

Lattice Radiant Software Tutorial with CrossLink-NX



February 25, 2020

Copyright

Copyright © 2020 Lattice Semiconductor Corporation. All rights reserved. This document may not, in whole or part, be reproduced, modified, distributed, or publicly displayed without prior written consent from Lattice Semiconductor Corporation (“Lattice”).

Trademarks

All Lattice trademarks are as listed at www.latticesemi.com/legal. Synopsys and Synplify Pro are trademarks of Synopsys, Inc. Aldec and Active-HDL are trademarks of Aldec, Inc. All other trademarks are the property of their respective owners.

Disclaimers

NO WARRANTIES: THE INFORMATION PROVIDED IN THIS DOCUMENT IS “AS IS” WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OF ANY KIND INCLUDING WARRANTIES OF ACCURACY, COMPLETENESS, MERCHANTABILITY, NONINFRINGEMENT OF INTELLECTUAL PROPERTY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL LATTICE OR ITS SUPPLIERS BE LIABLE FOR ANY DAMAGES WHATSOEVER (WHETHER DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION, OR LOSS OF INFORMATION) ARISING OUT OF THE USE OF OR INABILITY TO USE THE INFORMATION PROVIDED IN THIS DOCUMENT, EVEN IF LATTICE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME JURISDICTIONS PROHIBIT THE EXCLUSION OR LIMITATION OF CERTAIN LIABILITY, SOME OF THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU.

Lattice may make changes to these materials, specifications, or information, or to the products described herein, at any time without notice. Lattice makes no commitment to update this documentation. Lattice reserves the right to discontinue any product or service without notice and assumes no obligation to correct any errors contained herein or to advise any user of this document of any correction if such be made. Lattice recommends its customers obtain the latest version of the relevant information to establish that the information being relied upon is current and before ordering any products.

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.
<i><Italic></i>	Variables in commands, code syntax, and path names.
Ctrl+L	Press the two keys at the same time.
<i>Courier</i>	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 Software Tutorial with CrossLink-NX	7
About the Tutorial	7
Task 1: Create a New Radiant Project	8
Opening the New Project Wizard	9
Setting the Project Name and Location	9
Adding Source Files	10
Selecting a Device	11
Finishing the Project Setup	12
About the File List View	12
Task 2: Add HDL Code	14
Generating a Module from IP Catalog	14
Instantiating the Module	15
Task 3: Verify Functionality with Simulation	16
Starting a Simulation Run	17
Checking the Simulation Results	18
Task 4: Set Location Assignments	19
Task 5: Process the Design	21
About the Process Toolbar	21
Processing the Design	22
Task 6: Examine the Layout	22
Task 7: Examine Timing Analysis Results	24
Task 8: Analyze Power Consumption	25
Task 9: Add an On-Chip Debug Module	26
About the Logic Analyzer Core	27
Setting Up Trace Signals	28
Setting Up Trace Options	29
Setting Up Trigger Units	30
Setting Up a Trigger Expression	31
Setting Up Trigger Options	32
Creating Virtual Switches and LEDs	32
Creating User Register Access	34

Creating Hard IP Access	34
Inserting the Debug Logic	35
Task 10: Set Timing Constraints	35
Defining the TCK Clock	36
Defining the Oscillator Clock	37
Specifying the TCK/oclk Paths	37
Task 11: Programming the FPGA	38
Generating the Bitstream	38
Downloading the Bitstream	39
Task 12: Perform Logic Analysis	41
Creating a Reveal Analyzer Project	41
Changing the Trigger Event	42
Running the Logic Analyzer Core	43
Using the Virtual Switches and LEDs	44
Accessing the User Register	45
Accessing the Hard IP	46
Close the Radiant Project	46
Summary of Accomplishments	47
Recommended References	47
Revision History	49

Lattice Radiant Software Tutorial with CrossLink-NX

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™ device family.

Note

Some of the screen captures in this tutorial may have been taken from a version of 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.
- ▶ Examine static timing analysis.
- ▶ Analyze power consumption.
- ▶ Download a bitstream to an FPGA.
- ▶ Create on-chip debug logic.
- ▶ Perform logic analysis.

Time to Complete About 2½ 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 46. 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 2.0.1 (2.0 with Service Pack 1)
- ▶ (Optional) CrossLink-NX Evaluation Board to download a bitstream and to do on-chip debugging. If you do not have the board, you can still do most of the tutorial.

For more information on the CrossLink-NX Evaluation Board, go to:
<http://www.latticesemi.com/en/Products/DevelopmentBoardsAndKits/CrossLink-NXEvaluationBoard>

Online Help You can find online help 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 while displaying the values on the demo LEDs of the CrossLink-NX Evaluation Board. There are also a couple of 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.

