# Lattice Radiant Software 1.0 Help

February 13, 2018

## Copyright



## Trademarks



## Disclaimers

## Type Conventions Used in This Document

| Convention | Meaning or Use |
|---|---|
| **Bold** | Items in the user interface that you select or click. Text that you type into the user interface. |
| <*Italic*> | Variables in commands, code syntax, and path names. |
| **Ctrl+L** | Press the two keys at the same time. |
| `Courier` | Code examples. Messages, reports, and prompts from the software. |
| `...` | Omitted material in a line of code. |
| `.`<br>`.`<br>`.` | 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

# Getting Started with Lattice Radiant Software

Lattice Radiant™ software is the complete design environment for Lattice Semiconductor Field Programmable Gate Arrays (FPGAs). The software includes a comprehensive set of tools for all design tasks, including project management, design entry, simulation, synthesis, place and route, in-system logic analysis and more.

To help you experiment further, example projects are available. These examples are designed to illustrate different aspects of designing with Radiant software. To see the examples, click the (Open Example) button, or choose **File > Open > Design Example**. Then open an example folder and open the example's .rdf file. (An .rdf file defines a design project for Radiant software.)

The rest of the Help system provides complete instructions for the different stages of the Radiant software design flow and extensive reference material.

**See Also**

**Lattice on the Web**

- Lattice Semiconductor
- Answer Database
- Lattice Solutions
- Intellectual Property

## Tutorials

To quickly get some hands-on experience, try the following:

- Lattice Radiant Software Tutorial

# User Guides

To learn more about Radiant software, see the following User Guides and Reference Manuals:

- Lattice Radiant Software 1.0 Release Notes
- Lattice Radiant Software User Guide
- Lattice Radiant Software Guide for Lattice Diamond Users
- Migrating iCEcube2 iCE40 UltraPlus Designs to Lattice Radiant Software
- Reveal Troubleshooting Guide for Lattice Radiant Software
- Lattice Radiant Software 1.0 Help (HTML)
- Lattice Radiant Software 1.0 Help (PDF)
- Lattice Radiant Software 1.0 Installation Guide for Windows
- Lattice Radiant Software 1.0 Installation Guide for Linux
- Active-HDL On-line Documentation (Windows only)
- Synopsys Synplify Pro for Lattice User Guide
- Synopsys Synplify Pro for Lattice Reference Manual

# Getting Help

For almost all questions, the place to start is this Help. It describes the FPGA design flow using Radiant software, the libraries of logic design elements, and the details of the Radiant software design tools. The Help also provides easy access to many other information sources.

To make the most effective use of the Help, please review this section.

## Opening the Help

The Help can be opened in several ways:

- In Windows, choose **Start > Programs > Lattice Radiant > Accessories > Radiant Help**.
- In the main window, choose **Help > Radiant Help**.
- To go directly to the Help for the tool that you're using, choose **Help >** <*Tool Name*> **Help**.

JavaScript must be enabled in your default browser. If you are have trouble opening the Help, see "Troubleshooting the Help" on page 8.

## Using the Help

The Help has several features to help you find information.

**Contents** The contents organizes the information in the Help. Look here to see what subjects are covered or to begin in-depth study of a subject.

Click the (Contents) button to hide or show the contents pane.

**Search** The search locates all topics that contain specific text. This can be the fastest way to find information once you are familiar with the contents of the Help.

Enter one or more words and click the (Search) button. The Search page opens with a list of topics containing all of the words (an AND function).

Some tips:

- Enter phrases between a pair of double-quote marks. For example: "block module".
- Use an asterisk * as a wildcard to also find plurals and different verb tenses, or to search on word parts. For example, search* finds search, searches, searching, and searched. Searching on *annotate finds annotate and backannotate.
- Capitalization does not matter.
- Leave out punctuation except for hyphens in hyphenated words.

Search results are ordered by the number and types of hits found. The first few topics are most likely to have substantial information on the search terms. Topics at the end of the list may have just a casual mention of the terms.

### Search Navigation Tip

Your browser's forward and back buttons may not work the way you expect with the Search page. To go to the Search page, always click the (Search) button. If you're on the Search page and want to go back to the topic you were looking at, click the (Contents) button.

# Type Conventions Used in This Document

| Convention | Meaning or Use |
|---|---|
| **Bold** | Items in the user interface that you select or click. Text that you type into the user interface. |
| <*Italic*> | Variables in commands, code syntax, and path names. |
| **Ctrl+L** | Press the two keys at the same time. |
| `Courier` | Code examples. Messages, reports, and prompts from the software. |
| `...` | Omitted material in a line of code. |

| Convention | Meaning or Use |
|---|---|
| .<br>.<br>. | 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. |

# Troubleshooting the Help

If you are have trouble opening the Help, check for the following situations:

**Active Content or Scripts Are Blocked** Opening the online Help may be interrupted by one of the following messages on the Internet Explorer Information Bar:

- "To help protect your security, Internet Explorer has restricted this file from showing active content that could access your computer. Click here for options..."
- "To help protect your security, Internet Explorer has restricted this file from running scripts or ActiveX controls that could access your computer. Click here for options..."

To see the Help, click on the Information Bar and choose **Allow Blocked Content**. A dialog box with an expanded warning opens. Click **Yes**.

To avoid these warnings, either use a different browser or turn off the warning for active content in Internet Explorer.

**Note**

Doing either of these means that when you open any Web page that is resident on your computer—not just Radiant Help—the page will automatically run any active content that it has. While active content is common and can be very useful, malicious content can damage your files. Be sure you trust the software on your computer.

**To turn off the warning:**

1. In Internet Explorer, choose **Tools > Internet Options**.
2. Click the **Advanced** tab.
3. Under Security, select **Allow active content to run in files on My Computer**.
4. Click **OK**.

**Back and Forward Buttons Do Not Work with the Search Page** Your browser's back and forward buttons may not work the way you expect with

the Search page. Instead of returning to the Search page, you go to the previous or next topic. Instead of returning from the Search page to the previous or next topic, you stay on the Search page. The Search page is not in the stack of visited pages.

To go to the Search page, always click the  (Search) button. If you're on the Search page and want to go back to the topic you were looking at, click the  (Contents) button.

## Contacting Technical Support

**FAQs** The first place to look. The Answer Database provides solutions to questions that many of our customers have already asked. Lattice Applications Engineers are continuously adding to the Database.

**Technical Support Request** Submit a Technical Support Request through www.latticesemi.com/techsupport.

**For Local Support** Contact your nearest Lattice Sales Office.

# Copyright, Trademarks, and Disclaimers



**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.

# Managing Projects

You can create a project using the Lattice Radiant software with the New Project wizard. Launch the New Project Wizard by clicking the **New Project** button, or choose **File > New > Project**. The Radiant software creates a folder with a project file (.rdf) containing basic information about the project, the beginning of a physical design constraints (.pdc) file, which controls how the design is implemented in the map and place-and-route stages, and a default strategy (Strategy1.sty).

A “strategy” is a collection of settings for controlling the different stages of the implementation process (synthesis, map, place & route, and so on). Strategies can control whether the design is optimized for area or speed, how long place-and-route takes, and many other factors. The Radiant software provides a default strategy, which may be a good collection to start with, and some variations that you can try. You can modify Strategy1 and create other strategies to experiment with or to use in different circumstances.

The project folder also contains an implementation folder. Implementation folders contain the source files, process reports, and other information for different implementations, or versions, of a design. Having different project implementations helps you to experiment and compare different designs. You can check the details of each implementation in a set of process reports (choose **View > Reports**).

As you develop your design, routine tasks can be controlled through the Radiant software graphical user interface or through scripts. The Radiant software comes with command-line and Tcl commands for many of its functions.

**See Also**


- “Running the Radiant Software” on page 2
- “Creating a New Project” on page 3
- “Modifying a Project” on page 5
- “Importing Lattice Diamond Projects” on page 6
- “Targeting a Device” on page 6

# Running the Radiant Software

The Radiant software main window is the primary interface and provides an integrated environment for managing the project elements and processes, as well as accessing all Radiant software tools and views.

**To run the Radiant software main window:**

- From your Windows desktop, choose **Start > All Programs > Lattice Radiant Software > Radiant Software**.

  **Note**

  **Lattice Radiant Software** is the default Programs folder name when you install the Radiant software. Change this name accordingly if you have chosen another folder name during installation.

- From your Linux platform shell window or C-shell window, execute:

  `<install_path>/bin/lin64/Radiant`

**See Also** "Managing Projects" on page 1

# Pin the Radiant Software to Start Menu or Taskbar

You can pin the Radiant software to your Windows Start menu or taskbar, so you can open it quickly and conveniently, rather than looking for the Radiant software in the Start menu.

**To pin the Radiant software to the Start menu or Taskbar:**

1. Choose the Windows **Start** menu and find the **Lattice Software** icon.

2. Right-click the **Lattice Software** icon.
3. Click **Pin to Start Menu** or **Pin to Taskbar**.

**Note**

You should not pin the Radiant software to the Taskbar through the minimized icon on the taskbar while the Radiant software is running.

If you want to remove the pinned Radiant Software from the Start menu or the taskbar, right-click on a pinned **Radiant Software** icon from the Start menu or the taskbar, choose **Unpin from Start Menu** or **Unpin this program from taskbar**.

**See Also** ▶"Managing Projects" on page 1

# Creating a New Project

A project is a collection of all files necessary to create and download your design to the selected device. The New Project wizard guides you through the steps of specifying project name and location, selecting a target device, and adding existing sources to the new project.

**Note**

Do not place more than one project in the same directory.

**To create a new project:**

1. From the Radiant software main window, click the **New Project** button, or choose **File > New > Project**.

   The New Project wizard opens.
2. Click **Next**.
3. In the Project Name dialog box, do the following:
   - Under Project, specify the name for the new project.

     File names for Radiant software projects and project source files must start with a letter (A-Z, a-z) and must contain only alphanumeric characters (A-Z, a-z, 0-9) and underscores (_). Spaces are allowed.
   - To specify a location for your project, click **Browse**. In the Project Location dialog box, you can specify a desired location.
   - Under Implementation, specify the name for the first version of the project. You can have more than one version, or "implementation," of the project to experiment with. For more information on implementations, refer to "Working with Implementations" on page 18.
   - When you finish, click **Next**.

4. In the Add Source dialog box, do the following if you have an existing source file that you want to add to the project. If there are no existing source files, click **Next**.

Click **Add Source**. You can import HDL files at this time.

   a. In the Import File dialog box, browse for the source file you want to add, select it, and click **Open**.

