

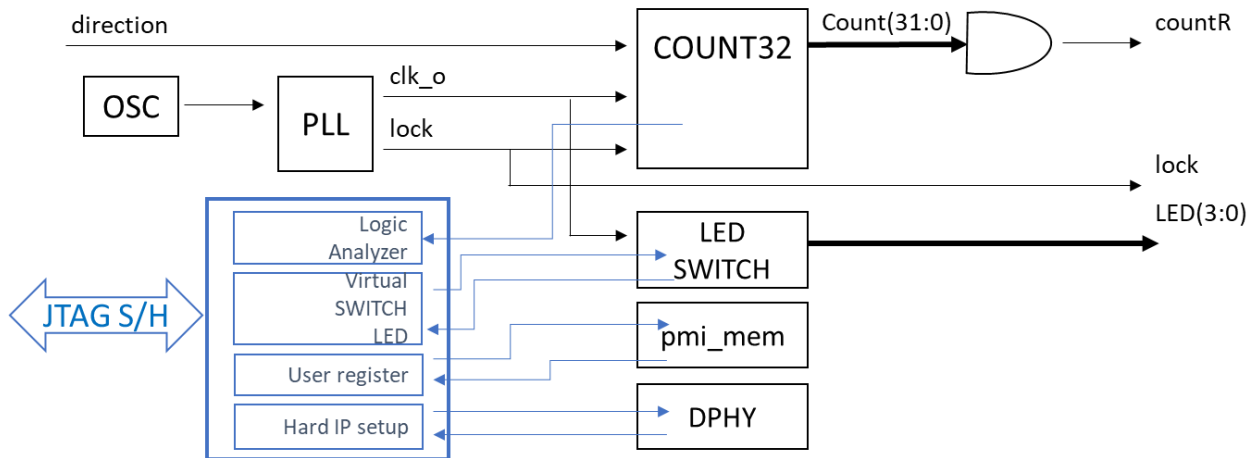
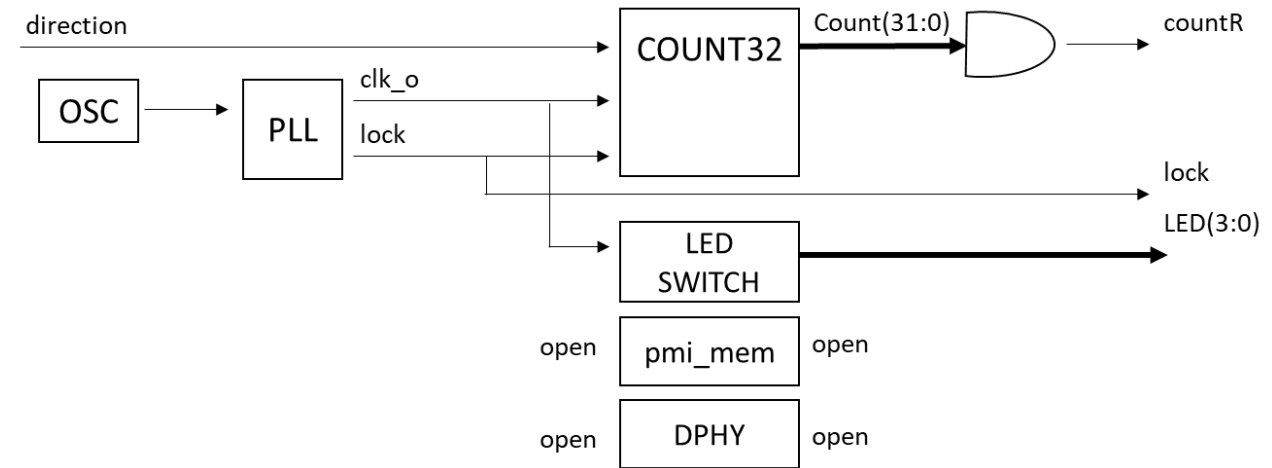


Lattice Radiant 3.2 Tutorial with CrossLink-NX (LIFCL)

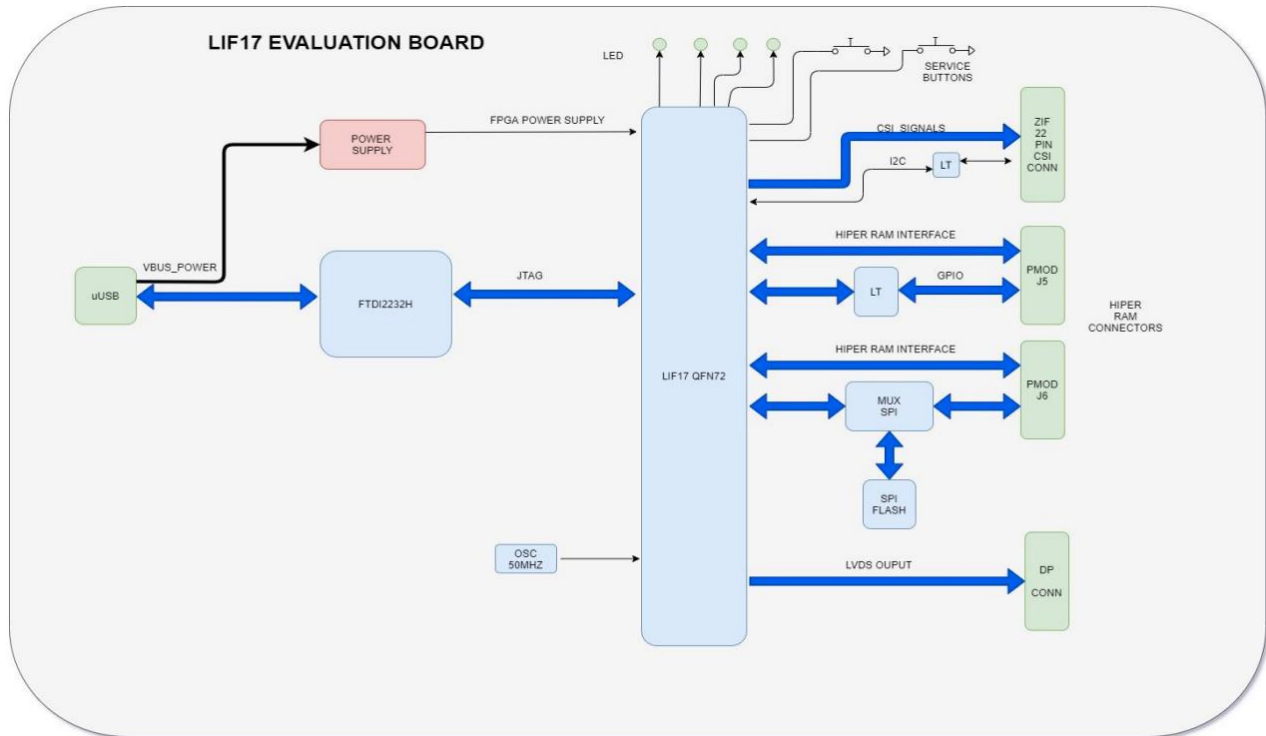
MAS-LIFCL-EVB



Reference design block diagram:



MAS-LIFCL-EVB block Diagram:



MAS-LIFCL-EVB User Manual:

[LIFCL_EVALUATION_BOARD_HARDWARE_MANUAL_R2.pdf \(maselettronica.com\)](http://maselettronica.com/LIFCL_EVALUATION_BOARD_HARDWARE_MANUAL_R2.pdf)

Lattice Radiant 3.0 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.
- ▶ Download a bitstream to an FPGA.
- ▶ Perform logic analysis.

