   The Object Edit dialog box opens.

5. From the Object Type menu, choose **CLOCKPIN**.

6. In the Filter box, under the Available Objects list, type **H**.

   The list is reduced.

7. Select: osc_inst.OSCA_inst/**HFCLKOUT**.

8. Click **OK**.

The dialog box closes. A get_pins command appears in the Object Clock column.

9. Click in the Clock Name column and type **oclk_N**. This is an alias to use instead of the long pathname.

10. Click in the **Period** column and enter **8**.

    The Frequency column is filled in. In the lower half of the editor, a create_clock constraint appears.

    The lower half of the editor should look like Figure 12.

Figure 12: Timing Constraints

(Auto) create_clock -name {oclk_N} -period 4.444 [get_pins {osc_inst.OSCA_inst/HFCLKOUT}]

create_clock -name {oclk_N} -period 8 [get_pins osc_inst.OSCA_inst/HFCLKOUT]

You now have two create_clock constraints for HFCLKOUT. That's OK. The constraint that you created will override the constraint from synthesis.

11. Click the Save button in the toolbar.

    The timing constraints are added to the existing .pdc file. The map and place & route processes are reset, and need to be run again.

12. In the Process Toolbar, click the Run All button.

    The export process finishes with no error messages.

13. Go to the Reports tab.

    The Project Summary report shows **no** timing errors.

14. Close the Post-Synthesis Timing Constraint Editor.

Close the Radiant Project

If this were a real project, you would now program the FPGA on your prototype board. Then you would start Reveal Analyzer/Controller to study the internal operation of your design in detail.

But without a board, the tutorial ends here. You can close the project and exit the Radiant software.

To gain more skill with the Radiant software, study the online help (**Help > Lattice Radiant Software Help**). And begin work on your own project!

To close the project:

1. Choose **File > Close Project**.

   The Save Modified Files dialog box opens.

2. Select the files that you want to save.

3. Click **OK**.

The design project and associated tools close. The Radiant window returns to the Start Page.

4. You can continue to work with the Radiant software or exit by choosing **File > Exit**.

Summary of Accomplishments

You have completed the *Lattice Radiant 2.2 Tutorial with CrossLink-NX (LIFCL)*. In this tutorial, you have learned how to:

- Create a new Radiant project.
- Customize IP using IP Catalog.
- Verify functionality with simulation.
- Set timing and location assignments.
- Process the design.
- Analyze power consumption.
- Analyze static timing.
- Use Reveal Inserter to add on-chip debug logic.

Recommended References

You can find additional information on the subjects covered by this tutorial in the Radiant software online Help and in the *Lattice Radiant Software User Guide*.

Revision History

The following table gives the revision history for this document.

Date	Version	Description
10/20/2020	2.2	Updated for Radiant 2.2. Rewrote “Verify Functionality with Simulation” task to use Mentor ModelSim instead of Aldec Active-HDL.
6/2/2020	2.1	Updated for Radiant 2.1. Added section for PMI and Source Template. Removed use of CrossLink-NX Evaluation Board until an updated board is available.
2/25/2020	2.0.1	Modified to include use of the CrossLink-NX Evaluation Board. Added sections for programming the FPGA, and setting up and running on-chip debug.
12/17/2019	2.0	Added a link for downloading the design files. Expanded Task 1 with more information about the main window and the File List view.
11/12/2019	2.0	Initial Release.