      The source file is then displayed in the Source files field.

   b. Repeat the above to add more files.

   c. To copy the added source files to the implementation directory, select **Copy source to implementation directory**. If you prefer to reference these files, clear this option.

   d. To create empty Lattice Design Constraint (.ldc) file and Post-Synthesis Constraint File (.pdc) files that can be edited at a later time, select **Create empty constraint files.** Refer to "Timing and Physical Constraints" on page 119 for more information about constraint files.

   e. When you finish, click **Next**.

1. In the Select Device dialog box, select a device family and a specific device within that family.

   a. Choose the options you want for that device.

   b. Click **Next**.

2. In the Select Synthesis Tool dialog box, select the synthesis tool that you want to use. This choice can be changed at any time. See "Selecting a Synthesis Tool" on page 48 for more information. When you finish, click **Next**.

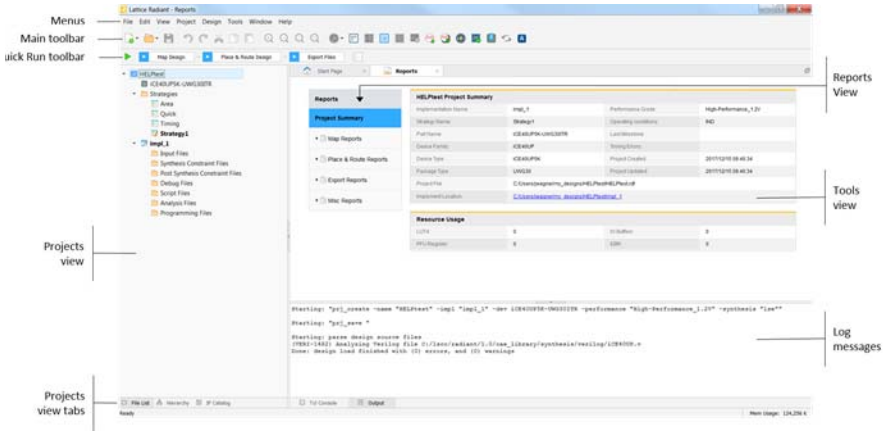
3. In the Project Information dialog box, make sure the project settings are correct and then click **Finish**.

   **Note**

   If you want to change some of the settings, click **Back** to modify them in the previous dialog boxes of the New Project Wizard.

Once a project has been created, the Radiant software workspace opens in the main window and displays the following:

**Figure 1: Radiant Software Main Window Workspace**




**See Also** ▶"Managing Projects" on page 1

# Modifying a Project

After creating a project, you can modify the project by using the right-click menu.

**To modify a project:**

1. Click on the File List tab at the bottom left side of the screen to display the File List view.
2. Right-click on any part of the project in the File List view.

   The right-click menu varies upon the different part of the project you have chosen.

   - You can choose to edit a device in the Device Selector, add source files, implementations, or strategies as needed to the project, clone a strategy, set the current highlighted strategy as active strategy, exclude certain source files from an implementation or remove a file.
   - You can also edit properties of the project in the Project Properties dialog box. You can set the top-level unit, specify the VHSIC Hardware Description Language (VHDL) Library name, and Verilog Include Search path in the Project Properties dialog box.

**See Also** ▶"Managing Projects" on page 1

# Importing Lattice Diamond Projects

Design projects created in Lattice Diamond software can be imported into the Radiant software using the Import Diamond Project wizard. Imported Lattice Diamond projects are targeted to devices supported in the Radiant software. Lattice Diamond design preferences will be converted into Radiant software design constraints.

**To import a Diamond software project into the Radiant software:**

1. In the Radiant software, choose **File > Open > Import Diamond Project**.
2. In the **Select Diamond Project** dialog box, browse to the Diamond Project file (.ldf) and select it.

   The Import Diamond Project wizard appears. Click **Next.**
3. In the Select Device dialog box, choose Device Family, Device, Operating Condition, Package, Performance Grade, and Part Number. Click **Next.**
4. In the Project Name dialog box, specify the Name and Location of your project. Lists of files to be imported, and files not to be imported, appear.
5. Click **Copy to Radiant project folder** if you wish to copy the imported files into your new Radiant software project.
6. Click **Finish**.

**See Also** ▶"Managing Projects" on page 1

# Targeting a Device

The Radiant software lets you re-target a design to a different device any time during the design process.

**To target a device:**

1. In the File List view, double-click the device name.

   The Device Selector dialog box opens. It contains all available devices and their options.
2. Under Select Device, select a device family and a specific device within that family. Then choose the options you want for that device.
3. When you finish, click **OK**.

   The specified target device appears in the File List view.

**See Also** ▶"Managing Projects" on page 1

# Viewing Project Properties

After creating a project, you can view the project-related information in the Project Properties dialog box.

**To view project properties:**

1. Right-click on any part of the project in the File List view, and choose **Properties**.

   The Project Properties dialog box opens.

2. In the dialog box, you can see the name, category, and location of the selected file. You can enter values for some of the selected sections of the project in the Value field.

**See Also** ▶"Managing Projects" on page 1

# Saving Project Files

You can save changes to source files and to project properties. You can also save copies of individual files and whole design projects.

**Saving Changes** As you are making changes to source files or to the project properties you should frequently save the files. The Radiant software offers three save commands for this:

- **File > Save**: Saves the currently active item.
- **File > Save All**: Saves all the content changes.
- **File > Save Project**: Saves project properties such as the target device, implementation file lists, and strategy setting.
- **File > Save Project As**: Opens the Save Project As dialog to save the active project.
- Click the **Save** icon (  ).

Also, if there are any unsaved changes when you close a project, a dialog box will open offering a chance to specify which files you want to save.

**Copying a Project** You can make a copy of a design project to use as the beginning of another project. (If you want the copy to backup or move the project, see "Archiving a Project" on page 8 instead.) This process saves all source and other project files that are within the project folder to another folder. Referenced files are not included.

**To copy a design project:**

1. Choose **File > Save Project As**.

   The Save Project As dialog box opens.

2. In the dialog box, browse to where you want to save the copy.
3. Click the **Create New Folder** button to create a new project folder.
4. Type a new name for the folder and press **Enter**.
5. Double-click the new folder to open it.
6. If desired, give the project a new name in the "File name" box.

7. Click **Save**.

   If there are open files, the Save Modified Files dialog box may open. Select files to be saved before copying and click **OK**.

   The current design project closes and the new one opens.

Check references to files outside of the project folder. These may need adjustment.

**Archiving a Project** The Radiant software offers you an easy way to archive the current project. A project archive is a single compressed file (.zip) containing the information for the entire project. After archiving a project, you can reload it in the main window at any time.

When you archive a project that contains source files stored outside the project folder, the remote files are compressed under the remote_files subdirectory in the archive.

**To archive the current project:**

1. In the main window, make sure the project you want to archive is open.
2. Choose **File > Archive Project**.
3. In the Archive Project dialog box, specify the file name and location.
4. Click **Save**.

   The archive file is created and stored in the specified location.

**To reload the archived project:**

1. In the Radiant software main window, choose **File > Open > Archived Project**.
2. In the Select Archived Project File dialog box, browse to the archive file.
3. Click **Open**.
4. In the Open Archived Project dialog box, check the destination directory. If this is not where you want to place the project files, click the **Browse** (**...**) button.
5. In the Browse for Folder dialog box, browse to where you want to place the project files.
6. Click **OK**.
7. In the Open Archived Project dialog box, click **OK**.

   The project files are extracted to the specified folder and the project is opened in the Radiant software.

**See Also** ▶"Managing Projects" on page 1

▶ "Copying a Project" on page 7

# Managing Project Sources

The Radiant software combines design sources into different categories and lists them in the File List view, as well as any folders. associated with them. The source files are classified and listed under these folders, which include Strategies, Input Files, Synthesis Constraint Files, Debug Files, Script Files, Analysis Files, and Programming Files.

The following files can be found in their respective folder, as shown in Figure 1

**Table 1: Source File Locations**

| Files | Folder |
|---|---|
| HDL/IP Module | Input Files |
| Logic Constraint | Post-Synthesis Constraint Files |
| Reveal Project | Debug Files |
| Power Calculator/Timing Constraints | Analysis Files |

You can also see implementations listed in the File List view. For details about implementations, see “Working with Implementations” on page 18.

**File List View** The File List view of the Radiant software main window lists all the design sources. The sources are categorized by different types and are identified with different icons. Bold items are active and will be used when processing the design project. Grayed out items will not be used.

**Table 2: Files Listed in the File List View**

| File Type | File Extension |
|---|---|
| Project Title | None |
| Target Device | None |
| Predefined Strategy (Area, I/O Assistant, Quick, and Timing) | .sty |
| Strategy1 (that can be customized) | .sty |
| Implementation | None |
| Verilog Files | .v, .veri, .ver, .vo, .h |
| Structural Verilog | .vm |
| SystemVerilog | .sv |
| IP Module Config Files | .ipx |
| Undefined or incorrect | Any source reference |
| Synthesis Constraint Files | .sdc, .fdc, .ldc |
| Reveal Project File | .rvl |

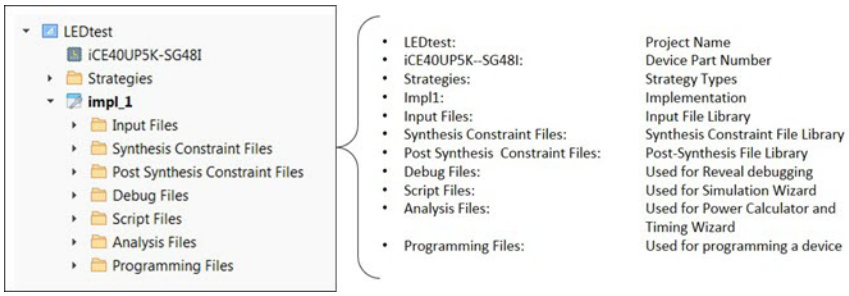
**Table 2: Files Listed in the File List View**

| File Type | File Extension |
|---|---|
| Simulation Project File | .spf |
| Reveal Analyzer Files | .rva |
| Power Calculator Files | .pcf |
| Programmer Project File | .xcf |

**Note for Linux**

In the Linux version of the Radiant software, when you save or export a file, the dialog box does not automatically append an extension to the file name you specify. That is the standard way Linux handles the Save As dialog box. You need to manually add the relevant file extension.

**Figure 2: File List**




**See Also** ▶"Managing Projects" on page 1

▶ "Running Processes" on page 42

# Creating a New Source File

You can create a new source from within the Radiant software or external to the Radiant software.

**To create a new source file:**

1. In the Radiant software main window, choose **File > New > File**.

   The New File dialog box opens.

2. In the dialog box, select the type of file you want to create under Categories and Source Files.

   a. Fill in the file name.

   b. Browse to choose a location for the file.

   c. Check the **Add to Implementation** option.

   d. Choose the Implementation you want the file to be part of.

3. Click **New**.

The Radiant software starts an editor that you can use to enter the information for your new source file.