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:

```
<install_path>/bin/linux64/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 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
- ▶ dphy1_s.v
- ▶ pmi_mem_32.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. We'll specify the FPGA on the CrossLink-NX Evaluation Board.

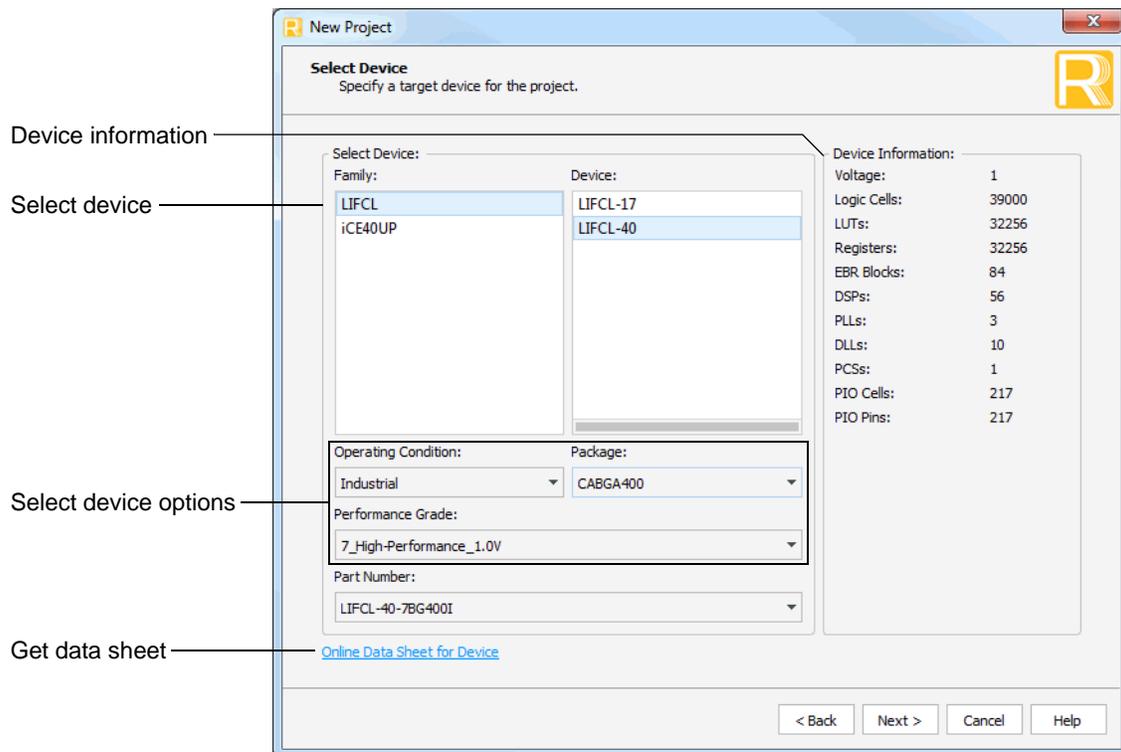
To select a device:

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**.

Note

If you choose to use Synplify Pro[®], some of the Radiant tools, such as Timing Constraint Editor and Physical View, 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.

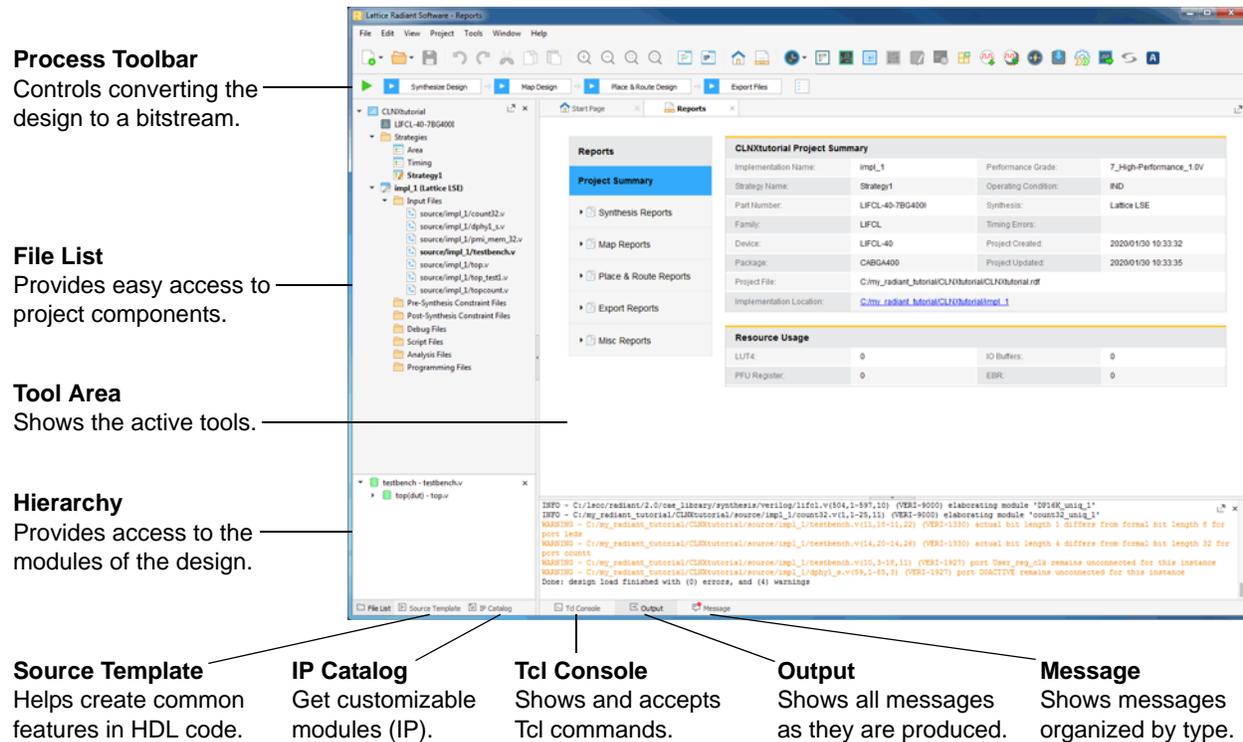
You will see activity in the Output view, at the bottom of the window, as the Radiant software re-analyzes the design hierarchy. In the File List view, **top.v** is bolded to show that it holds the top module. The Hierarchy view, which is underneath the File List view, also changes.

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.

Figure 2: Radiant Main Window



- ▶ 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.

Generating a Module from IP Catalog

In this section, you will customize and generate an oscillator (OSC) module to add to the design.

To customize and generate an OSC 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 is available to you.

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

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.

4. Double-click **OSC**.

The Module/IP Block Wizard opens.

5. For Component name, enter **my_osc**. 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. But we will keep it simple and just use the defaults.
7. Click **Generate**.
The Check Generating Result page appears. This may take a moment.
8. Ensure that **Insert to project**, in the lower-left corner, is selected and click **Finish**.
9. Go back to the File List view to see that `my_osc/my_osc.ipx` has been added to the list of Input Files. The module comes with a few associated files. In the Hierarchy view, a `my_osc` 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 OSC 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_osc here. ***`
3. In the File List view, right-click **my_osc.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. (In this case, the input port will be tied to one.) See below for the finished instantiation command. Bold is the text that you enter.