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 47](#). 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 3.0
- ▶ (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.

This tutorial is based on MAS-LIFCL-EVB

Figure 1: Evaluation Board



Control JP4 for configuration
 OPEN: configure from FLASH
 CLOSE: IO to PMOD connector
 CLOSE to run tutorial

Hex file contained into SPI Flash (Macronix MX25L12833F 8-pin SOP) is not compatible with reprogramming. You need to delete the SPI Flash before program SPI Flash. This is not mandatory for this tutorial.

**START Procedure:*

Plug the board on USB with JP4 CLOSED. Wait for 5 second and then OPEN JP4.

Open “Radiant Programmer” from Start/Lattice Radiant 3.0/Radiant Programmer as described on paragraph Downloading the Bitstream on page 39. Execute steps 1 2 3 4 and 5. On step 6 set Device Operation as follow: Target Memory = “External SPI”, Port Interface = JTAG2SPI, Access Mode = Direct Programming, Operation = Erase All. SPI Flash select: Family = SPI Serial Flash, Vendor Macronix, Device MX25L12833F, Package = 8-pin SOP

Then Execute steps 7 8 9 and 10. Now the board is ready for the tutorial

**END Procedure*

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>**.

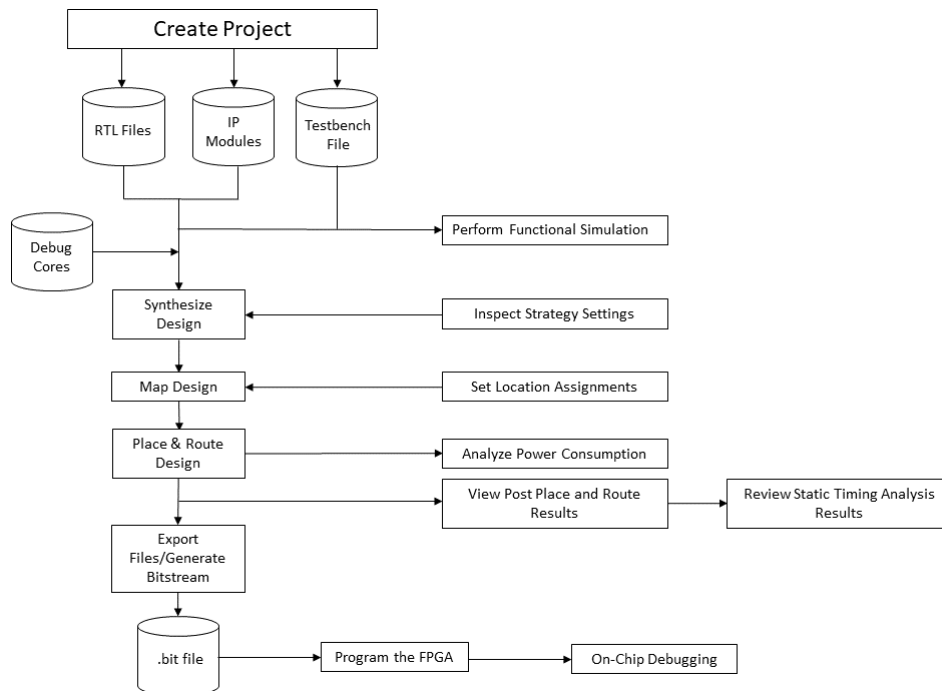
Sample Design This tutorial comes with a Verilog design that counts up and down while displaying the values on the demo LEDs of the CrossLink-NX Evaluation Board. There are also some additional modules so you can fully exercise the Radiant software’s on-chip debugging abilities: a dual-port RAM module, a module that uses the MIPI D-PHY interface, are built into CrossLink-NX. The tutorial includes a simple testbench to run the functional simulation.

In order to run the tutorial on MAS-LIFCL-EVB some change are needed to reduce the IO pin count, modified database has been provided as crosslink_nx_tutorial_MAS.zip (if you do not have the modified database see Par “How to modify the top for MAS-LIFCL-EVB only”)

About the Tutorial Data Flow

Figure 2 illustrates the tutorial data flow through the FPGA design system. You may find it helpful to refer to this diagram as you move through the tutorial tasks. All tasks shown in the flow diagram are performed in this tutorial.

Figure 2: Tutorial Data Flow



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 will 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/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 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. For this tutorial, we will use: **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. IF you have crosslink_nx_tutorial_MAS.zip, Unzip it on a tmp directory
IF you do not have it Browse to:
<Radiant_install_path>/docs/tutorial/crosslink_nx_tutorial.
3. Select the following files:
 - ▶ count32.v
 - ▶ dphy.v
 - ▶ led_switch.v
 - ▶ pmi_mem_32.v
 - ▶ tb_top.v
 - ▶ top.v (to be modified)
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

In this task we will select a device based on the tutorial requirement. We'll specify the FPGA on the CrossLink-NX Evaluation Board.

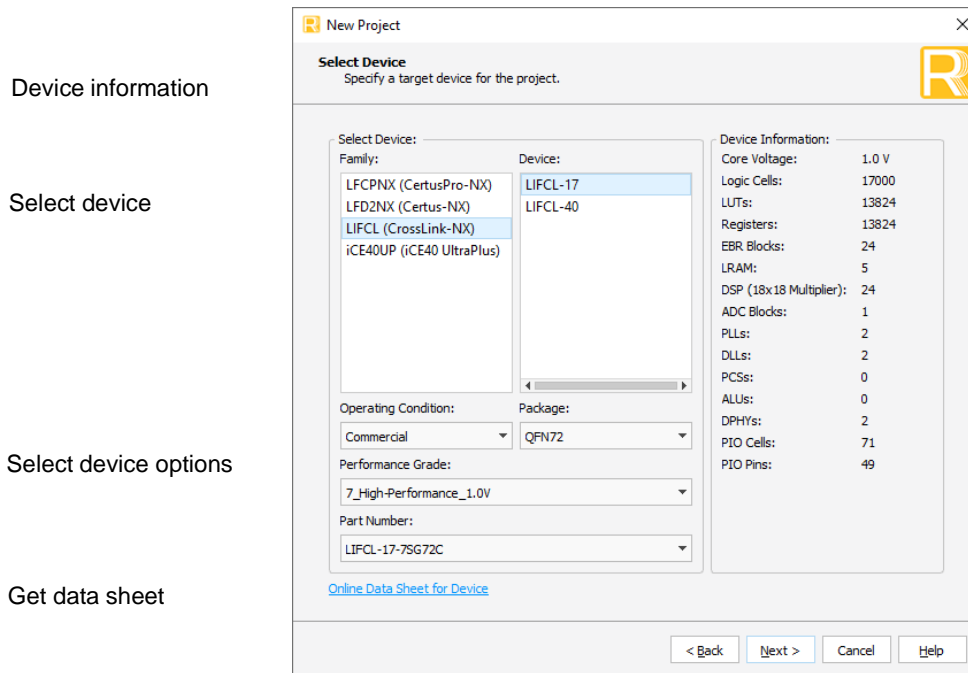
To select a device:

1. Select the device family: **LIFCL (CrossLink-NX)**.
2. Select the specific device within the family: **LIFCL-17**.
3. Select the following device options:
 - ▶ Operating Condition: **Commercial**
 - ▶ Package: **QFN72**
 - ▶ Performance Grade: **7_High-Performance_1.0V**

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

The dialog box should resemble [Figure 3](#). 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 3: New Project Wizard's Select Device Page



Device information

Select device

Select device options

Get data sheet

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 will 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, Netlist Analyzer will not be available. Synplify Pro has a similar tool, but it is 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 4](#) 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.