4. In the editor, create a source file.

**Note**

- File names for Radiant software projects and project source files must start with a letter (A-Z, a-z) and must contain only alphanumeric characters (A-Z, a-z, 0-9) and underscores (_).
- In the Linux version of the Radiant software, when you save or export a file, the dialog box does not automatically append an extension to the file name you specify. That is the standard way Linux handles the Save As dialog box. You need to manually add the relevant file extension.

**See Also** ▶"Managing Projects" on page 1

- "Analyzing a Design" on page 26

# Importing an Existing Source File into a Project

You can easily import existing source files into your project. Source files for a project can be stored in different locations.

The files added to the project appear in the Input Files folder. You can adjust the file order by drag-and-drop. If several modules are detected as being uninstantiated, the last one will be automatically treated as the top-level module. You can also set the top-level module from the implementation property dialog box.

**To import an existing source file:**

1. Choose **File > Add > Existing File** from the Radiant software. Or, with a project opened in the Radiant software, right-click the current implementation and choose **Add > Existing File**.

   The Add Existing File dialog box opens.

2. Browse for the source file you want to import.
3. Select the file and click **Open**

   The selected file is added to the Input Files folder of the File List view. If the source file is stored outside the project folder, the path of the file is displayed.

4. Double-click the imported file. The file can be opened in the associated editor (this is a Windows-only feature).

**Tip**

- Radiant software project creation supports mixed language source files, allowing you to freely mix Verilog HDL and VHDL design. The Radiant software will figure out the hierarchy structure that is associated with the module names. You need to pay special attention to the module names and the case, as Verilog is case-sensitive and VHDL is case-insensitive.

**See Also** ▶"Managing Projects" on page 1

- "Analyzing a Design" on page 26
- "Setting the Top-Level Unit for Your Project" on page 16

## Adding Reveal Debug Information

Reveal Inserter manages the addition of Reveal debug information into the Radiant software source file. By default the Reveal Inserter project file (.rvl) will be automatically added to the File List view once Reveal Inserter has successfully merged trigger and trace signal features into your design. The procedure below describes how to manually import an .rvl file.

**To import a Reveal project file (.rvl):**

1. In the Radiant software main window, choose **File > Add > Existing File**.

   The Add Existing File dialog box opens.

2. Browse for the .rvl file you generated and select it.
3. Click **Add**.

   The selected .rvl file is listed in the File List view.

   You can have multiple Reveal project files for your project. However, you can only set only one .rvl file active at one time.

For more information, refer to "Testing and Debugging On-Chip" on page 328.

**See Also** ▶"Managing Projects" on page 1

## Importing Test Stimulus Files Using Simulation Wizard

If your design needs Verilog test fixture (.v) or VHDL test bench (.vhd), use the **Simulation Wizard**.

If multiple hierarchical test stimulus files are to be used, you should first import the top-level test file, and then add lower-level test stimulus as dependency files. The Radiant software will run all these test files in a hierarchical order during the simulation process.

**To import a test stimulus file:**

1. In the Radiant software, choose **Tools > Simulation Wizard**.

   The Preparing the Simulator Interface dialog box opens.

2. Click **Next**.
3. In the Simulator Project Name dialog box, enter a name for your simulation project in the Project name field.
   a. Browse to find a desired location for your project.
   b. Choose either Active-HDL or ModelSim as the Simulator.
4. In the Process Stage dialog box, all the available process stages of the FPGA implementation strategy are automatically displayed. Choose the stage you want to run simulation.
5. Click **Next**.
6. In the Add Source dialog box, browse for your desired Verilog test fixture (.v) or VHDL test bench (.vhd) and select it. Check the **Copy Source to Simulation Directory** option if you need.
7. Click **Next**.
8. In the Summary dialog box, all your simulation project information is listed, including simulator name, project name, project location, simulation stage, simulation files, and simulation libraries.

   Check the “Run simulator” option if you want to run the simulation right away.

9. Click **Finish** to exit the Simulator Wizard. Your simulation project (.spf) is added to the Script Files folder of the File List view.

**Note**

The simulation flow in the Radiant software supports source files that can be set in the File List view to be used for the following purposes:

- Simulation & Synthesis (default)
- Simulation only
- Synthesis only.

This allows the use of test benches, including multiple file test benches. Additionally, multiple representations of the same module can be supported, such as one for simulation only and one for synthesis only.

Refer to “Simulating the Design” on page 96 for more details on how to use the Simulation Wizard.

**See Also** ▶“Managing Projects” on page 1

# Managing Constraint Files

The following are the different types of constraint files available in the Radiant software.

- Synthesis constraint file: FDC, LDC
- Post-Synthesis Constraint file: PDC

**About Synthesis Constraint Files** You can add .fdc files for Synplify Pro or .ldc files for Lattice Synthesis Engine (LSE), depending on the synthesis tool you choose. The files are listed in the Synthesis Constraint Files folder in the File List view.

**Adding, Activating, Editing, and Removing Synthesis Constraint Files**
You can add, activate, edit, or remove synthesis constraint files in your project.

**To add a synthesis constraint file:**

1. Right-click the **Synthesis Constraint Files** folder in the File List view, and choose **Add**.
2. Choose **New File** if you want to add a new synthesis constraint file.
   a. Select the source files to add.
   b. Enter a file name and select a location.
   c. Click **New**.
3. Choose **Existing File** if you want to add an existing synthesis constraint file.
   a. Browse to the location of the synthesis constraint file to add and select it.
   b. Click **Add**. The selected file is added to your project.

**To activate a synthesis constraint file:**

- Right-click the synthesis constraint file in the File List view of the Radiant software, and choose **Set as Active**.

  The selected synthesis constraint file is now activated.

**Note**

- You can make zero or one .ldc constraint file active in your project.
- You can make multiple .fdc constraint files active in your project.

**To edit a synthesis constraint file:**

- Double-click the synthesis constraint file you want to edit from the File List view of the Radiant software. Or, right-click the synthesis constraint file you want to edit from the File List view of the Radiant software and click **Open**.

  The selected synthesis constraint file is opened in the Source Editor. You can edit the selected file in the editor. See “Using Templates” on page 64 for more information.

**To remove a synthesis constraint file:**

- Right-click the synthesis constraint file in the File List view of the Radiant software, and choose **Remove**.

  The selected file is removed from your project.

**About Post-Synthesis Constraint Files** You can add one or more physical constraint files (.pdc) to your project. They are listed in the Post-Synthesis Constraint File folder of the File List view.

**Note**

You can set only one .pdc file active in your project at one time.

**Adding, Activating, Editing, and Removing Post-Synthesis Constraint Files** You can add, activate, edit, or remove Post-Synthesis Constraint files (.pdc) of your project.

**To add a Post-Synthesis Constraint file:**

1. Right-click the **Post-Synthesis Constraint Files** folder in the File List view of the Radiant software, and choose **Add**.
2. Choose **New File** if you want to add a new Post-Synthesis Constraint file.
   a. Select the source files to add.
   b. Enter a file name and select a location.
   c. Click **New**.
3. Choose **Existing File** if you want to add an existing Post-Synthesis Constraint file.
   a. Browse to the location of the Post-Synthesis Constraint file to add and select it.
   b. Click **Add**. The selected file is added to your project.

**To activate a Post-Synthesis Constraint file:**

- Right-click the Post-Synthesis Constraint file in the File List view of the Radiant software, and choose **Set as Active**.

  The selected file is activated.

**Note**

You can set only one .pdc file active in your project at one time.

**To edit an Post-Synthesis Constraint file:**

- Double-click the Post-Synthesis Constraint file you want to edit from the File List view of the Radiant software. Or, right-click the Post-Synthesis Constraint file you want to edit from the File List view of the Radiant software.

The selected Post-Synthesis Constraint file is opened in Source Editor. You can edit the selected file in the editor. See "Using Templates" on page 64 for more information.

**To remove an Post-Synthesis Constraint file:**

- Right-click the Post-Synthesis Constraint file in the File List view of the Radiant software, and choose **Remove**.

  The selected file is removed from your project.

See "Applying Design Constraints" on page 115 for more information.

**See Also** "Managing Projects" on page 1

# Setting the Top-Level Unit for Your Project

The top-level unit (or module) of the design must be specified. If it's not, the Radiant software will attempt to determine the top-level unit on its own.

Since the possibility exists that the Radiant software could choose the incorrect unit, it is recommended that you manually set the top-level unit.

**To set the top-level unit for your project:**

1. In the File List view, right-click the active implementation folder  .
2. From the drop-down menu, choose **Set Top-Level Unit**.

   The Project Properties dialog box opens with the cursor in the T**op-Level Unit** Value cell..
3. Type the name of the top-level unit.

   The Value cell also shows an arrow for a drop-down menu. Usually the menu shows one name or just "<Edit>." But if the Radiant software found multiple possible top-level units, the menu will show those names. If the correct top-level unit is in the menu, you can choose it.
4. Click **OK**.

   If the hierarchy is up-to-date, the source file that contains the top-level unit is displayed in bold.

You can also change the top-level unit when experimenting with different designs or switching between simulation and synthesis.

**See Also** "Managing Projects" on page 1

# Modifying a Source File

Double-click on a source file to edit it using the Source Editor.

**Note**

For Windows file associations to work properly with the Radiant software, choose **Tools > Options > File Associations** and assign default programs for any of the programs listed.

You can use the Source Editor to enter or edit VHDL (.vhd), Verilog HDL (.v), or constraints (.pdc) files.

**Modifying a Source File using a Different Text Editor** Alternatively, you can use the text editor of your choice to edit your source files, and then import them into your project using **File > Add > Existing File** from the Radiant software main window.

**Note**

A .pdc file can also be created or edited in the Spreadsheet View.

Do the following to open a source file using a different text editor:

1. In the File List view, right-click the text file and choose **Open with** from the pop-up menu.
2. Click **Add** if you want to add your favorite text editor;

   **Note:** If you've already added a new text editor, click **Edit** to open the source file in that editor.
3. Specify command arguments for the external editor. The mapping between the argument substitution values and the value which replaces them are:

   %F = File Name; %L = Line Number

   For example, to configure the gVim text editor, use the following command line: <*path*>/gvim +%L %F

   a. In the Options dialog box, choose **Options > Environment > File Associations**. From the File Associations table, choose the file type you want to open/edit in the new text editor.

   b. Click **Add** from the **External programs for ... file extension** field to add the text editor. Use the **Edit** or **Remove** button to manage your file and associated editor list.

After adding or editing the external text editor, you can save it as your default text editor by clicking **Save As Default** in the **Open With** dialog box.

**Note**

If you change a source file in a third-party text editor, you need to choose **Design > Refresh Design** so that the Radiant software can refresh the process list to reflect the current process status and update views, such as Hierarchy.