```
my_osc osc_inst(.hf_out_en_i(1'b1),
               .hf_clk_out_o(oclk));
```

6. Click the Save  button in the toolbar.
In the Hierarchy view, the `my_osc` module moves to be under the top module.
7. Close Source Editor and the IP Information page by clicking the X in their tabs.

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 Aldec[®] Active-HDL[™] Lattice 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. See Table 1. You can use other simulators outside of the Radiant software.

Table 1: HDL Simulators Integrated with the Radiant Software

Simulators	Windows	Red Hat Linux	Ubuntu Linux
Aldec [®] Active-HDL [™] LE	Integrated and included.	Not available.	Not available.
Mentor [®] ModelSim [®]	Integrated; get separately.	Integrated; get separately.	Use another simulator separately.
Mentor Questa [®]	Integrated; get separately.	Integrated; get separately.	

If you are not using the Active-HDL 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 a moment.
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.

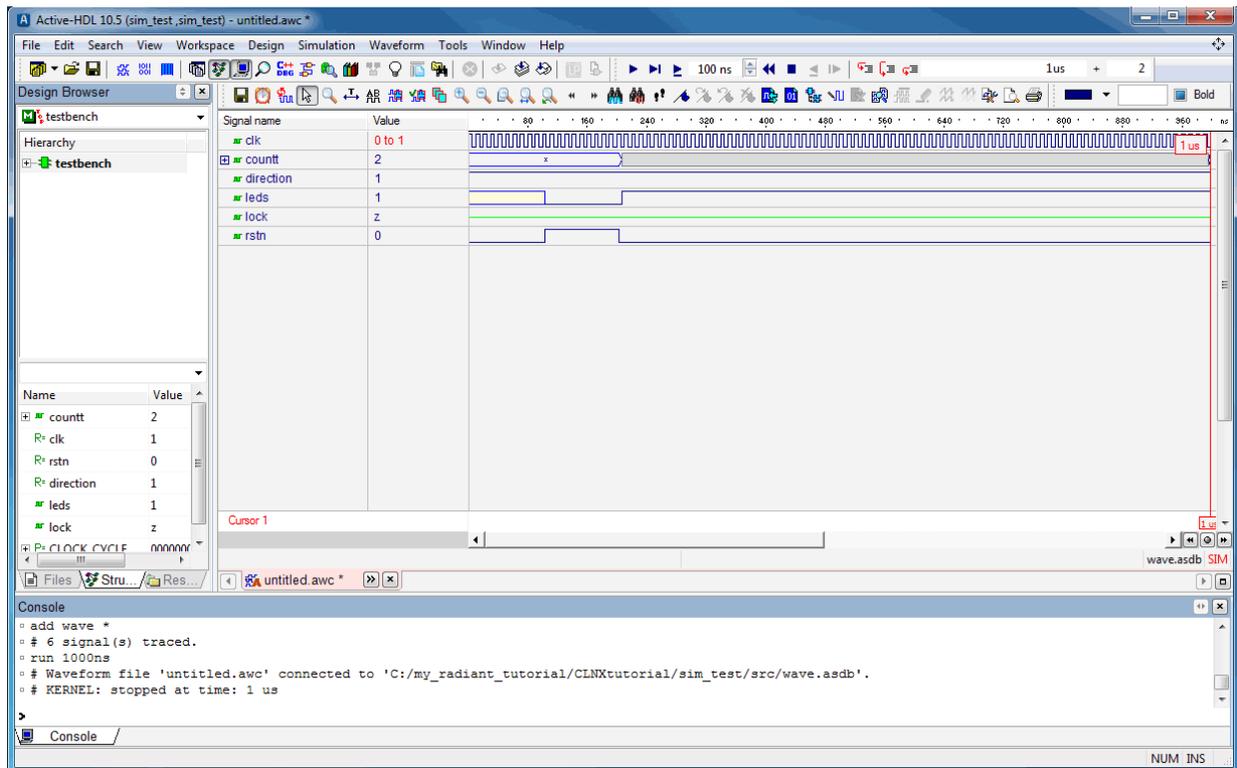
Checking the Simulation Results

Note

If you are not using Active-HDL LE, you can skip this section.

Now that you've run the simulation, you can see what happened on the top-level signals of the testbench as shown in [Figure 3](#). Active-HDL stopped automatically after the first microsecond of simulation time. The testbench is set to run longer, over 50 μ s, but this is enough to see the startup.

Figure 3: Simulated Waveform



To check the simulation results:

1. The values of countt may not be visible. Click the Zooms In View  button in the toolbar until you can see the values.
2. We can add more signals. In the Hierarchy view, expand:
testbench > dut : top > top_test1_inst : top_test1 > counter : topcount
3. Click on **counter : count32**.
The list of signals changes.
4. Drag **countai** to the Signal Name column of the waveform view.

5. Rerun the simulation to see what is happening with the countai register. Choose **Simulation > Restart Simulation** and then **Simulation > Run**. Now you get the full 50 μ s.

You see the rstn signal activated by the testbench. This drives the leds and countai values to zero. After rstn is released, countt and countai start counting.

6. Click anywhere to see what the values are at that moment.
7. Choose **File > Exit** to close Active-HDL.
The Design Browser dialog box appears.
8. Click **OK** to stop the simulation.
The Save File dialog box appears.
9. Click **No**.

There's a lot more that you can do with Active-HDL. For more information, choose **Help > Product Help** in the Active-HDL window.

Task 4: Set Location Assignments

You will use the Device Constraint Editor to assign pins for the board's Demo LEDs and GSRN button. 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 from the board's user guide, typing is probably the easiest way.

To set 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:
 - ▶ 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.

Figure 4: 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 several steps that convert the high-level Verilog and VHDL description into code that can actually program a specific FPGA:

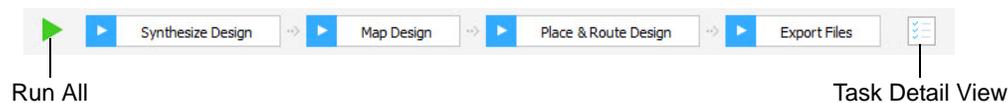
1. Synthesis converts HDL into a gate-level netlist that is optimized for the FPGA.
2. Map converts the netlist into a network of device-specific components, such as PFU (programmable function units) and I/O buffers.
3. Place and route converts the mapped network into specific components and signal routes 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 5: 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.