- In the File List view, right-click **tb_top.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** displayed in bold letters to show that it holds the top module. The Hierarchy view, which is underneath the File List view, also changes.

Figure 4: Radiant Main Window

Process Toolbar

Controls converting the design to a bitstream.

File List

Provides easy access to project components.

Tool Area

Shows the active tools.

Hierarchy

Provides access to the modules of the design.

Source Template

Helps create common features in HDL code.

IP Catalog

Get customizable modules (IP).

Tcl Console

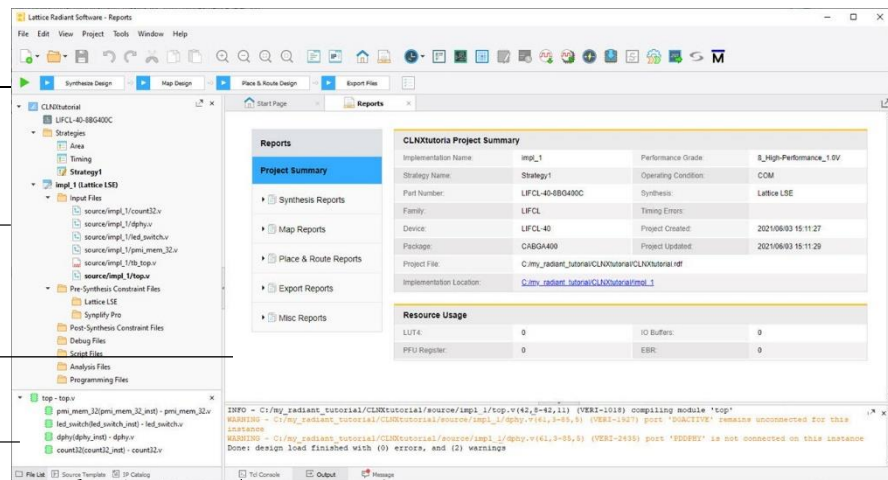
Shows and accepts Tcl commands.

Output

Shows all messages as they are produced.

Message

Shows messages organized by type.



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 Modules from IP Catalog

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

To customize and generate an OSC module:

1. 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.

2. 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.

3. Double-click **OSC**.

The Module/IP Block Wizard opens.

4. For Component name, enter **my_osc**. Use the default for the Create In location.

5. Click **Next**.

6. Set the following value:

- ▶ HFCLK Frequency (MHz): **20.4545**

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

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.


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:
 - ▶ CLKI: Frequency (MHz) (10 - 800): **20**
 - ▶ CLKOP: Frequency Desired Value (MHz) (10 - 800): **40**
8. 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.

9. Click **Generate**.
The Check Generated Result page appears. This may take a moment.
10. Ensure that **Insert to project**, in the lower-left corner, is selected and click **Finish**.
11. 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 Modules

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 instance here
*//////////`

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

- 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.

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

- Click the Save button in the toolbar.

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

To instantiate the PLL module:

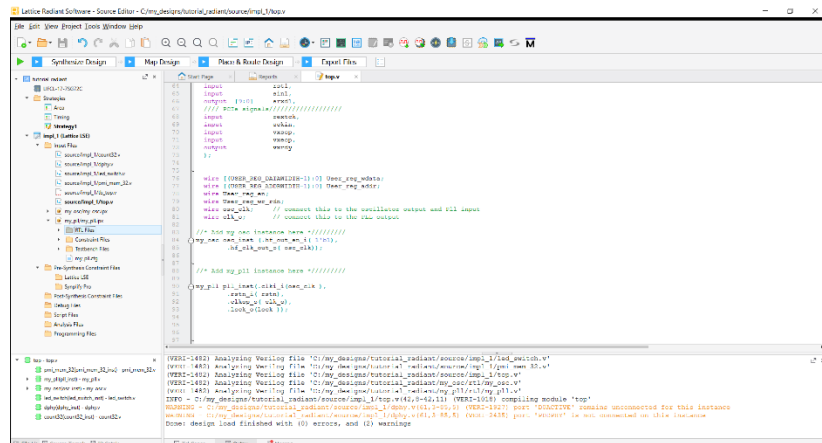
- In the Source Editor, in **source/impl_1/top.v**, scroll down to a comment that says: `/* Add my_pll instance here */`
- In the File List view, right-click **my_pll.ipx** and choose **Copy Verilog Instantiation**.
- Go to Source Editor and paste the code below the comment.
- You need to fill in a name for the instance and signal names for the ports. See below for the finished instantiation command. Bold is the text that you enter.

```
my_pll pll_inst(.clki_i(osc_clk),
               .rstn_i(rstn),
               .clkop_o(clk_o),
               .lock_o(lock));
```

- Click the Save button in the toolbar.

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

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



How to modify the top for MAS-LIFCL-EVB only (ONLY if you do not have crosslink_nx_tutorial_MAS.zip)

In order to reduce the number of IO used in tutorial following the changes needed to top.v.

Step1: remove 32 bit bus count without removing logic.

Comment line "output [31:0] count" and add a single wire output as follow:

```
58 |         output          countR,
59 |         //output [31:0] count,
```

Instead of 32 bit bus, a single wire reduce the pin number.

Somewhere add.

```
wire [31:0] count;
assign countR = (count==32'b0) ? 1'b0 : 1'b1;
```

Then, also remove the unnecessary 32 bit User_reg_rdata:

```
//output [31:0] User_reg_rdata,
```

Step2: align the leds output (original 7:0) to board led[3:0]

```
56 |         //output [7:0] leds,
57 |         output [3:0] leds,
```

And modify: add wire and assign declaration (line 154 and 155) and connect .leds to leds_int

```
151 | //-----
152 | // LED and SWITCH Control for Reveal controller
153 | //-----
154 | wire [7:0] leds_int;
155 | assign leds = leds_int[3:0];
156 |
157 | led_switch led_switch_inst (
158 |     .clk      (clk_o),
159 |     .reset    (rstn),
160 |     .leds     (leds_int)
161 | );
```

Modified top.v has been distributed with this document.

The tb_top.v needs to be modified accordingly. Comment line 51.

```
top dut(
    .rstn      (~rstn),
    .clk       (clk),
    // .count   (countt),
    .lock      (lock),
    .direction (direction)
);
```


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.