**See Also** ▶"Managing Projects" on page 1

## Excluding a Source File

You can exclude specific source files from logic synthesis and simulation. The source files do remain in your design project, but won't be synthesized or simulated by the software.

**To exclude a source file:**

1. In the File List view, right-click the target source file and choose **Exclude from Implementation**.

   The selected file is excluded from the implementation and the target file is grayed out.

**Note**

To include the source file again, right-click the target source file and choose **Include in Implementation**.

**See Also** ▶"Managing Projects" on page 1

## Removing a Source File

You can remove a source file from a project by using the Radiant software **Remove** command or the Windows' **Delete** command.

Removed source files are no longer part of your project.To simply exclude a file from logic synthesis and simulation, see "Excluding a Source File" on page 18.

**To remove a source file from a project:**

1. In the File List view, select the file that you want to remove.
2. Right-click the filename and choose **Remove**, or press the **Delete** key on your keyboard.

   The Radiant software removes the source from the project.

**See Also** ▶"Managing Projects" on page 1

# Working with Implementations

Implementations define the design structural elements of a project. An implementation contains the structure of a design and can be thought of as the source and constraints to create the design. For example, one implementation may use inferred memory and another instantiated memory.

There can be multiple implementations in a project, but only one implementation can be active at a time and there must be at least one implementation.

An implementation is automatically created whenever you create a new project. You can create new implementations using the source files of an existing implementation. You can also clone (copy) an existing implementation, which includes all related files and settings.

Implementations consist of:

- Gate-Level Simulation File
- Input files
- Constraint files
- Debug files
- Script files
- Analysis files
- Programming files

### Creating New Implementations

To create a new implementation:

1. With the project opened in the Radiant software, choose **File > New > Implementation**.
2. In the New Implementation dialog box, enter a name for the new implementation.

   This name also becomes the default name for the folder of the implementation.
3. Change the name of the implementation's folder in the Directory text box, if desired.
4. Change the location of the implementation's folder if desired.
5. Choose the synthesis tool from the drop-down menu.
6. Choose the default strategy from the drop-down menu.
7. Add sources files by clicking **Add Source** and choosing either:
   - **Browser** to select individual files.
   - **From Existing Implementation >** <*implementation name*> to use the source files of an implementation that already exists in this design project.

   Remove unwanted files by selecting the files and clicking **Remove Source**.
8. If you want to copy all of the source files into the new implementation folder, giving you the freedom to change the files without affecting other implementations, select the **Copy source to implementation directory** option.

9. To view or set properties of the source files, select a file and click **Properties**. The Properties dialog box opens.
    a. To change a property, click in its **Value** cell and enter or choose the new setting.
    b. Click **OK**.
10. Click **OK**.

### Cloning Implementations

To clone an implementation:

1. In File List view, right-click on the name of the implementation that you want to copy and choose **Clone Implementation**.

    The Clone Implementation dialog box opens.
2. In the dialog box, enter a name for the new implementation. This name also becomes the default name for the folder of the implementation.
3. Change the name of the implementation's folder in the Directory text box, if desired.
4. Decide how you want to handle files that are outside of the original implementation directory. Select one of the following options:
    - **Continue to use the existing references**

      The same files will be used by both implementations.
    - **Copy files into new implementation source directory**

      The new implementation will have its own copies that can be changed without effecting the original implementation.
5. Choose the Synthesis tool to use in the Synthesis Tool text box.
6. Enter the default strategy in the Default Strategy text box.
7. Click **OK**.

**Customizing Files in an Implementation** After creating an implementation, you can choose to include or exclude source files for the implementation. You can customize files that are under the Constraint Files, Debug Files, Script Files, and Analysis Files folders.

**Customizing Input Files** You can choose to:

- Add a new source file or add an existing file.
- Include the file for Synthesis, Simulation, or Synthesis and Simulation.
- Exclude the file from the current implementation.
- Remove the file from the project.

**Activate Files** You can have multiple files under each of the Constraint Files, Debug Files, Script Files, and Analysis Files folders in the File List view. But only one file can be active in each of those folders. To change a file's status, right-click the file and choose **active** or **inactive**.

**Saving an Implementation** If you want to save an implementation to the project, choose **File > Save Project** from the Radiant software main window.

**See Also** ▶"Managing Projects" on page 1

▶ "Using Strategies" on page 21

# Using Strategies

A **strategy** provides a unified view of all settings related to the optimization controls of an implementation tool, such as logic synthesis, mapping, and place-and-route. There are two different kinds of strategies: predefined strategy and customized strategy.

Any number of strategies can be applied to your project to find the best solution to meet your design objectives. For example, you might have one strategy with fast placement and routing attempt to take a quick look at the design, and another that's higher level and more thorough that takes longer to meet timing constraints. Although you can create an unlimited number of strategies, only one can be active at a time for each implementation, and each implementation must have an active strategy.

Strategy settings are listed in the Strategies dialog box. Open the dialog box by double-clicking a strategy name in the File List view. Each strategy is stored as an .sty file.

**Predefined Strategy** Several strategies are predefined and supplied by Lattice: Area, I/O Assistant, Quick, and Timing. They are designed to solve particular types of designs.

- **Area**

  The Area strategy attempts to minimize area by enabling the tight packing option available in map. Use this strategy to achieve area requirements.

  Applying this strategy to large and dense designs may cause some difficulties in the place-and-route process with longer run time or not completing routing. However, when it works, you will see an area reduction.

- **Quick**

  The Quick strategy uses a very low effort level in place-and-route to get results with minimum processing time. If your design is small and your target frequencies are low, this is a good strategy. Even if your design is large, you might want to start with this strategy to get a first look at place-and-route results and to tune your constraints file with minimum processing time.

  The quality of these results in terms of achieved frequency realistically should be low, and large or dense designs may not complete routing. However, you will get results quicker when it works.

The settings of predefined strategies cannot be modified in the Strategies dialog box. You can view their settings by:

- Double-clicking the predefined strategy name in the Files List view and opening the Strategies dialog box.
- Right-clicking on the predefined strategy name in the File List view and choosing **View** to open the Strategies dialog box.
- **Timing**

  The Timing strategy attempts to achieve timing closure. The Timing strategy uses a very high effort level in place-and-route. Use this strategy if you are trying to reach the maximum frequency on your design. If you cannot meet timing requirement with this strategy, try to customize a strategy with individual settings.

  This strategy may increase your runtime on place-and-route compared to the Quick and Area strategies. However, you will get improved time performance results when it works.

**Customized Strategy** Excluding the predefined strategies, all other strategies can be customized in the Strategies dialog box. Edit the settings of a strategy by:

- Double-clicking the strategy name in the File List view and opening the Strategies dialog box.
- Right-clicking the strategy in the File List view and choosing **Edit** to open the Strategies dialog box.

**See Also** ▶"Managing Projects" on page 1

- "Creating a Strategy" on page 22
- "Cloning a Strategy" on page 23
- "Adding an Existing Strategy" on page 23
- "Specifying Strategy Options" on page 24
- "Saving a Strategy" on page 25
- "Removing a Strategy" on page 25
- "Strategy Reference Guide" on page 380

# Creating a Strategy

New strategies can be created for your design.

**To create a new strategy in the Radiant software:**

1. Choose **File > New > Strategy**.

   The New Strategy dialog box opens.
2. Enter a name for the new strategy.
3. Specify a file name for the new strategy
4. Choose a directory to save the strategy file (.sty) to.

The new strategy is created with all the default settings of the current design. You can modify its settings in the Strategies dialog box.

To save the strategy changes to your current project, choose **File > Save Project** from the Radiant software main window.

**See Also** ▶"Managing Projects" on page 1

# Cloning a Strategy

Clone any previously created strategy or any of the Lattice predefined strategies from the File List view. The new cloned file contains all settings of the strategy being cloned. You can then modify the settings for the cloned strategy from within the Strategies dialog box.

**To clone a strategy:**

1. In the File List view, right-click on the strategy you want to clone and choose **Clone** <*name*> **Strategy**.

   The Clone Strategy dialog box opens.
2. Enter a name for the new strategy in the Strategy ID field. Note that the name used automatically appears as the file name.
3. Change the file name, if desired.
4. Choose a directory to save the strategy file (.sty) to.

The cloned strategy contains all the settings of the base strategy. You can modify the settings for the cloned strategy in the Strategies dialog box.

To save the strategy changes to your current project, choose **File > Save Project** from the Radiant software main window.

**See Also** ▶"Managing Projects" on page 1

# Adding an Existing Strategy

A previously created strategy can be added to your design.

**Note**

A strategy copied from another design project may include incorrect settings, such as path names, that could cause problems. If you copy a strategy from another project, review its settings carefully and adjust them as needed.

**To add existing strategy to your design:**

1. In the Radiant software, choose **File > Add > Existing Strategy**.
2. In the Select Existing Strategy dialog box, browse to the desired strategy file (.sty) and click **Open**.

3. In the Add Existing Strategy dialog box, enter the name for the new strategy. The name chosen appears in the File List view.

**Note**

If the selected strategy file is not in the current project directory, you may want to select the **Copy strategy file to project directory** option to avoid using a file being used by another project.

4. Click **OK**.

   The new strategy appears in the File List view.

5. Double-click the new strategy to open the Strategies dialog box.
6. Review the settings to ensure they are appropriate for your current project. Be sure to check the following:
   - Under Synthesize Design > LSE:

     **Macro Search Path**

     **Memory Initial Value File Search Path**
   - Under Post Synthesis:

     **External Module Files(.udb)**
7. After completing your review, click **OK**.

To save the strategy to your current project, choose **File > Save Project** from the Radiant software main window.

**See Also** ▶"Managing Projects" on page 1

# Specifying Strategy Options

Use the Strategy dialog box to specify settings for a specific design strategy process.

**To specify strategy settings:**

1. In the File List view, double-click the active strategy.

   The Strategies dialog box opens.
2. In the left-hand pane, select a process. The left-hand pane shows a list of processes. Each process has its own settings.
3. In the right pane, double-click the Value column for the setting to be changed. A pulldown menu is displayed with a choice of settings. You can also enter filenames or directory paths into the field shown.
4. Select the value from the drop-down list, or type in the text value, and click **Apply**.
5. Repeat steps 2-4 to specify more settings.
6. When you finish, click **OK** to close the dialog box.

All of your changes are saved to your strategy (.sty) file.

**Tip**

When a setting is highlighted, a definition of it is displayed at the bottom of the dialog box. You can also press **F1** to view additional information for the highlighted option.

**See Also** ▶"Managing Projects" on page 1

▶ "Strategy Reference Guide" on page 380

# Saving a Strategy

To save a new or cloned strategy, or any change of a strategy setting to your project, click **File > Save Project** from the Radiant software main window.