Task 6: Examine the Layout

After routing, you can see a display of the layout using the Physical View and Floorplan View tools and cross-probing between the two views.

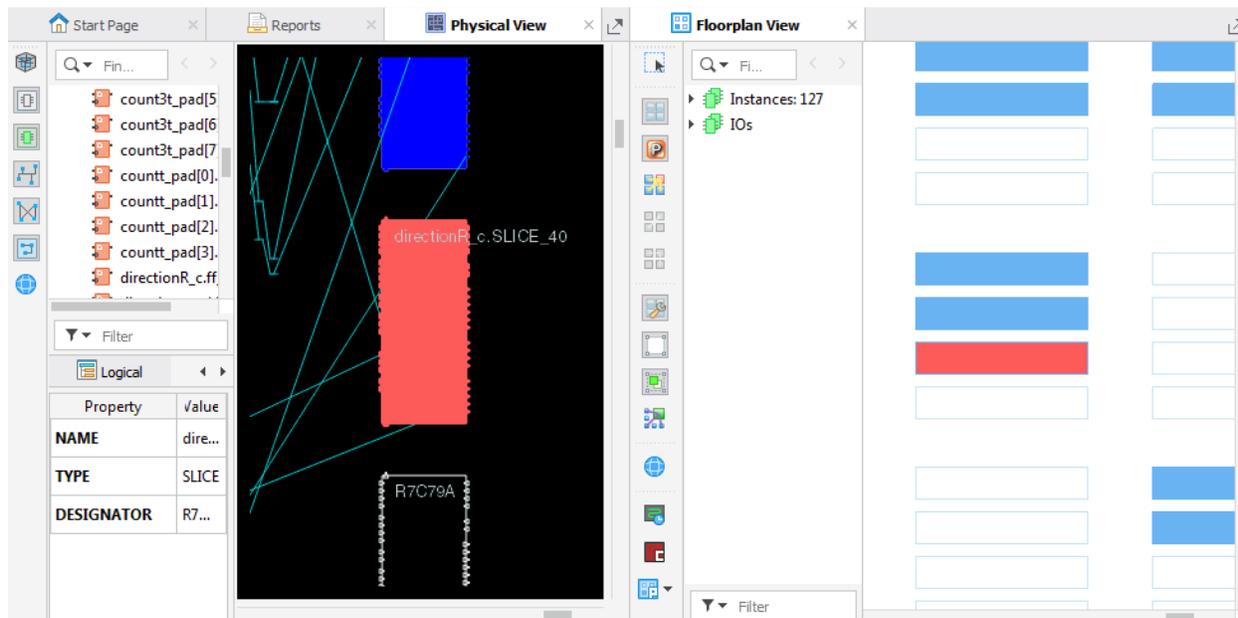
To see the layout:

1. Choose **Tools** >  **Physical View**.
Physical View shows a detailed layout of your design that includes switch boxes and physical wire connections.
2. To the left of the diagram are lists of ports, instances, and nets. Expand the Instances list and choose one of the primitives, such as Instances > top_test1_inst > Primitives > **leds_i8.ff_inst**.
The display zooms to the component.
3. Right-click on the component and choose **Show in > Floorplan View**.
Floorplan View opens with the display zoomed to the same component. Floorplan View provides a large-component layout of your design.
4. To easily see cross-probing between Floorplan View and Physical View, ensure both views are attached to the Radiant main window. Right-click on the Floorplan View tab and choose **Split Tab Group**.

The pane splits to show both views. See [Figure 6](#). If you lost sight of the selected component, click **leds_i8.ff_inst** in the list again.

When both Floorplan View and Physical View are open, an item that you select in one of these views is automatically selected in the other. Cross-probing is especially useful for immediate examination of connections in both views.

Figure 6: Cross-Probing



- Right-click on the Floorplan View tab and choose **Move to Another Tab Group**.

Floorplan View fills the whole pane as before.

- In the toolbar of Floorplan View, click the arrow of the Placement  button and choose  **IO**.

Floorplan View changes to show the I/O of the device.

- In the list, expand Instances, scroll down to the bottom, and click **rstn_pad.bb_inst**.

Floorplan View zooms in to the I/O for rstn: G19, which you set in the constraint file. The padlock symbol shows the I/O 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.

- Close Floorplan View and Physical View.

Task 7: Examine Timing Analysis Results

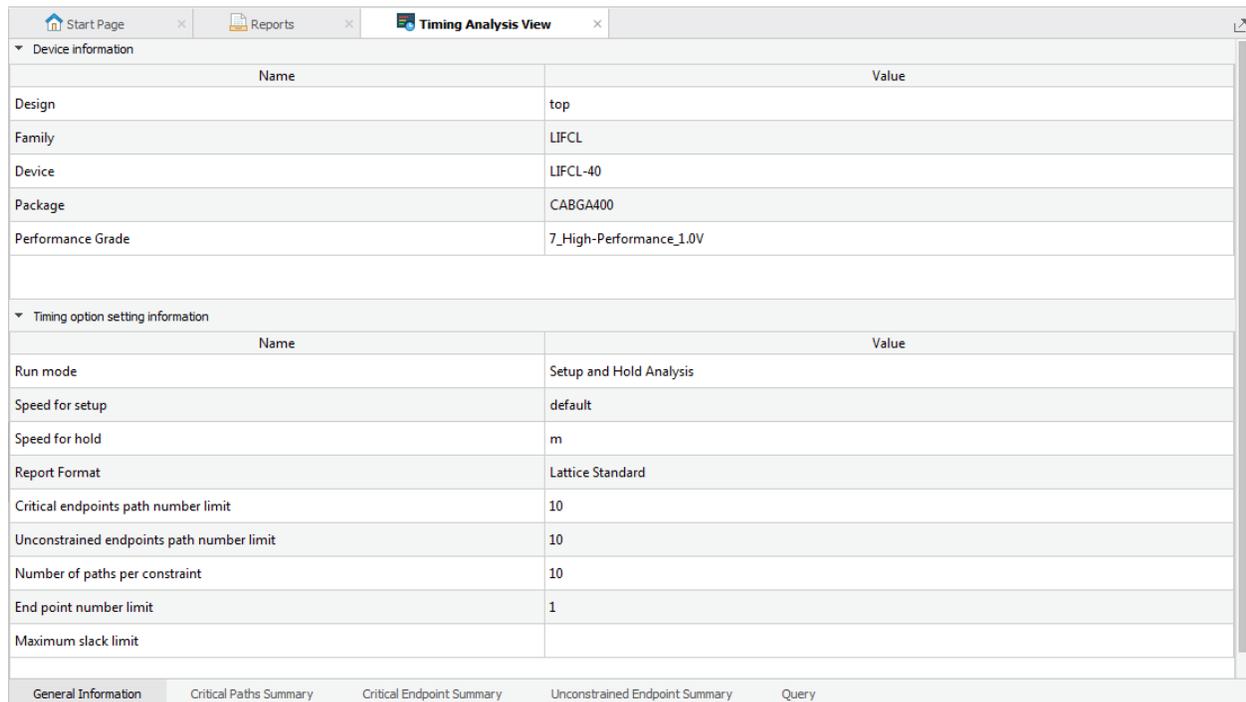
Static timing analysis can determine if your circuit design meets timing constraints. Rather than simulation, it employs conservative modeling of gate and interconnect delays that reflect different ranges of operating conditions on various dies, providing complete verification coverage.

To examine timing analysis results:

1. Choose **Tools** >  **Timing Analysis View**.

Timing Analysis View appears as shown in [Figure 7](#).

Figure 7: Timing Analysis View



Device information	
Name	Value
Design	top
Family	LIFCL
Device	LIFCL-40
Package	CABGA400
Performance Grade	7_High-Performance_1.0V
Timing option setting information	
Name	Value
Run mode	Setup and Hold Analysis
Speed for setup	default
Speed for hold	m
Report Format	Lattice Standard
Critical endpoints path number limit	10
Unconstrained endpoints path number limit	10
Number of paths per constraint	10
End point number limit	1
Maximum slack limit	
General Information Critical Paths Summary Critical Endpoint Summary Unconstrained Endpoint Summary Query	

2. Click the **Critical Paths Summary** tab (on the lower-left of Timing Analysis View).

A spreadsheet appears with a list of critical paths.

3. Select row **1**.

The window splits to show more information.

4. Close Timing Analysis View.

Task 8: 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

The screenshot shows the Power Calculator application window with the following sections:

- Device:** Family: LIFCL, Performance grade: 7_High-Performance_1.0V, Device: LIFCL-40, Operating conditions: Industrial, Package type: QFN72, Part Number: LIFCL-40-7SG72I.
- Environment:** Thermal Profile..., Ambient Temperature(°C): 25, Effective Theta-JA(°C/W): 8.36, Junction Temperature(°C): 25.14, Maximum Safe Ambient(°C): 99.44.
- Device Power Parameters:** Process Type: Typical, Power File Revision: Preliminary.
- Voltage/Dynamic Power Multiplier:**

	Voltage	DPM
Vcc	1.000	1.00
Vccpg	1.000	1.00
Vccaux	1.800	1.00
Vccauxa	1.800	1.00
Vccauxh	1.800	1.00
Vccauxhi	1.800	1.00
Vccauxho	1.800	1.00
Vccio 3.3	3.300	1.00
Vccio 2.5	2.500	1.00
Vccio 1.8	1.800	1.00
Vccio 1.35	1.350	1.00
Vccio 1.5	1.500	1.00
Vccio 1.2	1.200	1.00
Vccio 1.0	1.000	1.00
Vccapll	0.900	1.00
- Current by Power Supply:**

	Static (A)	Dynamic (A)	Total (A)
Vcc	0.003841	0.000000	0.003841
Vccpg	0.002678	0.000000	0.002678
Vccaux	0.002824	0.000000	0.002824
Vccauxa	0.000000	0.000000	0.000000
Vccauxh	0.000000	0.000000	0.000000
Vccauxhi	0.000061	0.000000	0.000061
Vccauxho	0.000185	0.000000	0.000185
Vccio 3.3	0.000000	0.000000	0.000000
Vccio 2.5	0.000000	0.000000	0.000000
Vccio 1.8	0.002219	0.000000	0.002219
Vccio 1.35	0.000000	0.000000	0.000000
Vccio 1.5	0.000000	0.000000	0.000000
Vccio 1.2	0.000000	0.000000	0.000000
Vccio 1.0	0.000000	0.000000	0.000000
Vccapll	0.000304	0.000000	0.000304
- Power by Power Supply:**

	Static (W)	Dynamic (W)	total (W)
Vcc	0.003841	0.000000	0.003841
Vccpg	0.002678	0.000000	0.002678
Vccaux	0.005084	0.000000	0.005084
Vccauxa	0.000000	0.000000	0.000000
Vccauxh	0.000000	0.000000	0.000000
Vccauxhi	0.000111	0.000000	0.000111
Vccauxho	0.000334	0.000000	0.000334
Vccio 3.3	0.000000	0.000000	0.000000
Vccio 2.5	0.000000	0.000000	0.000000
Vccio 1.8	0.003995	0.000000	0.003995
Vccio 1.35	0.000000	0.000000	0.000000
Vccio 1.5	0.000000	0.000000	0.000000
Vccio 1.2	0.000000	0.000000	0.000000
Vccio 1.0	0.000000	0.000000	0.000000
Vccapll	0.000273	0.000000	0.000273
- Power by Block (W):**

Block	Power (W)
Logic Block	0.002197
Clocks	0.000004
I/O	0.006314
I/O Term	0.000000
DSP	0.000177
PLL	0.000589
Block RAM	0.000019
LRAM	0.000006
SGMIICDR	0.000036
DDRDL	0.000224
DLDEL	0.000002
DQS	0.000068
MIPIDPHY	0.000436
ADC	0.000068
ALU	0.000002

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 9: 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**.

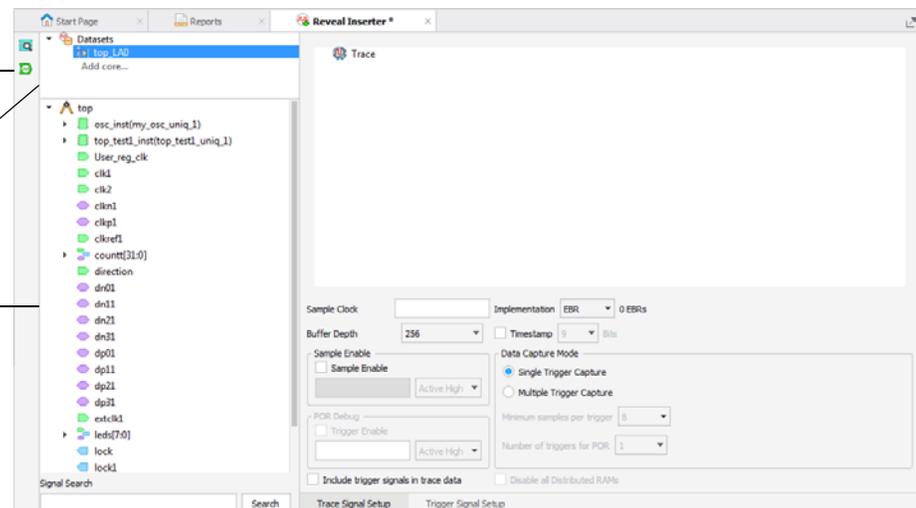
The Trace Signal Setup tab appears. The Dataset View expands to include a core named top_LA0. See [Figure 9](#).

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 has also been labeled with “@Tc” to show that it is a trace signal.
6. Right-click the 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 has also been 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.

- 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.
- 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 while running tests.

To set up the trigger units:

- Click on the **Trigger Signal Setup** tab.
- In the Trigger Unit section, at the top, click **Add**.
A new row appears with default values.
- Click in the Name cell and enter **tu_countai**.
- 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.
- 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.
- 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.

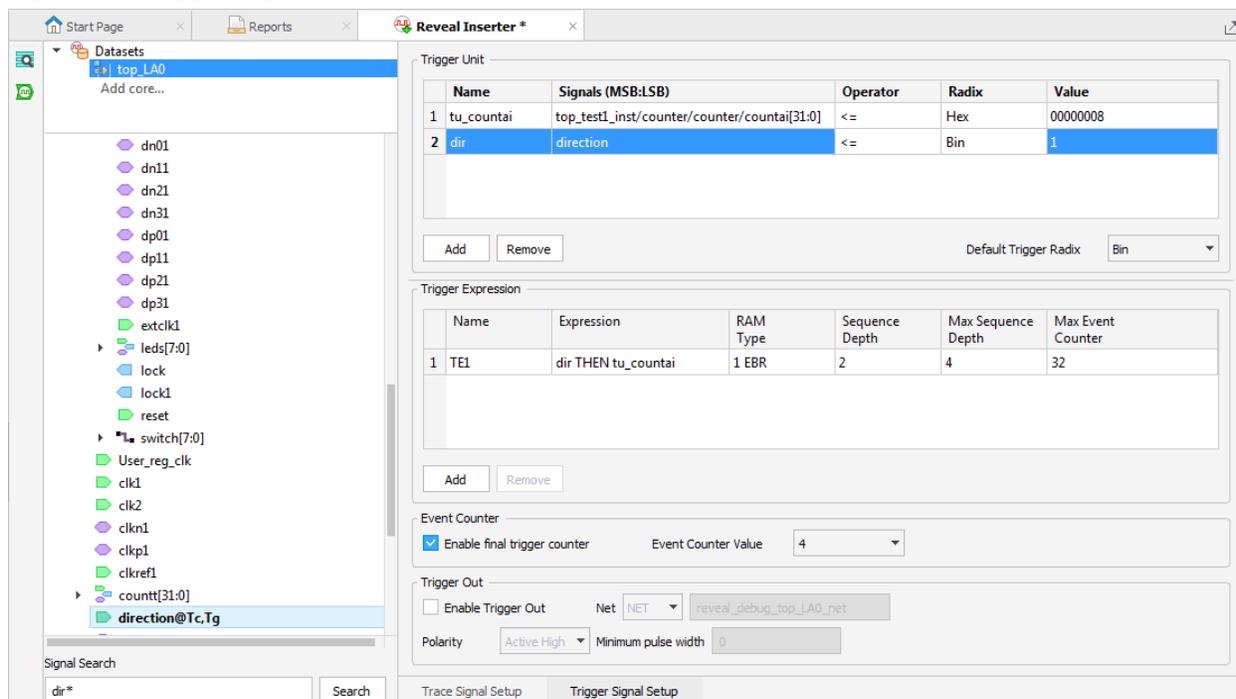
The Trigger Signal Setup tab should now resemble [Figure 11](#).

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. You may need to scroll to see it fully.