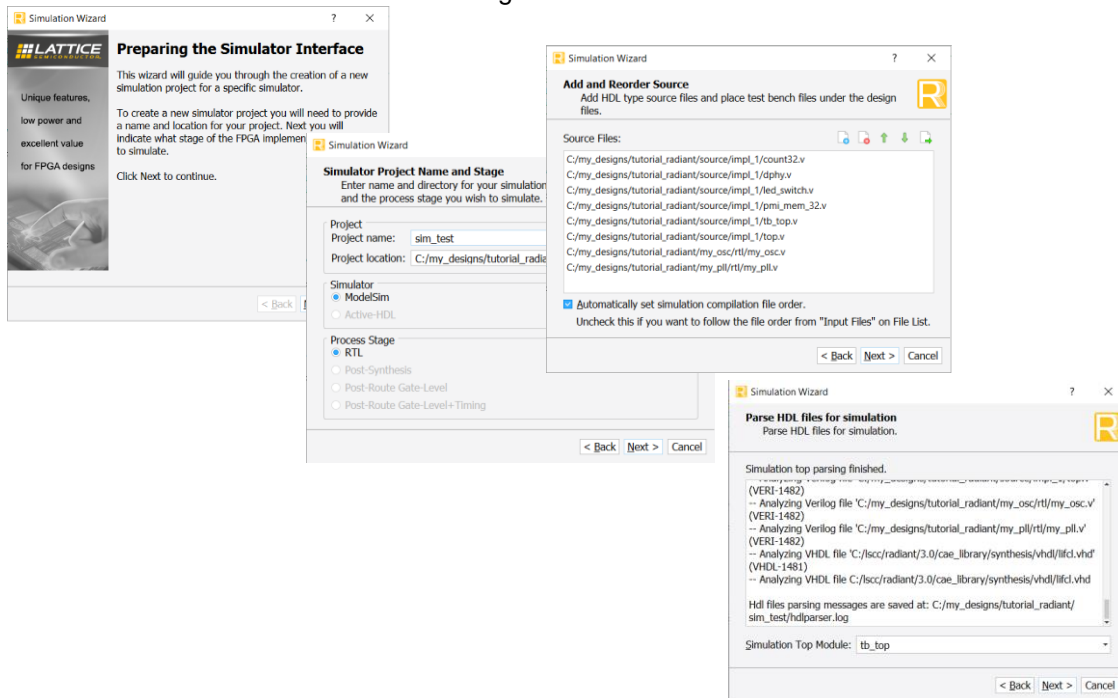
This tutorial comes with a simple testbench. You will probably create your own testbenches using your simulator. Simulators usually include tools for creating testbenches.

Starting a Simulation Run

While you can start your simulator directly, it is 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 “tb_top.” 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

Once the simulation is launched, ModelSim automatically stops after the first microsecond of simulation time. The testbench is set to run longer, but this is enough to see the startup.

To check the simulation results:

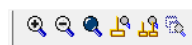
1. You probably want to expand the Wave view. Do one of the following:
 - ▶ Expand the ModelSim window.
 - ▶ Undock the Wave view. Click the Dock/Undock  button that is in the upper-right corner of the Wave view. Then expand the Wave window.
2. To make other adjustments to the Wave view, choose **Simulate > 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 **1000ns**. This is the amount of time that the Run command simulates.
4. Click **OK**.

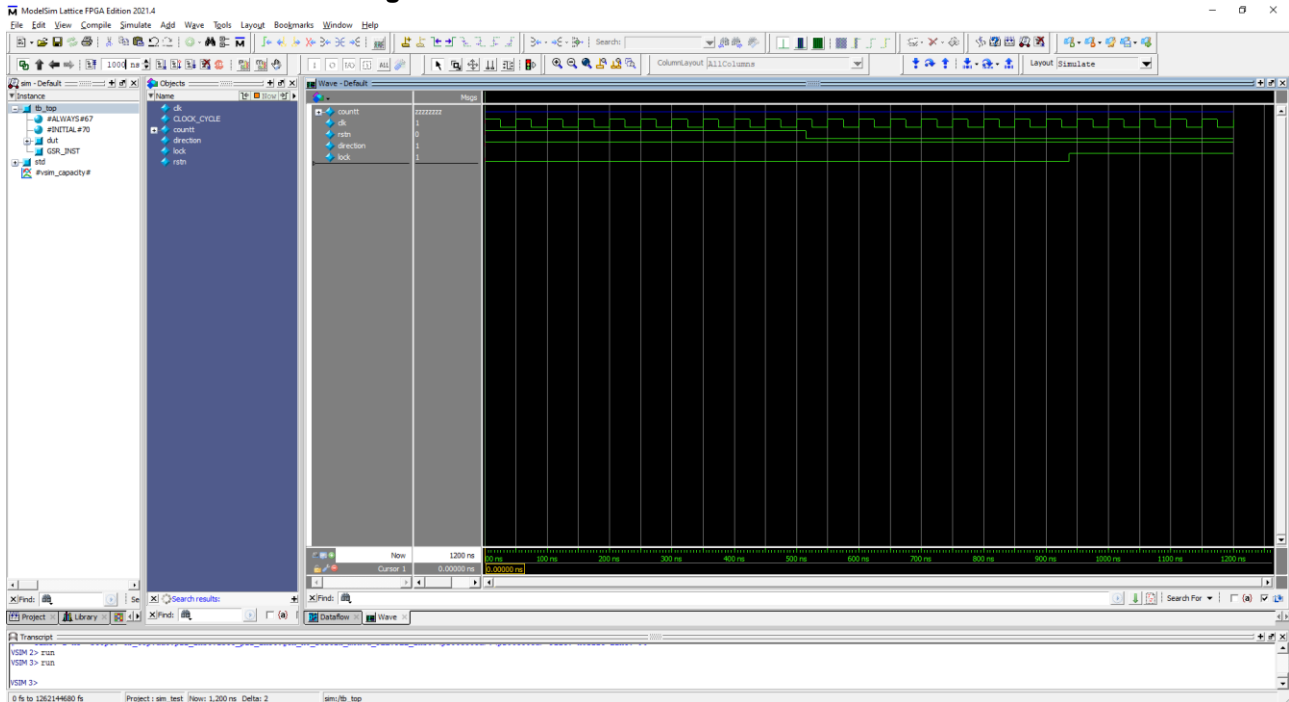
The values shown in the Objects and Wave views change to hexadecimal.

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:



Once the zoom steps are done, the ModelSim Wave View may look like [Figure 5](#), depending on the cursor location.

Figure 5: Simulated Waveform



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. 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.

7. 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 time-line. 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 > count32_inst
2. Drag **countai** from the Objects view to the Wave view.
3. 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.
4. Click **OK**.
The waveforms in the Wave view disappear.
5. Then choose **Simulate > Run > Run -All** or click the Run -All  button and wait the simulation \$stop
ModelSim's source editor opens with the testbench.v file.
6. Close testbench.v and go back to the Wave view. Now you see the full 5 μ s.
7. 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.
8. When you are done exploring ModelSim, choose **File > Quit** to close ModelSim.
The Quit Vsim dialog box opens.
9. Click **Yes**.