**See Also** ▶"Managing Projects" on page 1

# Setting Active Strategy

Set an active strategy if you have more than one strategy in a project.

**To set an active strategy:**

- Right-click the target strategy and choose **Set as Active Strategy**.

To view the settings of the active Strategy, choose **Project > Active Strategy**. From the submenu, choose the settings to view. The Strategies dialog box opens with the settings you have chosen. You can view or edit the values of the settings.

**See Also** ▶"Managing Projects" on page 1

# Removing a Strategy

Remove a strategy from the design by right-clicking on the target strategy and choosing **Remove**.

**See Also** ▶"Managing Projects" on page 1

# Getting Help on Strategies

To find the online Help description of strategy settings, do one of the following:

- Open the "Strategy Reference Guide" in the Contents pane of the Radiant software Online Help to display all process options in the order they appear in the Strategies dialog box. See "Strategy Reference Guide" on page 380.

- Choose the desired process option. Strategies associated with this process are displayed.
- Use the scroll bar to select a setting and view its description.

- In the Radiant software File List view, double-click a strategy name to open the Strategies dialog box.
  - Highlight a setting to display its brief definition at the bottom of the dialog box, or press **F1** to view additional information for the highlighted option.

**See Also** "Managing Projects" on page 1

- "Strategy Reference Guide" on page 380
- "Creating a Strategy" on page 22
- "Cloning a Strategy" on page 23
- "Adding an Existing Strategy" on page 23
- "Specifying Strategy Options" on page 24
- "Saving a Strategy" on page 25
- "Setting Active Strategy" on page 25
- "Removing a Strategy" on page 25

# Analyzing a Design

You can analyze your design in the following ways while it is being implemented.

- Before synthesis: the Hierarchy view provides a nested list of all modules and commands to access the source code for individual modules, set up test benches, and more. See "Hierarchy View" on page 27.
- After synthesis: the Hierarchy view also provides information on how the modules are using device resources.

After synthesis, schematic views are also available. These views depend on which synthesis tool you are using:

- LSE: you can use the Radiant software's Netlist Analyzer to see and analyze the design.
- Synplify Pro: you can use its HDL Analyst. See *Synplify Pro for Lattice User Guide.*

After mapping, the Hierarchy view provides updated information on how the modules are using device resources. Also, if you are using LSE, Netlist Analyzer shows how the design is mapped to the device's architecture.

**See Also** "Managing Projects" on page 1

# Hierarchy View

The Hierarchy view shows the design hierarchy as a nested list of modules, and launches automatically when you open a project. If the Hierarchy view is not open, choose **View > Show Views > Hierarchy** from the Radiant software main window.

The Hierarchy view also shows the full path name of the files that define each module and, once synthesis has been run, the amount of device resources that each module would consume.

**See Also** ▶"Managing Projects" on page 1

## Commands in the Hierarchy View

Right-click in the Hierarchy view to get access to the following commands.

**Goto Source Definition** This command opens the source editor with the source code that contains the definition for that object, and automatically scrolls the view to the position in the code where the object is defined. For objects that include multi-line definitions, the entire definition is highlighted.

**Goto RTL Definition** After synthesis has been run with LSE, this command opens Netlist Analyzer with a schematic view of the design element.

**Verilog Test Fixture Declarations** This command creates the Verilog test fixture declarations include file (.tfi) based on the Verilog unit selected from the Hierarchy view. This include file should be referenced by your test fixture using: 'include "<*file_name*>.tfi". By including the.tfi file in your simulation test fixtures, you ensure that the design and the test fixtures stay synchronized.

**Verilog Test Fixture Template** This command creates a Verilog test fixture template file (.tft) based on the Verilog unit selected from the Hierarchy view. The template file includes a Verilog design unit with a module declaration. Use the procedure below to create a Verilog test fixture from the .tft file.

**To create a Verilog test fixture from a Verilog Test Fixture Template (.tft) file:**

1. Open the .tft file from the project directory and save it with a .v file extension.
2. Import the new .v file into the project as a Verilog test fixture. Then add codes to the user defined section to complete the stimulus for your design.

**VHDL Test Bench Template** This command creates a VHDL template file (.vht) based on the VHDL unit selected from the Hierarchy view. The template file includes a VHDL design unit with a component declaration and an instantiation based on the entity interface of the first VHDL design unit encountered.

**Note**

Avoid using user-defined record type array in your design, as this may cause the signal width to be incorrect in the VHDL test bench template file generated.Try using standard data type such as std_logic_vector.

**Set as Top-Level Unit** YUse this command to set the selected module as the top-level unit.

**See Also** ▶"Managing Projects" on page 1

## Device Resources

All used device resources are shown to the right of each module. The first number in each column represents the resource used by that module and all of its submodules. The second number, in parentheses, is the number of resources used by that module and any of its submodules. The numbers are updated after the synthesis and map stages of the implementation process are completed. For definitions of the resource types, see Table 3 on page 28.

The Hierarchy view only shows the design and not any Reveal modules. However, when the top module of the design displays a device resource, any Reveal module used is shown as well.

**Table 3: Resource Definitions**

| Hierarchy View Term | Synthesis Report Term | Definition |
|---|---|---|
| LUT4 | OrcaLut, Lut | Four-input look-up tables that can implement Boolean functions with up to four variables. |
| Registers | FD | Sequential elements with clock, enable, and synchronous or asynchronous set and reset. |
| IO Registers | IF, OF | Sequential elements (registers) within input/output blocks. |
| DSP MULT | MULT | Configurable multipliers for high speed operations. |
| DSP ALU | ALU | Configurable adders for high speed addition, subtraction, and logic functions like AND/OR. |
| EBR | SP, DP, PDP | Embedded Block RAM to store large sets of data; configurable for word size and depth. |

**Table 3: Resource Definitions**

| Hierarchy View Term | Synthesis Report Term | Definition |
|---|---|---|
| Distributed RAM | SPR, DPR | RAM distributed with every logic element to store small sets of data such as register files. |
| ROM | ROM | EBR configured as read-only memories. |
| PFUMux | PFU | Dedicated multiplexers that implement 5-input functions using two LUT4 that directly drive the mux inputs. |
| L6Mux | L6M | Dedicated multiplexers that implement 6-input functions using four LUT4 and two PFUMux that drive the L6Mux inputs. |
| Carry cells | CCU | Dedicated logic cells to implement fast carry generation and propagation in adders. |
| SLICE | | Elements with two LUT4 that feed two registers and associated logic that can perform functions such as LUT8. |
| Instantiated | | Anything not covered by the other terms. |

**See Also** ▶"Managing Projects" on page 1

## Changing the Display

Adjust your Radiant software display by re-arranging the columns and sorting the rows. You can also determine what you want to view.

**To change which columns are displayed:**

1. Right-click in any column heading.
2. In the drop-down menu, select the resource that you want to show or hide.

   **Note**: Resources that are not being used by the design are automatically hidden in Hierarchy view regardless of the option settings.

▶

**To sort the modules:**

1. Click in the heading of the desired column.

   The rows are sorted according to the contents of that column.
2. To reverse the order of the rows, click in the same column header again.

**To change the width of columns:**

- ▶ Click on the right-hand border of a column and drag to make that column narrower or wider.

**See Also** "Managing Projects" on page 1

"Commands in the Hierarchy View" on page 27

# About Netlist Analyzer

Netlist Analyzer works with LSE to produce schematic views of your design while it is being implemented. Use the schematic views to better understand the hierarchy of the design and how the design is being implemented.

**Note**

Synplify Pro also provides schematic views.

**To start Netlist Analyzer:**

1. Synthesize the design with LSE.
2. Choose **Tools > Netlist Analyzer**.

   The Netlist Analyzer window opens with the RTL netlist showing.
3. In the window's tool bar, click one of the following buttons:

   RTL View to see the design's logic (gates and registers) after LSE has optimized it.

   Technology View to see the design after synthesis and translated into Lattice primitive modules.

   Open External File to browse to a desired netlist database (.vdb) file.

   A new tab opens and is filled as described below.

**About Netlist Analyzer Views** The Netlist Analyzer window consists of four parts:

- Tool bar provides buttons for various functions.
- Netlist browser provides nested lists of module instances, ports, nets, and clocks.
- Schematic view shows a schematic of the design. Depending on the size of the design, the schematic may consistof multiple sheets.
- Mini-map, which is a miniature view of the sheet, helps you pan and zoom in the schematic view.

Netlist Analyzer can have multiple schematics open. The open schematics are shown on tabs along the bottom of the window.

Bold lines are buses. Green lines are clock signals.

Note that you can adjust the view of a schematic and navigate through the hierarchy in multiple ways.

**See Also** ▶"Managing Projects" on page 1

## Setting Netlist Analyzer Options

With Netlist Analyzer you can specify the following:

- Labels in the schematics
- How the hierarchy gets expanded
- Highlight color
- Size of the sheets

**To change tool options:**

1. Choose **Tools > Options**.

   The Options dialog box opens.

2. In the left-hand list, find and expand **Netlist Analyzer**.

Refer to the following instructions for specific options.

**To specify labels:**

1. Under Netlist Analyzer, select **Text**.
2. Select the labels you want to see in the schematics.

**To create a schematic bookmark:**

1. Under Netlist Analyzer, select **Bookmarks**
2. In the Add Bookmark dialog box, add a name for the bookmark. All bookmarks are associated with a specific .vdb file with a time stamp. This feature allows you to save a schematic graph after a series of operations (such as filter and expand) for future use. This features also allows you to save and restore different critical paths of the design.

**To control how the hierarchy gets expanded:**

1. Under Netlist Analyzer, select **Schematic**.
2. Adjust the values of the following items:
   - Hierarchy Depth: The number of levels to show when using the Dissolve Instances command. The default, 0, displays all levels.
   - Levels of Logic to expand: The number of additional levels to show with the Expand To/From command.
   - Search depth to expand: The number of additional levels to show with the Expand to FF/IO and Expand to Primitive commands. The default, 0, displays all levels.

**To change the colors:**

1. Under Netlist Analyzer, select **Schematic**.

2. Click on the **Select Color** or **Highlight Color** block. The select color is used to outline objects that you select in the schematic view. The highlight color is used to outline objects to keep highlighted after selecting them.

   The Select Color dialog box opens.

3. Select a color by doing one of the following:
   - Click one of the basic colors on the left.
   - Click in the color pallet on the right.
   - Enter values in the boxes at the bottom right. You can work in either column as they are two separate methods of coding color.

   The rectangle in the lower-middle changes to show the selected color. You can change the darkness with the vertical slider at the upper-left.

4. Click **OK**.

   In the Options dialog box, the color block changes to match.