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_dp` 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 `pmi_ram_dp_inst`. Expand it.
3. Fill the User Register Setup tab by dragging the matching signals from the `pmi_ram_dp_inst` module:
 - ▶ Clock: **Clock**
 - ▶ Clock_enable: **ClockEn**
 - ▶ Wr_Rdn: **WE_RDN**
 - ▶ Address: **Address[8:0]**
 - ▶ WData: **WData[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 DPHY1
2. In the Enabled column, select **DPHY1**.

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.

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 task.

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 set up constraints blocking timing analysis on paths between Reveal's JTAG clock and the design's oscillator clock. These clocks are not related. You'll start by defining the two clocks and then specify the paths between them.

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 38](#) 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 .lde file that LSE reads and post-synthesis produces a .pdc file that the map process reads. The tutorial uses post-synthesis because the clocks are not readily available before synthesis. In large designs, you might use post-synthesis to avoid rerunning synthesis.

Defining the TCK Clock

This “create_clock” constraint defines the TCK port as a clock with a period of 100 ns. The name defaults to TCK.

To define the TCK clock:

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

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

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 **CLOCKPORT**.

6. Under Available Objects, select **TCK**.

In the Object Command box, “get_ports TCK” appears.

7. Click **OK**.

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

8. Click in the **Period** column and enter **100**.

Other columns of the row are filled in. In the lower half of the editor, a create_clock constraint appears.

Defining the Oscillator Clock

This “create_clock” constraint defines the HFCLKOUT pin of the osc_inst module as a clock with a period of 40 ns. The name is specified as oclk.

To define the oscillator clock:

1. Double-click in the empty cell in the third row in the Object Clock column and click on the three periods.

The Object Edit dialog box opens.

2. From the Object Type menu, choose **CLOCKPIN**.
3. In the Filter box, under the Available Objects list, type **H**.

The list is reduced.

4. Select: **.../HFCLKOUT**.
5. Click **OK**.

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

6. Click in the Clock Name column and type **oclk**. This is an alias to use instead of the long pathname.
7. Click in the **Period** column and enter **40**.

Other columns of the row are filled in. In the lower half of the editor, a create_clock constraint appears.

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

Specifying the TCK/oclk Paths

We can block timing analysis of the paths between the two clocks with the set_clock_groups constraint. This constraint specifies that the clocks are mutually exclusive.

To block the TCK/oclk paths from timing analysis:

1. Click the Clock Group tab.
2. Double-click in the Group cell and click on the three periods.
3. In the Object Edit dialog box, select **oclk** and **TCK**.
4. Click **OK**.
5. Click anywhere in the window.

A new Group column appears and the other cells get check boxes and red Xs. The red Xs just mean that the constraint is not finished. It needs a reason why the clocks are not related.

6. Select **Logically Exclusive**.

The red Xs disappear.

The lower half of the editor should look like [Figure 12](#).

Figure 12: Timing Constraints