Task 4: Set Location Assignments

You will use the Device Constraint Editor to assign signals to the pins of the FPGA 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 assign pins:

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

The Device Constraint Editor appears.

2. First click on "synthesize design" and verify log. If you see a yellow bar with a message saying the "Design database inmemory is outdated," click **Reset Database**, which is to the right of the message and then synthesize again.

3. Click the **Port** tab, in the lower-left.

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

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

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

A filter list appears.

6. In the Search box, type **rstn**.

The filter list is reduced to the rstn port.

7. Click **OK**.

8. The spreadsheet is reduced to the rstn port.

9. Click in the Pin cell enter **36** and IO_TYPE enter **LVCMOS18H**

In the Device View, 36 shows a green dot, indicating an input port.

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

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

The filter list is reduced to the leds ports.

12. Click **OK**.

The spreadsheet is reduced to the leds ports.

13. 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]: Pin **19** IO_TYPE **LVCMOS18H**

- ▶ leds[1]: Pin 20 IO_TYPE LVCMOS18H
- ▶ leds[2]: Pin 8 IO_TYPE LVCMOS18H
- ▶ leds[3]: Pin 9 IO_TYPE LVCMOS18H

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

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

15. In the Search box, type **direction**.

The filter list is reduced to the direction port.

16. Click **OK**.

The spreadsheet is reduced to the direction port.