5. Drag the Options dialog box and click **Apply** to see the result of your view change.

**To change the sheet size:**

1. Under Netlist Analyzer, select **Sheet**.
2. Change the width and height as desired.

**See Also** ▶"Managing Projects" on page 1

## Adjusting the Schematic View

Netlist Analyzer schematics are usually displayed on several sheets, depending on the size of the design. More than one schematic view can be open at the same time.

**To change the sheet being displayed:**

- Do one of the following:
  - Click the Next Sheet button.
  - Click the Previous Sheet button.
  - Click the arrow in the Goto Sheet button. In the drop-down menu, choose a sheet number.
  - Click the name at the end of a net on the left or right side of a schematic (but not a port). This takes you to the continuation of the net on another sheet.

**To change the layout in a sheet:**

- Right-click in the schematic and choose **Regenerate**.

**See Also** ▶"Managing Projects" on page 1

## Zooming and Panning

You can zoom and pan within the Netlist Analyzer schematics using a variety of methods, including toolbar commands, dragging with the schematic, and Netlist Analyzer's mini-map.

**Toolbar Commands** The following commands are available on the Radiant software toolbar.

 Zoom In – enlarges the view of the entire layout.

 Zoom Out – reduces the view of the entire layout.

 Zoom Fit – reduces or enlarges the entire layout so that it fits inside the window.

 Zoom To – enlarges the size of one or more selected objects on the layout and fills the window with the selection.

 Pan – enables you to use the mouse pointer to drag the layout in any direction. This command is useful for viewing a hidden area on the layout after zooming in.

**Zooming with Function Key Shortcuts** The following key combinations enable you to instantly zoom in or out from your keyboard.

- Zoom In – Ctrl++
- Zoom Out – Ctrl+-

**Zooming with the Mouse Wheel** The mouse wheel gives you finer zoom control, enabling you to zoom in or out in small increments.

- While pressing the **Ctrl** key, move the mouse wheel forward to zoom in and backward to zoom out.

**Zooming by Dragging** Zoom by holding the right mouse button and dragging:

- To zoom to fit the window, drag up and to the left. The image adjusts to fill the window.
- To zoom out, drag up and to the right. The distance you drag determines the amount of zoom. 1 means half as big, 2 means ¼ as big, and so on. The image is reduced and centered in the window.
- To zoom in, drag down and to the right. The distance you drag determines the amount of zoom. 1 means twice as big, 2 means four times as big, and so on. The image is enlarged and centered in the window.
- To zoom in on a specific area, start at the upper-right corner of the area that you want to see better and drag to the lower-left corner of the area. The area that you drag across is adjusted to fill the window.

**Zooming and Panning with Mini-Map** The mini-map is a small display representation of a schematic that appears in the lower-left corner of the Netlist Analyzer window. The mini-map shows the entire sheet with a purple rectangle marking the area that is actually showing in the schematic view.

If the mini-map is not shown, click the Mini-map button at the lower-right corner of the schematic view.

To enlarge the mini-map, click on one of the control tabs on the edge of the mini-map and drag.

To zoom, click on one of the control tabs of the purple rectangle and drag. As the rectangle changes size, the schematic view zooms in or out.

To pan, click in the purple rectangle and drag it so that it covers the area that you want to see. As you drag, the schematic view shifts.

Click on the spot that you want to see. The schematic view jumps to that spot and centers the view.

**See Also** ▶"Managing Projects" on page 1

## Saving Schematics

You can save a schematic as either a PDF or SVG file. As a PDF file, the whole schematic is a single file with each sheet an 8.5-inch by 11-inch page. As an SVG file, only the currently displayed sheet is saved.

**To save a schematic:**

1. If saving as an SVG file, select the desired sheet.
2. Choose **File >** **Export**.

   The Export Schematic dialog box opens.
3. Browse to where you want to save the file.
4. Enter a file name.
5. Click the **Save as type** drop-down menu and choose PDF or SVG.
6. Click **Save**.

**See Also** ▶"Managing Projects" on page 1

## Printing Schematics

Use the print functions of the Radiant software to print the schematics.

**To print a schematic:**

1. Choose **File > Print Preview**.

   The Print Preview window opens.
2. Expand the Print Preview window to the desired size.
3. To maximize the printout, click for landscape mode.
4. Click the Page setup button and adjust the paper size and margins, if necessary.

5. Click the Print button.
6. Adjust the printer settings if necessary and click **Print**.

**See Also** "Managing Projects" on page 1

# Navigating the Design with Netlist Analyzer

Explore a design by selecting one or more objects and then right-clicking in the schematic view and choosing a command from the drop-down menu. These commands help you to:

- Focus on one part of the design.
- Flatten the layers of a hierarchy.
- Trace the paths between objects.
- Expand selected parts of the schematic.

Most commands can be reversed by clicking the Back button. They can be repeated by clicking the Forward button. To jump back to the original view after using one or more commands, right-click in the schematic view and choose **Restore Current Schematic**.

While exploring a design, you may see a need to set a synthesis constraint. If so, drag the port, net, or register to the relevant tab of LDC Editor. See "Applying Lattice Pre-Synthesis Timing Constraints" on page 123.

**To select objects:**

- Click the Select button and do one of the following:
  - Click on the object in the schematic view. Ctrl-click to select more objects.
  - Click on the object in the netlist browser. Ctrl-click to select more objects.
  - Click and drag to draw a selection rectangle around one or more modules in the schematic view. This method only selects modules, not ports or nets.
  - Right-click in the schematic view and choose **Select All Schematic > Instances**. This command selects all module instances showing on all sheets of the schematic.
  - Right-click in the schematic view and choose **Select All Schematic > Ports**. This command selects all ports displayed on all sheets of the schematic.
  - Right-click in the schematic view and choose **Select All Sheet > Instances**. This command selects all module instances displayed on the current sheet of the schematic.

- Right-click in the schematic view and choose **Select All Sheet > Ports**. This command selects all ports displayed on the current sheet of the schematic.

You can keep an object highlighted, but not actively selected, by right-clicking in the schematic view and choosing **Highlight**. This highlights the selected objects using the selected color while you continue to explore the design. Highlighted objects are not “selected” for commands. Highlighting can be erased by using the Back button.

To erase highlighting and return an object to normal display, select the object and then right-click and choose **Unhighlight**.

**See Also** “Managing Projects” on page 1

## Focusing on Part of the Design

You can focus on a specific part of a design in one of several ways:

- Filter the schematic.
- Reduce it to selected modules.
- Trim away objects that are in the way.
- Request more details of a single module.

**To filter the schematic:**

1. Select one or more modules from either the netlist browser or schematic view.
2. Click the Filter Schematic button.

   The schematic is replaced with the selected modules. Only nets connecting the selected modules are included. Use the commands described below to expand the schematic.

**To filter with signal paths:**

1. Select one or more modules.
2. Right-click anywhere in the schematic view and choose **Isolate Path**.

   The schematic is replaced with the selected modules *and* modules connected to the selected modules. Only nets connecting the modules are included.

**To trim the schematic:**

1. Select one or more objects to hide.
2. Right-click anywhere in the schematic view and choose **Less**.

   The selected objects disappear. If the objects include a branch of a net, the remaining branches become dotted lines to show that part of the net is hidden.

**To see the structure inside a module:**

1. Click the Push down/Pop up button.
2. Drag the cursor to the module.

   If there is further hierarchy to see (that is, if the module is not already a primitive), the cursor changes to a blue down arrow.
3. Click in the module with the blue down arrow.

   The schematic changes to show just the structure of the module at the next lower level of hierarchy. To go further down the hierarchy, continue to click in a module with a down arrow.

To go back up the hierarchy, click a green up arrow when it is not over a module.

To perform other commands in the schematic, click the Select button.

**To quickly push down or pop up in a module:**

Do the following to push down into a module while staying in select mode:

- Right-click the module and drag straight down until the words “push down” appear. Then release the mouse button.
- To go back up the hierarchy, right-click anywhere in the schematic view and drag straight up until the words “pop up” appear. Then release the mouse button.

**To work on one module in RTL view:**

1. Go to the Hierarchy view. If it is not showing, choose **View > Show Views > Hierarchy**.
2. Right-click on the desired module and choose **Goto RTL Definition**.

   Netlist Analyzer opens with the RTL view of the module. The rest of the design is not available in this schematic.

**See Also** “Managing Projects” on page 1

## Flattening the Design

An alternative to viewing a hierarchy is to flatten the design, which displays all the layers together in a larger, more detailed schematic. By doing so, you do not have to keep track of where you are in the hierarchy. You can flatten the entire schematic or just selected modules.

**To flatten the entire schematic:**

- Right-click anywhere in the schematic view and choose **Flatten Schematic**.

  The entire design is shown in terms of primitives.

To return to the original schematic, right-click and choose **Unflatten Schematic**.

**To flatten a module:**

1. Select the module.
2. Right-click and choose **Dissolve Instances**.

   The schematic expands to include the selected module's primitives. Boundaries, ports, labels of the module, and all submodules are included to help identify and select the modules and their ports.

**See Also** ▶"Managing Projects" on page 1

## Tracing Paths between Objects

Use Netlist Analyzer to trace a signal within a complex schematic.

**To trace a signal path:**

1. Select two or more objects on the same or different sheets.
2. Right-click and choose **Expand > Expand Paths**.

   Nets and modules between the selected objects are highlighted. Modules along the paths are expanded to show primitives.

**See Also** ▶"Managing Projects" on page 1

## Expanding from a Module

After focusing a schematic on one or more modules, do the following to see what the modules are connected to.

**To see all the nets attached to a module:**

1. Select the module.
2. Right-click and choose one of the following:
   - **Show Connectivity**

     Highlights all nets attached to the selected module, expanding the schematic as necessary.
   - **Expand > Expand To/From**

     Highlights all nets attached to the selected module and all primitives attached to those nets, expanding the schematic as necessary.

**To see the next higher level of the hierarchy:**

1. Select a module, right-click and choose **Show Context**.

2. If nothing happens, click the Push down/Pop up button. Move the cursor to the schematic view, and click the green up-pointing arrow when it appears.

**See Also** ▶"Managing Projects" on page 1

## Expanding from a Port

Instead of expanding all the nets from a module, you can expand the net from a single port. If the module is not a primitive, you have the choice of expanding outward or inward to a lower level of the module's hierarchy. Make this choice by selecting a port pointing out or in.

**To see what a single port is attached to:**

1. Double-click the port.

   The schematic expands to show the port's net and one other object that is on the net.

2. If the net appears as a dashed line, only part of the net is being shown. Double-click the port again to see another branch of the net.

   If the net shows as solid line, the full net is being displayed.

   If the net is fully visible, nothing else can be done.

**To see a fuller signal path from a single port:**

1. Select the port. If the port is on a module, take care whether it is pointing out or in.
2. Right-click and choose **Expand.**

   A sub-menu drops down.