```
(Pre Synthesis) create_clock -name {osc_inst_hf_clk_out_o} -period 4.444 [get_pins osc_inst/lsccl_osc_inst/
u_OSC.OSCA_inst/HFCLKOUT]
```

```
create_clock -name {TCK} -period 100 [get_ports TCK]
```

```
create_clock -name {oclk} -period 40 [get_pins osc_inst/lsccl_osc_inst/u_OSC.OSCA_inst/HFCLKOUT]
```

```
set_clock_groups -group [get_clocks {oclk TCK}] -logically_exclusive
```

7. 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.

8. Close the Post-Synthesis Timing Constraint Editor.

Task 11: Programming the FPGA

Use the Process Toolbar to generate files for exporting. One of the files exported will be a bitstream file (.bit) that can be used to program an actual CrossLink-NX device on a circuit board.

Note

The rest of the tutorial requires the CrossLink-NX Evaluation Board. If you do not have the board, you can stop the tutorial now. Go to [“Close the Radiant Project” on page 46](#).

Generating the Bitstream

The final step in the Process Toolbar is Export Files. This generates the bitstream file used to program the FPGA.

To generate files for export:

1. In the Process Toolbar, click **Export Files**.

The Radiant software generates the bitstream file and saves it in the directory of the implementation.

2. In the Reports view, check that the timing errors are gone.
3. Click **Export Reports** and examine the available reports.

At the end of the Bitstream report is the pathname of the bitstream file.

4. In the File List view, right-click on impl_1 and choose **Open Containing Folder**.

A window opens showing the contents of the impl_1 folder.

5. Look for a file named **CLNXtutorial_impl_1.bit**.
6. Close the impl_1 folder window.

Downloading the Bitstream

This task requires that you have a CrossLink-NX Evaluation Board. In this section, you will use the Radiant Programmer to download a bitstream to a CrossLink-NX FPGA.

To download the bitstream to the FPGA on the board:

1. Connect the USB cable from your computer to the CrossLink-NX Evaluation Board. Give the computer a few seconds to detect the USB device.

For more information, refer to the [CrossLink-NX Evaluation Board User Guide](#).

2. Choose **Tools** >  **Programmer**.

The Radiant Programmer opens in a separate window.

3. In the Cable Setup box, click **Detect Cable**.

Radiant detects the cables and Lattice cable drivers that are installed.

If more than one cable is found, the Programmer: Multiple Cables Detected dialog box opens. Choose the cable that says “Lattice CrossLink-NX Eval Board.”

Under the Detect Cable button, you should see:

- ▶ Cable: HW-USBN-2B (FTDI)

The board uses an FTDI USB2-type of cable.

- ▶ Port: FTUSB-*<number>*

Ports for this type of cable are labeled FTUSB-*<number>*. The number is assigned based on the USB port address.

4. Choose **Run** >  **Scan Device**.

A progress bar appears while Programmer scans the board. This will take a moment.

When the scan is done, the spreadsheet view changes to show “LIFCL” and “LIFCL-40.” Also, the spreadsheet view splits to show a diagram of the connection between your computer and the FPGA. You may have to expand the Programmer window to see the whole diagram.

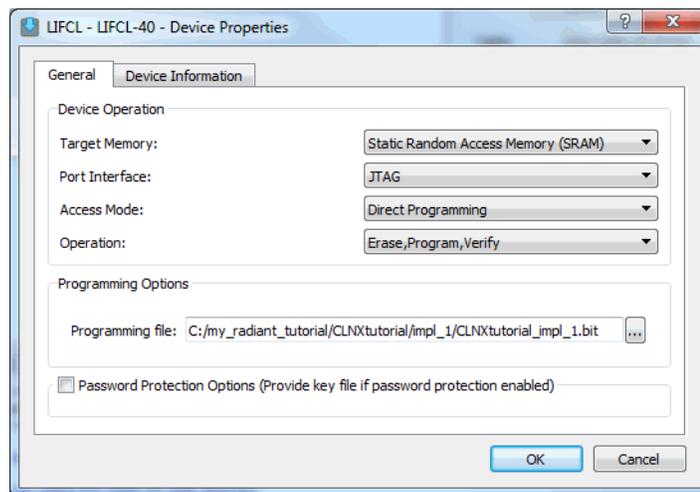
5. Click on row 1 in the spreadsheet and choose **Edit** >  **Device Properties**.

The Device Properties dialog box opens.

6. Ensure the settings are as follows:
 - ▶ Target Memory: **Static Random Access Memory (SRAM)**
 - ▶ Port Interface: **JTAG**
 - ▶ Access Mode: **Direct Programming**
 - ▶ Operation: **Erase,Program,Verify**
 - ▶ Programming file: `<project_path>/impl_1/CLNXtutorial_impl_1.bit`
 - ▶ Password Protection Options: cleared (*not* selected)

The Device Properties dialog box should resemble [Figure 13](#).

Figure 13: Device Properties Dialog Box



7. Click **OK**.
8. In Programmer, choose **Run >  Program Device**.
A processing bar appears. Programming takes a few moments. In the Output view, info messages appear. On the board, the blinking lights stop as the boot-up design is erased.
When the programming is done, "PASS" appears in the Status column.
9. Close Programmer.
A dialog box opens asking if you want to save changes.
10. Click **Yes**.
In the File List view, source/impl_1.xcf appears under Programming Files.

Task 12: Perform Logic Analysis

In this task, you will use Reveal Logic Analyzer to set up the final options for the trigger event and view the trace signals.

Creating a Reveal Analyzer Project

You must first create a Reveal Logic Analyzer project.

To create a new Reveal Logic Analyzer project:

1. In the Radiant software main window, choose **Tools** >  **Reveal Analyzer/Controller**.

The Reveal Analyzer Startup Wizard dialog box appears.

2. In the upper left of the Reveal Analyzer Startup Wizard dialog box, select **Create a new file**.
3. Double-click in the top box and type **eval_board** to name the file.
4. In the pull-down menu on the top row next to the file name, choose **HW-USBN-2B (FTDI)**.
5. Click **Detect**.

Radiant detects the FTDI USB2 cables and they appear in the USB port menu to the left.

6. For USB port, choose the port attached to the board. This is the same port used in Programmer.

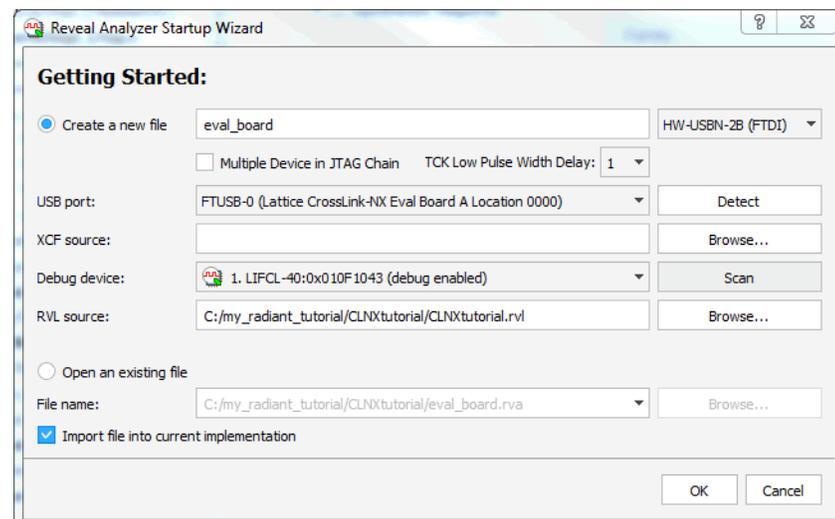
7. Click **Scan**.

The FPGA is displayed in the Debug device box.

8. In the RVL Source box, browse to `<project_directory>/CLNXtutorial.rvl`.

The startup wizard should resemble [Figure 14](#).

Figure 14: Reveal Analyzer Startup Wizard



9. Click **OK**.

Reveal Logic Analyzer appears with the LA Trigger tab selected. It contains the same trigger units and trigger expressions that you set up in Reveal Inserter.

In the File List view, "eval_board.rva [CLNXtutorial.rvl]" appears under Debug Files.

Changing the Trigger Event

In Reveal Analyzer, you cannot create new trigger units or trigger expressions, but you can change how they are named and defined. In trigger units, you can change the operator, radix, and value. In trigger expressions, you can change the expression including the operators and which trigger units are used. You can also modify some of the trigger options and the trigger position.

You probably do not want to change the trigger units or expressions, at least for the first test run. But you can make many adjustments in Reveal Analyzer without processing the design again.

To change the triggers:

1. Click the LA Trigger tab.
2. Leave the trigger units and trigger expression as they are. You can experiment with changing them later if you like.

In trigger units, you cannot change the signals used but you can change everything else.

In trigger expressions, you can change the name and the expression. But changes to the expression are limited by the maximum sequence depth and event counter set in Reveal Inserter. You can also disable an expression.

3. In the Trigger Options area (lower-left), skip the Enable TE option.

This option specifies how to combine the trigger expressions into a "trigger event:" OR or AND. But we only have one expression.

4. For Samples Per Trigger, choose **16**.

You are limited to 64 because that is the amount you set for the buffer depth in Reveal Inserter.

5. For Number of Trigger, enter **1**.