17. Fill in the Pin cells as follows.

- ▶ direction: 37, LVCMOS18H

18. 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 {19} [get_ports {leds[0]}]
ldc_set_location -site {20} [get_ports {leds[1]}]
ldc_set_location -site {8} [get_ports {leds[2]}]
ldc_set_location -site {9} [get_ports {leds[3]}]
ldc_set_location -site {36} [get_ports rstn]
ldc_set_location -site {37} [get_ports direction]
ldc_set_port -iobuf {IO_TYPE=LVCMOS18H} [get_ports {leds[0]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS18H} [get_ports {leds[1]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS18H} [get_ports {leds[2]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS18H} [get_ports {leds[3]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS18H} [get_ports rstn]
ldc_set_port -iobuf {IO_TYPE=LVCMOS18H} [get_ports direction]
```

19. Click the Save  button in the toolbar.

The Save dialog box opens.

20. 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.

21. 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:

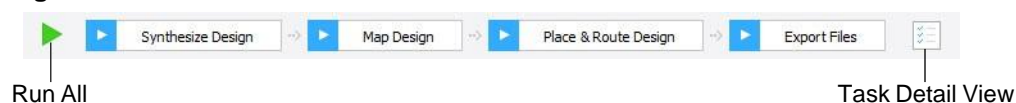
1. Synthesize converts HDL into a gate-level netlist that is optimized for the FPGA.
2. Map converts the netlist into a network of device-specific components, such as PFU (programmable function units) and I/O buffers.
3. Place and route converts the mapped network into specific components and signal 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 `led_switch_inst:1` > **leds_i3.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.

- In the toolbar of Physical Designer, click the arrow of the Routing Mode button and choose **IO Mode**.

Physical Designer 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**.

Physical Designer zooms in to the I/O for rstn: 36, 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.

- Familiarize with others function on Physical Design view.
- Close Physical Designer.

Task 7: Analyze Power Consumption

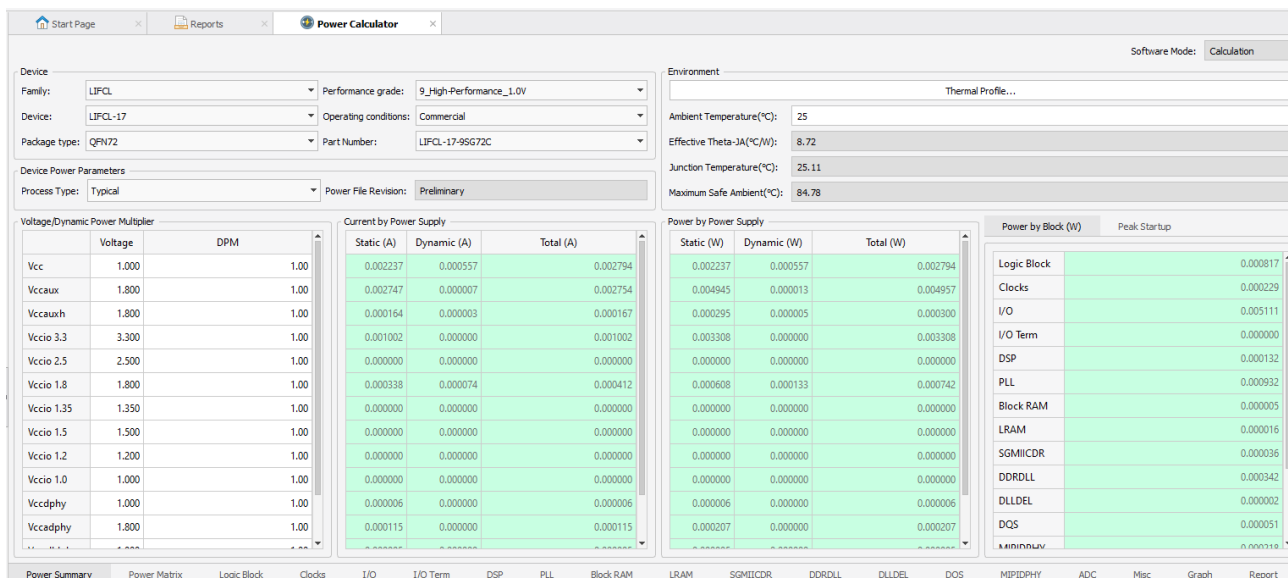
Power Calculator estimates the power dissipation for a given design. Power Calculator uses parameters such as voltage, temperature, process variations, air flow, heat sink, resource utilization, activity, and frequency to calculate the device's static and dynamic power consumption.

To analyze power consumption:

- 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:

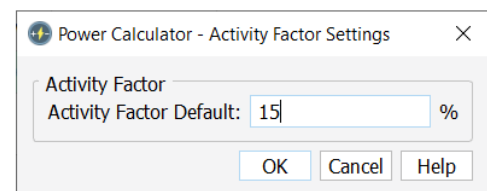
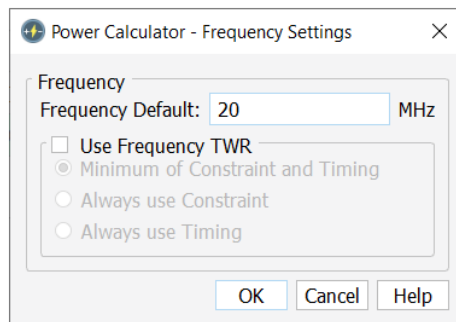
In estimation mode, Power Calculator provides estimates of power consumption based on the device resources or template that you provide. This mode enables you to estimate the power consumption for your design before the design is complete or even started.

► 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.

- Set the Frequency and Activity Factor using the wizard Edit>Frequency settings... and Edit>Activity Factor settings... 20MHz and 15%.



- For Process Type in the Device Power Parameters section, choose **Worst**. Then click Thermal Profile in the Environment section and select **Small Board** on Thermal Profile dialog box.

After a moment, the new temperature results become available in the Environment section.

- Close Power Calculator. A Confirm dialog box appears. Click **Yes**.
- 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, DPHY, and SGMIIICDR 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.

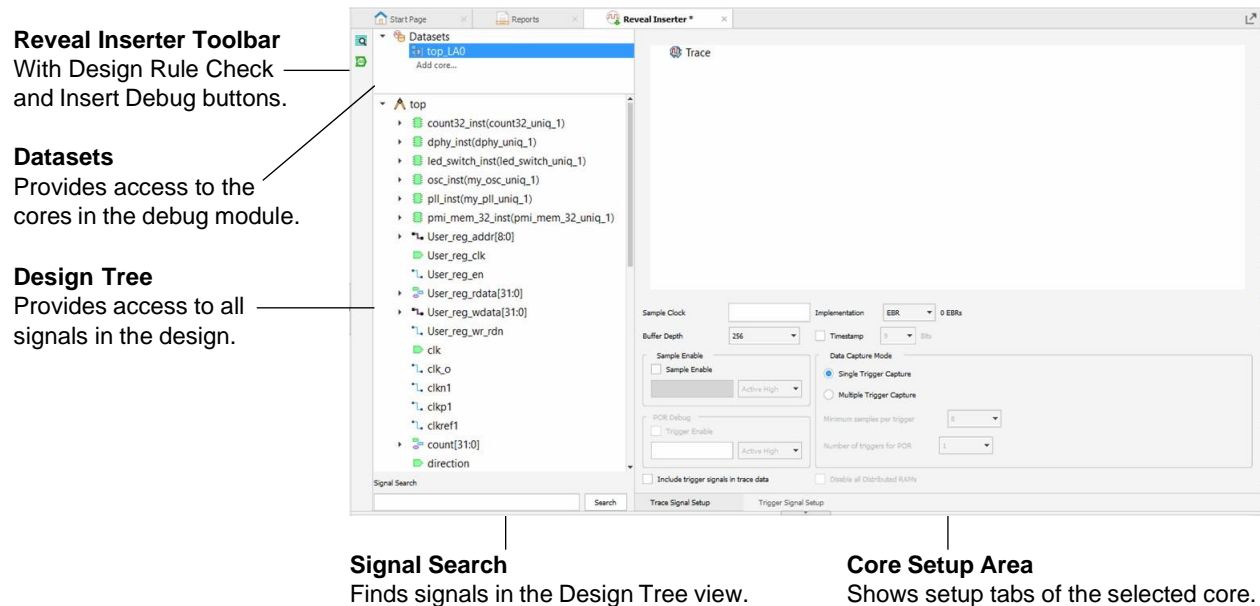
Setting Up Trace Signals

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



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. Under the same **count32_inst/(count32_uniq_1)** list, drag the **direction** 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
7. Under the same **count32_inst/(count32_uniq_1)** list, drag **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, ensure **EBR** is selected in the dropdown list.

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 **512**.

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 32 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 increased to 2 EBRs when you selected Timestamp.