3. Choose one of the following from the sub-menu:

   - **Expand to FF/IO**

     Highlights the net until it finds a flip-flop or an I/O pin, expanding the schematic as necessary. Modules along the path are expanded to show primitives.

   - **Current Level > Expand to FF/IO**

     Same as above except that modules are *not* expanded.

   - **Expand to Primitive**

     Highlights the net and all attached primitives, expanding the schematic as necessary.

   - **Current Level > Expand to Primitive**

     Same as above except that modules are *not* expanded.

**See Also** ▶"Managing Projects" on page 1

## Expanding from a Net

You can expand a net to show all connected primitives and ports or to show just the primitive or port driving it.

**To see all modules on a net:**

1. Select the net.
2. Right-click and choose **Go to Connected Instances**.

   Highlights the selected net and all connected primitives and ports, expanding the schematic as necessary.

**To see just the driver of a net:**

1. Select the net.
2. Right-click and choose **Go to Driver**.

   Highlights the selected net and the driving primitive or port, expanding the schematic as necessary.

**See Also** ▶"Managing Projects" on page 1

## Searching in Netlist Analyzer

With larger designs, finding objects may be easier with a text search. The Find function in Netlist Analyzer allows you to search for objects in the schematic using instance, symbol, port, and net names. Found objects can be selected in the netlist browser and schematic view.

**To search the design in Netlist Analyzer:**

1. Choose **Edit > Find**.

   The Find dialog box opens.
2. Select the type of object that you want to search for by clicking one of the tabs at the top of the dialog box.
3. Select the part of the design to search for by selecting a level in the Search box.
4. At the bottom-left of the dialog box is the Un-Highlight Search box. Click in this box and type in your search term. You can use the following wildcards:

   - ***** for any number of characters
   - **?** for a single character

   Or, click the arrow at the end of the box and choose from a previously used search term.
5. Click either **Find 200** or **Find All**.

   The found objects are listed in the UnHighlight box.

6. If you clicked Find 200 and want to see more results, click either **Next 200** or **Find All**.
7. To highlight found objects in the schematic view, do one of the following:
   - Select the desired objects in the UnHighlight box and then click the -> (right arrow) button.
   - For all found objects, click the **All ->** button.

   The selected objects move to the Highlight box and are highlighted in the netlist browser and schematic view. The schematic view does not expand to show objects in other layers or on other pages.
8. To reduce the number of objects in the Highlight box, select unwanted objects and then click the <- (left arrow) button. You can find objects in the Highlight list using the Highlight Search box. Anything entered in the Highlight Search box is automatically searched for and selected.
9. When you have the Highlight list the way you want, click **Close**.

   Highlighted objects in the schematic view stay highlighted after the Find dialog box closes.

**See Also** ▶"Managing Projects" on page 1

## Getting More Information about an Object

Additional information about an object in a schematic view is available, including names, number of fanouts, source code, and pins that the nets connect to.

**To get more information about an object:**

1. Select one or more objects.
2. In the schematic view, right-click the object of interest and choose **Properties**.
   - The Properties dialog box opens showing a list of properties for the object.
   - If the object is a module, the dialog box also has a drop-down menu listing all the ports of the module. Choosing a port shows the properties for that port.
   - If the object is a bus, the dialog box has a drop-down menu listing all the nets that make up the bus. Choosing a net shows the properties for that net.
3. Click **Close** when finished.

**To get the hierarchy name of an object:**

1. Select one or more objects.
2. In the schematic view, right-click and choose **Copy**.
3. Paste into any text editor.

The format of the name will be similar to this example:

```
{i:UART_INST/u_txmitt/add_24}
```

The first letter is i for instance, n for net, or p for port.

**To get the source code for an object:**

1. Select one or more objects.
2. In the schematic view, right-click the object of interest and choose **Jump to > Jump to HDL File**.

   The Source Editor opens with the code highlighted.

**See Also** ▶"Managing Projects" on page 1

# Running Processes

A process is a specific task in the overall processing of a source or project. Typical processing tasks include synthesizing, mapping, placing, and routing. You can view the available processes for a design in the Process Toolbar.

**Process Toolbar**


Click the Task Detail View to see detailed information of the processes.

Processes are grouped into categories according to their functions.

- **Synthesize Design**

  Click on this process to have LSE synthesize the design. By default, this process runs the LSE tool. For details, see "Synthesizing the Design" on page 161.

  If you are using Synplify Pro, choose Synplify Pro as the synthesis tool (**Project > Active Implementation >Select Synthesis Tool).** See "Selecting a Synthesis Tool" on page 48 for details).

- **Map Design**

  This process maps a design to an FPGA. Map Design is the process of converting a design represented as a network of device-independent components (such as gates and flip-flops) into a network of device-specific components (configurable logic blocks, for example). For details, see the **Implementing the Design > Mapping** section in the online Help.

  - **Map Timing Analysis**

    Runs Map timing analysis.

- **Place & Route Design**

  After a design has been translated into the Native Circuit Description (.ncd) format, you can run the Place & Route Design process. This process takes a mapped physical design .ncd file, places and routes the

design, and outputs a file that can be processed by the design implementation tools. For details, see "Place and Route" on page 175.

- **Place & Route Timing Analysis**

  Runs Place & Route timing analysis.

- **I/O Timing Analysis**

  Runs I/O timing analysis.

- **Export Files**

  You can check the desired file you want to export and run this process.

  - **Bitstream File**

    This process takes a fully routed physical design or .ncd file as input and produces a configuration bitstream (bit images). The bitstream file contains all of the configuration information from the physical design defining the internal logic and interconnections of the FPGA, as well as device-specific information from other files associated with the target device. For details, refer to "Bit Generation" on page 190.

**See Also** "Managing Projects" on page 1

- "Managing Project Sources" on page 9
- "Process State" on page 43
- "Starting a Process" on page 44
- "Forcing a Process to Run" on page 45
- "Stopping a Process" on page 45
- "Refreshing Process State" on page 45
- "Cleaning Up Processes" on page 46

# Process State

The Process Toolbar shows the state of the processes in the active implementation. Each state is identified with different icons, as shown in the table below.

| Process State | Process Toolbar Icon | Description |
|---|---|---|
| Initial |  | The process has never been run or an input dependency time stamp has changed since it was last run. |
| Completed |  | The process has been run successfully. |
| Completed |  | Process has been completed with unprocessed subtask(s) |
| Error |  | The process did not complete successfully. |

**Conditions that Re-initialize Process State** A process or report is re-initialized under the following conditions:

- A process state earlier in the sequence has changed.
- An input file of the process or report has changed since the last run. For example, changing the .pdc file will cause the map process to be rerun.

  **Note**

  Input file changes are only detected if done by the Radiant software Source Editor or a process or report triggered by the Radiant software main window. Changes applied by third-party text editors will NOT be detected.

- Process settings have changed.
- Target device has changed. All processes will be re-initialized.
- A **Force Run** function is run.

**See Also** "Managing Projects" on page 1
- "Running Processes" on page 42
- "Starting a Process" on page 44
- "Forcing a Process to Run" on page 45
- "Stopping a Process" on page 45
- "Refreshing Process State" on page 45
- "Cleaning Up Processes" on page 46

# Starting a Process

You can start a process on a single source file or on an entire project.

**To start a process, do one of the following:**

- In the Process Toolbar, click the desired process.
- Right-click the process in the Process Toolbar and choose **Run**.

**See Also** "Managing Projects" on page 1
- "Running Processes" on page 42
- "Process State" on page 43
- "Forcing a Process to Run" on page 45
- "Stopping a Process" on page 45
- "Refreshing Process State" on page 45
- "Cleaning Up Processes" on page 46

# Forcing a Process to Run

If the process is up-to-date (indicated by a check mark to the left of the process), it will not run again. However, you can force a process to run by doing the following:

- Right-click the process in the Process Toolbar and choose **Force Run**.

**See Also** "Managing Projects" on page 1

- "Running Processes" on page 42
- "Process State" on page 43
- "Starting a Process" on page 44
- "Stopping a Process" on page 45
- "Refreshing Process State" on page 45
- "Cleaning Up Processes" on page 46

# Stopping a Process

Do the following to stop running a certain process:

- Right-click the process in the Process Toolbar and choose **Stop**.

**See Also** "Managing Projects" on page 1

- "Running Processes" on page 42
- "Process State" on page 43
- "Starting a Process" on page 44
- "Forcing a Process to Run" on page 45
- "Refreshing Process State" on page 45
- "Cleaning Up Processes" on page 46

# Refreshing Process State

You can refresh the process state by choosing **Design > Refresh Design**. This will also update the Hierarchy view and Reveal Inserter with any design changes.

**See Also** "Managing Projects" on page 1

- "Running Processes" on page 42
- "Process State" on page 43
- "Starting a Process" on page 44
- "Forcing a Process to Run" on page 45
- "Stopping a Process" on page 45
- "Cleaning Up Processes" on page 46

## Cleaning Up Processes

Reset all processes by right-clicking a process in the Process Toolbar and choosing the **Clean Up Process** command. This command returns the status of all of the processes back to their initial state.

**See Also** "Managing Projects" on page 1

# Clearing Tool Memory

The Clear Tool Memory command, available from the Tools menu, clears the device, design, and preference information from system memory. Clearing the tool memory can speed up memory-intensive processes such as place-and-route. When your design is very large, it is good practice to clear memory prior to running place-and-route.

If you have open tool views that are affected by clearing the tool memory, a confirmation dialog box opens to give you the opportunity to cancel the memory clear.

**To clear tool memory:**

1. In the Radiant software main window, choose **Tools > Clear Tool Memory**.
2. In the dialog box, choose the tool you wish to clear memory information from.
3. Click **OK**.

For more information about shared memory, refer to the *Lattice Radiant Software User Guide*.

# Setting Options for Synthesis and Simulation

The Radiant software is integrated with LSE, the Synplify Pro synthesis tool, and the Active-HDL simulation tool. Besides the OEM tools, you can also let the Radiant software use LSE or Synplify Pro as the synthesis tool, and full-featured versions of ModelSim or Active-HDL as the simulation tool.

This section covers steps on how to do the following:

- Selecting synthesis and simulation tools
- Defining file order for synthesis and simulation
- Customizing synthesis tool processes
- Specifying VHDL library name
- Specifying the search path for Verilog include files

Performing interactive synthesis is also discussed in this section.

**See Also** "Managing Projects" on page 1

# About Lattice Synthesis Engine

For most devices, LSE can be used as your synthesis tool instead of Synplify Pro for Lattice or any other third-party synthesis tool.

LSE is a synthesis tool custom-built for Lattice products and fully integrated with the Radiant software. Depending on the design, LSE may lead to a more compact or faster placement of the design than another synthesis tool. LSE can be run from the Radiant software main window or through the command line, similar to the other integrated synthesis tools.