The number of samples multiplied by the number of triggers cannot be greater than the buffer depth specified in Reveal Inserter. So you could do up to four triggers with 16 samples.

6. In the Trigger Position section, select **Pre-selected** and then choose **Pre-Trigger**.

The trace data can include samples from before the trigger. The numbers next to "Trigger Position" shows which sample will have the trigger out of the total number of samples. Try the other choices.

You also have the option of selecting your own trigger position.

7. Select **Final Event Counter** and enter 1.

The data capture will be after one occurrence of the trigger event. The final event counter cannot be greater than the event counter value specified in Reveal Inserter.

Running the Logic Analyzer Core

Now that Reveal Logic Analyzer/Controller is set up, you can run the Logic Analyzer core. Then explore the LA Waveform view.

To capture data:

1. Click the Run  button in the Reveal Analyzer toolbar.

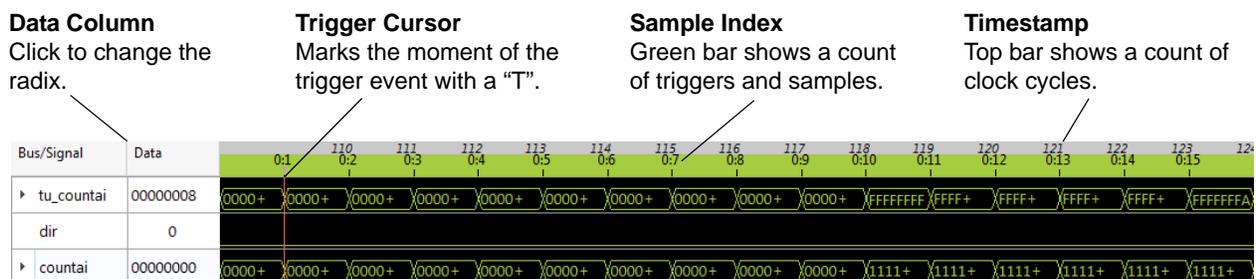
The Run button changes into the Stop  button and the status bar next to the button shows the progress.

Reveal Analyzer first configures the modules selected for the trigger event, and then waits for the trigger event to occur. When the trigger event occurs, the data is uploaded to your computer. The resulting waveforms appear in the LA Waveform tab. This takes a few moments.

If the trigger is taking too long to occur, you can force an immediate trigger by clicking the Manual Trigger  button. This button is next to the Stop  button. The waveform may show why the trigger event did not happen.

The waveform should resemble [Figure 15](#).

Figure 15: LA Waveform View in Reveal Logic Analyzer/Controller



- Click in the Data cell of countai. When the cell changes to a drop-down menu, choose **Hex**.
- Click the Zoom In  button in the toolbar until you can read the values for countai and so that the waveform is wider than the LA Waveform view.
- Click anywhere in the waveform.

A red line appears in the waveform. This is the active cursor. The values in the Data column change to match the trace sample that the active cursor is at.

5. Scroll away from the trigger cursor and then right-click anywhere in or under the waveform.

A menu appears with several commands. Some of these commands can help you move about in the waveform. For example, choose **Zoom > Zoom Trigger** to get back to the trigger cursor.

6. Right-click and choose **Add Cursor**.

A blue line replaces the red active cursor.

This is a user cursor. You can have several of them. Use them to mark interesting points in the data.

7. Scroll away from the user cursor and then right-click anywhere in or under the waveform. Choose **Go to Cursor > <number>**. The number is the sample index where the cursor is.

The trace samples can be saved three ways. To see the data in:

- ▶ The LA Waveform view again, save to this or another .rva file. Click the Save  button in the toolbar or choose **File > Save <file> As**.
- ▶ Another waveform viewer, such as ModelSim or Active-HDL, export to a value change dump file (.vcd). Right-click and choose **Export Waveform**.
- ▶ A spreadsheet, export to a text (.txt) file. Right-click and choose **Export Waveform**.

Using the Virtual Switches and LEDs

Try changing the virtual switches and watching the effect on the board and on the virtual LEDs.

You have a choice when setting the virtual switches. You can set up a value and then apply it, or you can immediately apply changes as you click individual switches. We'll try both.

To use the virtual switches and LEDs:

1. At the top of the Reveal Analyzer/Controller window, there is a drop-down menu. Choose **top_Controller**.

The window changes to show a set of virtual LEDs and switches.

2. In the Virtual Switch area, type **0x55** in the Data box. You can also set the value by clicking on the individual switches.

The virtual switches change to match the value.

3. Click **Apply**.

On the board, the lights start counting. This takes a moment.

4. In the Virtual LED area, click **Start Polling**.

The virtual LEDs start counting up similar to the demo LEDs on the board.

5. If the virtual LEDs are not keeping up with the board, try increasing the Polling Speed slider.

6. In the Virtual Switch area, select **Direct Mode**.

In Direct Mode, changing the switches immediately affects the board. The Data, Reset, and Apply controls are grayed out.

7. Click the switches to produce a value of **0xaa** in the Data box.

The real and virtual LEDs stop counting when you change the first switch. The LEDs start counting again (in reverse) when you change the last switch and have a value of 0xaa.

8. Unfortunately, you cannot run the Logic Analyzer core while the Controller core is running. The polling occupies the cable. Click **Stop Polling**.

The virtual LEDs stop but the real ones keep counting.

Accessing the User Register

Try reading and writing to RAM on the FPGA. You can write to individual addresses, initialize the entire block with a single value, or load a memory file. You can also dump the contents of the RAM to a memory file to analyze later.

To access the user register:

1. Click the User Register tab.

2. Click **Read**.

The Read Data box shows some random data.

3. In the Default Data box, type **0x0F**.

4. Click **Initialize**.

Each word in the RAM is loaded with the new value. This takes a moment.

5. Click **Read**.

The Read Data box shows 0x0000000f.

6. In the Write Data box, type a different value.

7. Click **Write**.

8. Click **Read** again.

The Read Data box shows the new value.

Accessing the Hard IP

Try reading and writing to control registers of the DPHY1 hard IP. For the most part, the controls look like the controls in the User Register tab. But some of the IP have extra controls. DPHY1 is an example.

For more information about DPHY1 and its control registers, see [“MIPI D-PHY Module - Lattice Radiant Software User Guide.”](#)

To access the hard IP:

1. Click the Hard IP tab.

The tab shows a set of boxes and buttons for the DPHY1 hard IP.

2. Click **Read**.

The Read Data box shows a string of zeros.

3. Type a 4-bit value into the Write Data box and click **Write**.

The DPHY control registers use 4-bit words.

4. Click **Read** again.

The Read Data box now shows the value you entered.

5. Explore the hard IP settings.

For the controls in the upper part of the window (High Speed Select, Bypass PLL Clock, and the PLL counters), click **Apply** to write changes. Click **Reload** to read the latest values.

6. When you are done, close Reveal Analyzer/Controller.

A dialog box asks if you want to save changes.

7. Click **No**.

Close the Radiant Project

The tutorial is done and you can close the project. You can also disconnect the board.

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.

Summary of Accomplishments

You have completed the *Lattice Radiant Software Tutorial with CrossLink-NX*. 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.
- ▶ Examine static timing analysis.
- ▶ Analyze power consumption.
- ▶ Download a bitstream to an FPGA.
- ▶ Use Reveal Inserter to add on-chip debug logic.
- ▶ Use Reveal Logic Analyzer to perform logic analysis.

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
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.