4. Leave Sample Enable cleared.

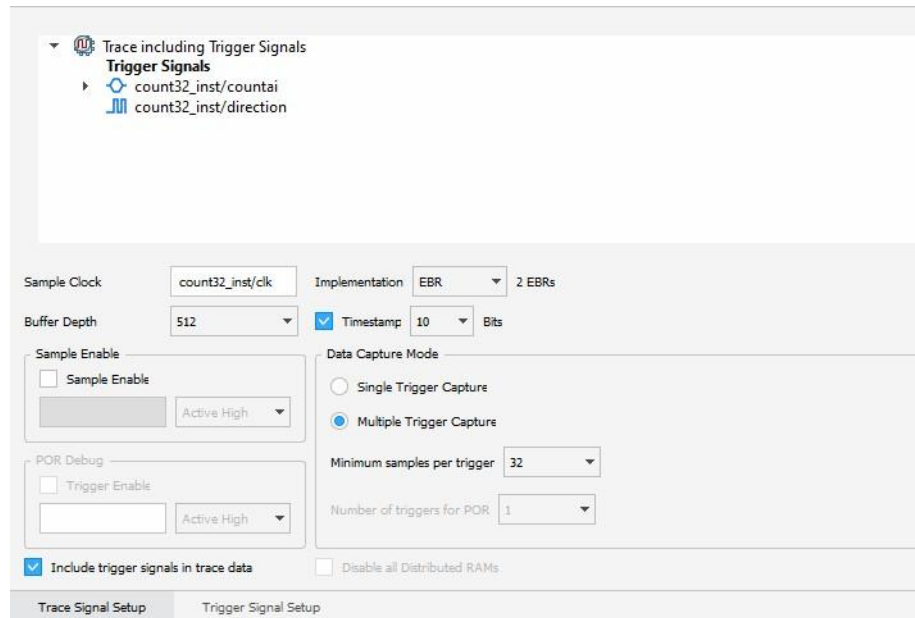
This option specifies a signal that can turn data capture on and off. Use sample enable to reduce the size of the trace buffer when there are stretches of data of no interest that are associated with a single signal.
5. For Data Capture Mode, select **Multiple Trigger Capture**.

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 32.

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.
8. Click the **Include trigger signals in trace data** box.

The Trace Signal Setup tab should now resemble [Figure 10](#).

Figure 10: 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**

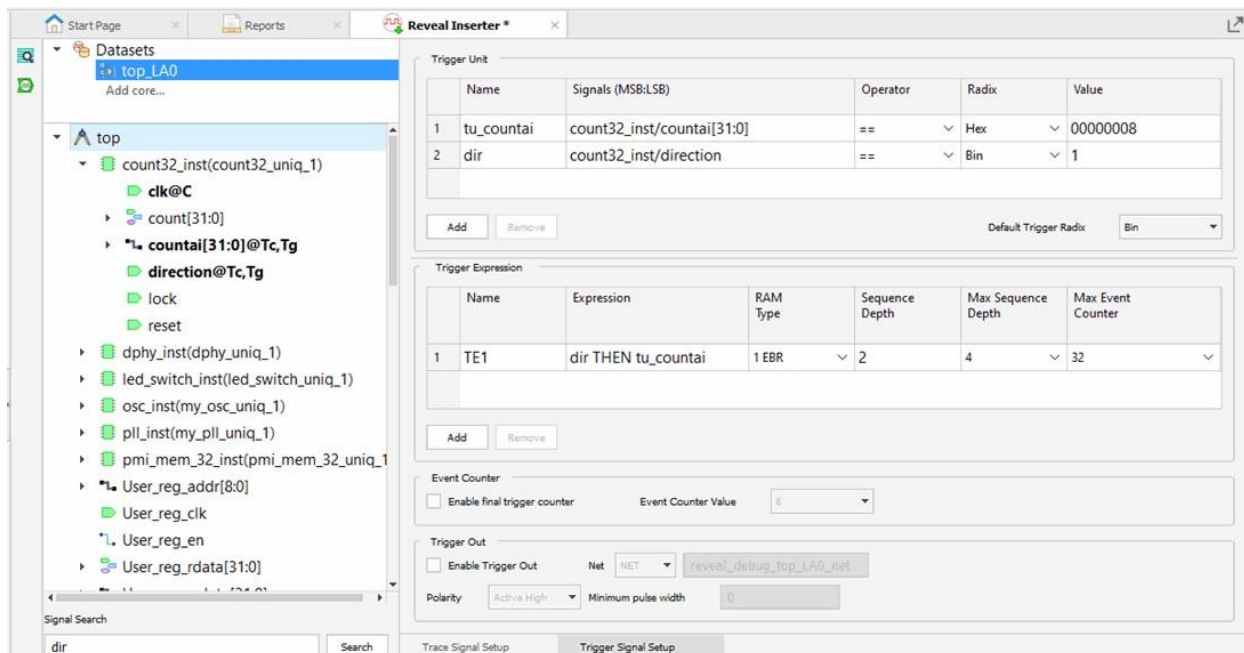
If you want to change the expression in Reveal Analyzer, this is the maximum number of sequences that will be possible.

▶ **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.

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

Figure 11: Trigger Signal Setup



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.

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 **led_switch_inst(led_switch_uniq_1)/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.

Note

Only wire signals can be added to the virtual switch list.

5. Under the same **led_switch_inst(led_switch_uniq_1)** list, 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.

6. 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 ports. 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 port types.

2. In the Design Tree view, look through the modules and find **pmi_mem_32_inst(pmi_mem_32_uniq_1)**. Expand it.
3. Fill the User Register Setup tab by dragging the matching ports from the ram1 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, DPHY, and SGMIIcdr 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 is **PLL1** and **DPHY1**,

2. In the Enabled column, select **DPHY1**. Leave **PLL1** unselected.

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 **Place & Route Design** button.
8. Go to the Reports tab.

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 - Section 1.1" shows the SDC constraints and operating conditions that guided the analysis.

Notice that you have one `create_clock` constraint and one `create_generated_clock` constraint. The `create_clock` constraint was auto-generated to specify the output of the FPGA oscillator.

Similarly, the `create_generated_clock` constraint was also auto-generated to specify the output of the PLL.

“Design Checking - Section 1.2” shows a list of combinational loops, if any.

4. Go to **2 CLOCK SUMMARY**.

“Clock Summary” shows an analysis for each clock domain defined in the constraints. The analysis shows the target frequency versus the actual (or maximum possible) and MPW (minimum pulse width) frequencies.

The “Clock Domain Crossing” section shows the slack with any other clocks that connect with the given domain. That is, a data path that has different clocks for its start and end points.

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

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

It is not necessary or desirable for all paths to have constraints. Check section 3.4, “Unconstrained Report” to make sure that all of the important paths are constrained.

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

“Detailed Report” shows details of the worst paths for setup slack and for hold slack. Each path has a section that starts with a summary of the path and the results of the analysis. 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 within the FPGA and the hierarchical name of the module's port, the type of delay, and the fanout from the pin.

If you want to visualize the path, the reports have 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. However, Netlist Analyzer often cannot show the detailed path.

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.

Timing are not closed for this design because reveal false path are missing:

```
set_false_path -from [get_clocks clk_o_N] -to [get_clocks rvltck]
set_false_path -from [get_clocks rvltck] -to [get_clocks clk_o_N]
```

Task 10: 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 47](#).

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

The Radiant Programmer opens in a separate window. In the File List view, `source/impl_1.xcf` appears under Programming Files.

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**

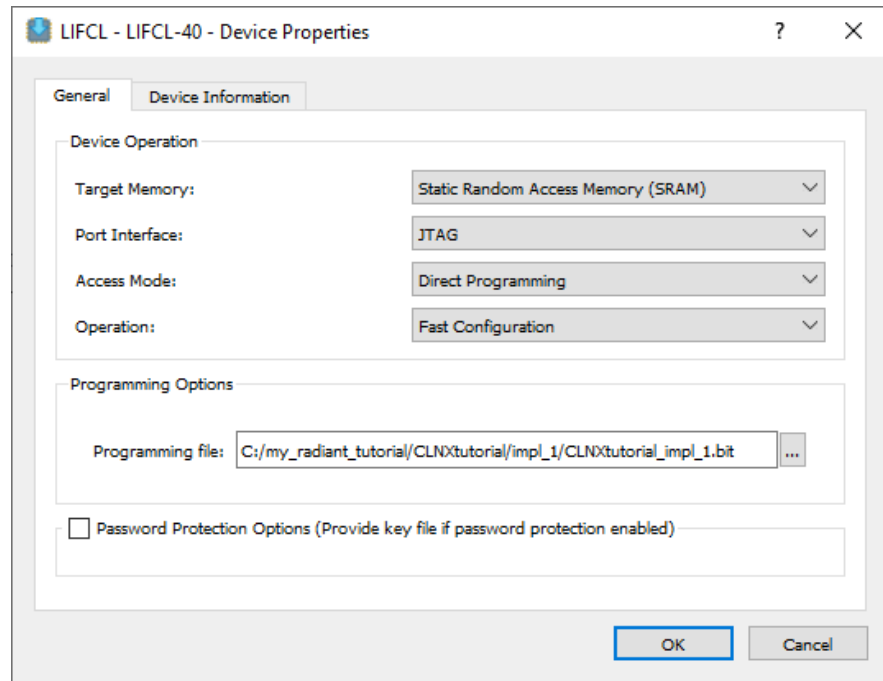
- ▶ Operation: **Fast Program**

- ▶ Programming file: *<project_path>/impl_1/CLNXtutorial_impl_1.bit*

- ▶ Password Protection Options: cleared (*not* selected)

The Device Properties dialog box should resemble [Figure 12](#).

Figure 12: 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**.

Task 11: 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.

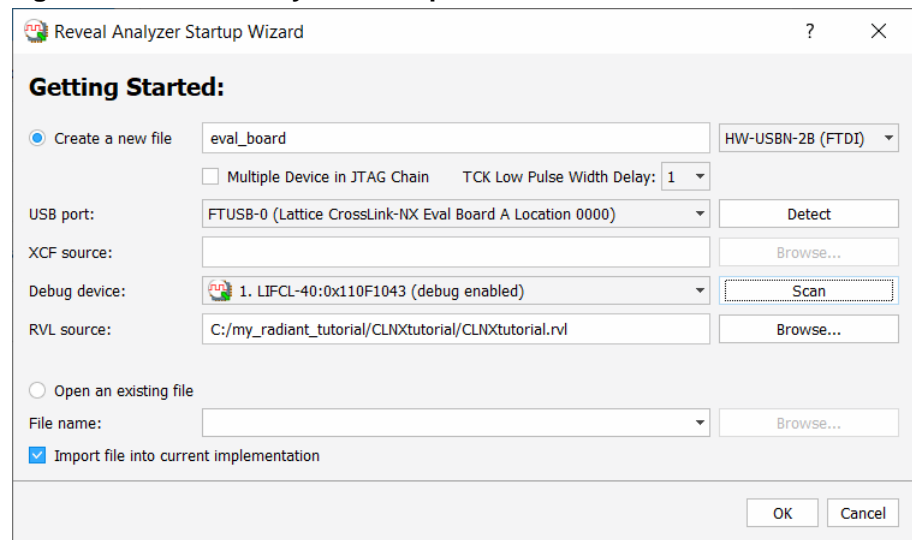
6. For USB port, choose the port attached to the board. This is the same port used in Programmer.
7. Click **Scan**.

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 13](#).

Figure 13: 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.

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.

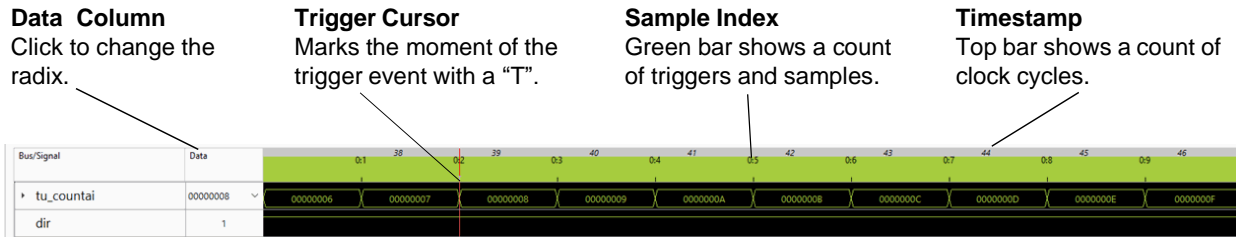
The Run button changes into the Stop button and the status bar next to the button shows the progress.

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



2. Click in the Data cell of countai. When the cell changes to a drop-down menu, choose **Hex**.

3. 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.
4. 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:

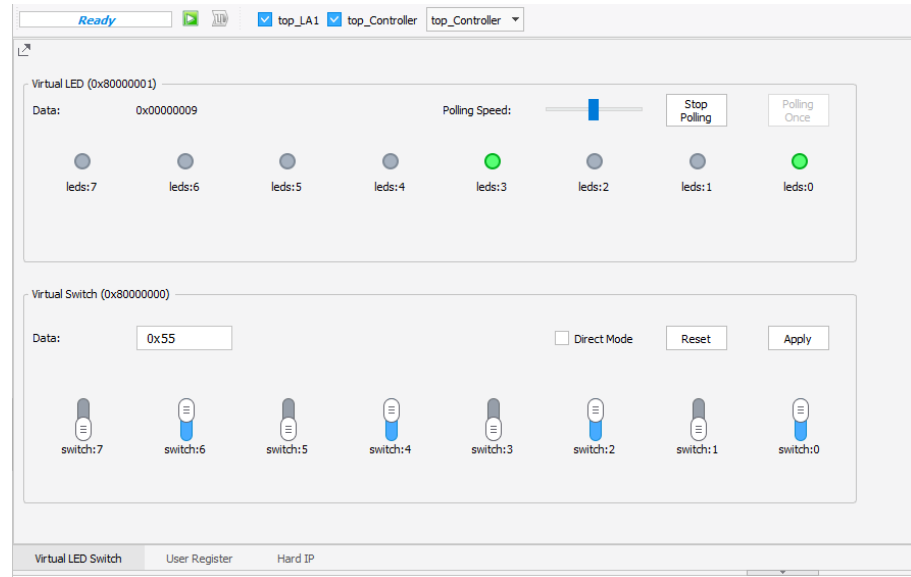
- ▶ The LA Waveform view again, save to this or another .rva file. Click the Save  button in the toolbar or choose **File > Save <file> As**.
- ▶ Another waveform viewer, such as ModelSim or GTKWave export to a value change dump file (.vcd). Right-click and choose **Export Waveform**.
- ▶ A spreadsheet, 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. The virtual switches and LEDs are shown in [Figure 16](#).

Figure 16: Virtual Switches and LEDs



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. Click **Reset**. This ensures that the Virtual LEDs and switches are reset.
3. In the Virtual LED area, click **Polling Once**. You will see a fixed Data value, and the virtual LEDs will light up accordingly.
4. In the Virtual Switch area, type **0x55** in the Data box. (You can also set the data value by clicking on the individual switches.)
5. Click **Apply**. This causes LEDs on the board to blink in the upcount direction. (This may take a moment.)
6. In the Virtual LED area, click **Start Polling**. You will see the virtual LEDs upcounting.
7. Move the **Polling Speed** slider to increase or decrease the polling speed of the virtual LEDs.
8. For downcounting, in the Virtual Switch area, type **0xAA** in the Data box. (You can also set the data value by clicking on the individual switches.)
9. Click **Apply**. This causes LEDs on the board to blink in the downcount direction. (This may take a moment.)
10. Alternately, you can also use the Direct Mode to see the LEDs change their direction. Select the **Direct Mode** box.

The Data, Reset, and Apply controls are grayed out, but you still can click switches to change the Data value.

In **Direct Mode**, if you change any of the individual switches it will change the data value in real time. This affects the LED behavior.

- In **Direct Mode**, for example, pull down switch 7. This changes the data value, and will cause the Virtual LEDs and the LEDs on the board to stop blinking. Pull switch 7 back up to see the Virtual LEDs and the LEDs on the board to start blinking.

For this tutorial design, as soon as you set the switches to **0x55** or **0xAA**, the virtual LEDs and the LEDs on the board start to blink.

- Click **Stop Polling**. This stops the movement of the virtual LEDs.

Note

You cannot run the Logic Analyzer core while the Controller core is running. The polling occupies the cable.

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:

- Click the User Register tab.

- Click **Read**.

The Read Data box shows some random data.

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

- Click **Initialize**.

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

- Click **Read**.

The Read Data box shows 0x0000000f.

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

- Click **Write**.

- 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, such as DPHY1, have extra controls.

- For more information about Hardened D-PHY, see [CrossLink-NX Hardened D-PHY Usage Guide Technical Note](#).

- ▶ For more information about DPHY control registers: see [FPGA-IPUG-02061-1.4 - MIPI DPHY Module - Lattice Radiant Software User Guide](#).

Note

This section gives a sample of Reveal Controller Hard IP access capabilities. This section does not describe all control features of the Hard IP.

To access the DPHY1 Hard IP:

1. Click the Hard IP tab.
The tab shows a set of boxes and buttons for the DPHY1 hard IP.
2. In DPHY1, click **Read**.
The **Read Data** box shows a string of zeros.
3. Type a 4-bit hex value into the **Write Data** box and click **Write**.
The DPHY1 control registers use 4-bit hex.
4. Click **Read** again.
The Read Data box now shows the value you entered.

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. You can also disconnect the board.

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**.
You can also disconnect the board.

Summary of Accomplishments

You have completed the *Lattice Radiant 3.0 Tutorial with CrossLink-NX (LIFCL)*. In this tutorial, you have learned how to:

- ▶ Create a new Radiant software 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.
- ▶ Download a bitstream to an FPGA.
- ▶ Use Reveal Logic Analyzer to perform logic analysis.

Recommended References

You can find additional information on the subjects covered by this tutorial in:

- ▶ [Radiant 3.0 Software Help](#) (HTML version, accessed from the Radiant software Help menu)
- ▶ [Radiant 3.0 Software Help \(PDF version\)](#)
- ▶ [Radiant 3.0 Software User Guide](#)
- ▶ [Reveal User Guide for Radiant Software](#)