In addition, LSE offers the following advantages:

- Enhanced RAM and ROM inference and mapping, including:
  - Multiple reads
  - Dual-port RAM in write-through, normal, and read-before-write modes mapped to EBR
  - Clock enable and read enable packing
  - Mapping for the minimal number of EBR blocks
  - EBR mapping for minimal power
  - Better support for wide-mode mapping
- Comprehensive GSR inference for area and timing.
- Post-synthesis Verilog netlist suitable for simulation.

When considering LSE, note the following limitations:

- IP must be generated with a synthesis tool other than LSE. So the synthesis of IP won't be affected by a change to LSE.
- Design code may need to be modified. Modifications may involve some VHDL requirements and inferring RAM, ROM, and I/O.
- SystemVerilog features are not supported.
- Does not include interclock domain paths in timing analysis.

**See Also** "Managing Projects" on page 1

- "Selecting a Synthesis Tool" on page 48

## Switching to Lattice Synthesis Engine

When changing to LSE from another synthesis tool, the following must be done:

- Consider creating a new design implementation. You can experiment with different settings without losing your previous work. See "Working with Implementations" on page 18.

- Modify the Synopsys Design Constraint (.sdc) file into an .ldc or create a new .ldc file. See “Lattice Synthesis Engine Constraints” on page 416 and “Applying Lattice Pre-Synthesis Timing Constraints” on page 123.
- Adjust the LSE strategy settings. See “LSE Options” on page 388 and “Optimizing LSE for Area and Speed” on page 164.
- Review the LSE coding tips and consider applying them to your design. See “Coding Tips for LSE” on page 68.
- Review the instructions for integrated synthesis. There are some differences with LSE. See “Integrated Synthesis” on page 166.

**See Also** “Managing Projects” on page 1

## Bus Naming in LSE Output

In the LSE output files, bus names are converted to individual signal names (sometimes known as “bit blasting”) and “_c_” is added. For example, the bus name sum[0:3] becomes:

sum_c_0
sum_c_1
sum_c_2
sum_c_3

**See Also** “Managing Projects” on page 1

# Selecting a Synthesis Tool

The Radiant software supports Verilog HDL and VHDL designs using LSE or Synplify Pro as the synthesis tool.

- The Radiant software is fully integrated with LSE and Synplify Pro. “Fully integrated” means that you can set options and run synthesis entirely from within the Radiant software.
- If you prefer, you can use other synthesis tools by running them independently of the Radiant software.

You can specify one of them as the synthesis tool for the currently active implementation of your project. Different implementations can have different synthesis tools specified.

**To select a synthesis tool:**

1. Make sure you have successfully installed LSE or Synplify Pro.
2. From the Radiant software main window, choose **Project > Active Implementation > Select Synthesis Tool**.

   The Project Properties dialog box appears with the active implementation selected.

3. In the Project Properties dialog box, double-click the synthesis tool in the Value column.
4. From the drop-down menu, select a tool.
5. Click **OK**.

If you have chosen **Synplify Pro**, you can either use Synplify Pro for Lattice or your own full-featured Synplify Pro. By default, the Radiant software uses the Lattice version. If you want to use Synplify Pro for Lattice, you can use the main window to start running processes. If you want to use your own full-featured Synplify Pro, you need to specify the directory it is in by doing the following:

1. Choose **Tools** > **Options** > **General**.

   The Options dialog box opens.
2. In the **Directories** tab, clear the **use OEM** option.
3. Specify the installation path for Synplify Pro in the Synplify Pro field.
4. Click **OK** to save your settings.

If you have a different synthesis tool, you need to specify the directory in the Options dialog:

1. Choose **Tools** > **Options** > **General**. The Options dialog box opens.
2. In the **Directories** tab, specify the installation path for the Synthesis field.
3. Click **OK** to save your settings.

**See Also** ▶"Managing Projects" on page 1

▶ "Customizing Synthesis" on page 53

# Specifying VHDL Library Name

The Radiant software supports VHDL design synthesis using LSE Synplify Pro. You can specify the library name for synthesizing individual VHDL sources and modules. By default, all project-related design sources are compiled into the "work" library.

The VHDL library name can also be specified in the Synplify Pro tool. See Synplify Pro Help for details.

**To specify the VHDL library target in the main window:**

1. In the File List view, right-click the VHDL source for which you want to specify a library name and choose **Properties**.
2. In the Project Properties dialog box, type in the library name you want in the Value field and click **OK**.

   The LSE Synplify Pro tool uses the specified library name to synthesize the selected VHDL source file or module.

**See Also** ▶"Managing Projects" on page 1

# Specifying Search Path for Verilog Include Files

The Radiant software supports Verilog design synthesis using LSE, or Synplify Pro. If a Verilog file is added to your design and additional files are referenced using the include directive, you can specify the search path in the Radiant software for searching include files.

**Note**

If the include file in your Verilog file changed after you opened the relevant project in the Radiant software, you need to use the **Rerun All** or **Rerun** command to force the process to rerun so that the Radiant software can reflect the changes of the include file.

You can also specify a Verilog include file search path directly in LSE, or Synplify Pro. See “Running SYNTHESIS from the Command Line” on page 570, and the Synplify Pro Help for details.

**Synplify Pro and LSE Search Path** If you select Synplify Pro or LSE as the synthesis tool, you can only specify the search path for the entire project. Synplify Pro or LSE searches for the include file in the following order, and stops at the first occurrence of the include file it finds.

1. The directory of the file that specifies the 'include' directive.
2. The directories that are specified in the “Search Path” option for the entire project.

### Specifying Search Path in the Radiant software

**To specify the search path in the Radiant software:**

1. In the File List view, right-click the implementation name if you want to specify the search path for the entire project.
2. Choose **Properties** from the right-click menu.

   The Project Properties dialog box opens.
3. In the dialog box, select **Verilog Include Search Path**, and do either of the following:
   - To add a single search path, click the Value field to enter a path.
   - To add multiple paths, click the **...** button from the Value field to get the Verilog Include Search Path dialog box. Next, click the  icon to browse for a new path and click **OK**. The new path is displayed in the Verilog Include Search Path dialog box. Repeat to add more paths.

     You can use the  and  button to arrange the order of the paths, or use the  button to delete a path. Click **OK** when finished.
4. Click **OK** to exit the Project Properties dialog box.

The search paths you added are applied. The Synplify Pro tool uses the specified search paths to search for the include files referenced in your design other than the directory of the file that specifies the include directive.

**See Also** ▶"Managing Projects" on page 1

# Specifying Verilog and VHDL Parameters

You can add Verilog compiler directives and parameters or VHDL generics for the entire design in the properties of each design implementation. You may find this method an easier way to experiment with these settings.

**To add Verilog compiler directives and parameters or VHDL generics:**

1. In the File List view, right-click the implementation's name and choose **Properties**.

   The Project Properties dialog box opens showing the implementation's properties.
2. In the HDL Parameters row, click under **Value**.
3. Type the statements as a single text line with different statements separated by semi-colons (;).
4. When done, click **OK**.

**See Also** ▶"Managing Projects" on page 1

# Selecting a Simulation Tool

The Radiant software supports functional and timing simulation for Lattice FPGA devices using ModelSim/Questa Sim or Active-HDL. You can specify either one as the simulation tool for your project in the Simulation Wizard (**Tools >** **Simulation Wizard** from main window). For more details on how to use Simulation Wizard, see "Simulating the Design" on page 96.

**See Also** ▶"Managing Projects" on page 1

# Specifying Input Files for Simulation

If your design includes one or more test bench files to use in your simulation but you don't want to synthesize them, you can specify this in the properties for each input file. New files are automatically marked for both synthesis and simulation.

**To change the synthesis/simulation property of a file:**

1. In the File List view, right-click the filename.
2. In the drop-down menu, choose **Include for** and then the desired property:
   - Synthesis and Simulation
   - Synthesis
   - Simulation

You can also set the synthesis/simulation property in the Project Properties dialog box. You can open the dialog box by choosing **Project > Property Pages**.

**See Also** ▶"Managing Projects" on page 1

# Defining File Order for Synthesis and Simulation

The File List view lists design input files in a specific order. The synthesis or simulation tool uses that order to sequentially synthesize or simulate each input file. You can adjust the file order by the "drag and drop" operations. The file list can include a combination of VHDL and Verilog files. If you have not manually set a top-level module (in the Project Properties dialog box of the main window), the last file on the file list is normally treated as the top-level module.

File order is especially important for VHDL files. Package files must be first on the list because they are compiled before they are used. If design blocks are spread over several files, specify them in the following file order:

1. The file containing the entity,
2. The architecture file.
3. The file with the configuration.

Verilog file order is important when define statements are dedicated to a single file. This file should be compiled prior to any files that refer to the variables.

**To define the file order for logic synthesis and simulation:**

- In the Input Files section of the File List view, change the file order as you like by dragging the file name and dropping it to the desired location.

  The new file order takes effect immediately. However, the order will only be saved once you save the design project.

**See Also** ▶"Managing Projects" on page 1

## Customizing Synthesis

The Radiant software supports Verilog and VHDL synthesizing using LSE, and Synplify Pro.

You can adjust synthesis tool options and perform the logic synthesis process for your Verilog or VHDL design within the Radiant software.

**To adjust synthesis tool options and perform the synthesis process:**

1. Choose **Project > Active Strategy >** <*synthesis tool*> **Settings**.
2. The option settings of the selected process, LSE, or Synplify Pro are listed by default in the Strategies dialog box.
3. Double-click the Value field of the option. You can specify the option using the drop-down menu of the Value field.
4. Click **OK** to accept the new synthesis tool options and exit the Strategies dialog box.
5. In the Process view, double-click the Synthesize Design process to execute the selected synthesis tool.

If you prefer leveraging all the interactive features available in the synthesis tool while maintaining HDL source in the main window, see "Synthesizing the Design" on page 161 for more details.

**See Also** ▶"Managing Projects" on page 1

# Finding Results

After you load a design in the Radiant software, you can find the information you need by doing the following.

In the active Reports view:

1. Choose **Edit > Find**. Type the desired text into the Find field at the bottom-left of the Reports view window that opens. The first occurrence of the desired text is automatically found and highlighted. Select the **Match Case** option if needed for the search, as text is case-sensitive. While typing, the Find field will turn red if no matches are found.
2. Click **Next** or **Previous** to find more.

In the active Source Editor:

1. Select **Edit > Find**, to open the Find and Replace dialog box. Enter the text to search for in the Find What field. Check **Match Case**, **Match Whole Word**, **Search Up**, **Regular Expression** options as needed.
2. Use the **Replace** or **Replace All** command to replace the text found.
3. Click **Next** or **Previous** to find more.

If you want to find information without loading the files, do the following.

1. Choose **Edit > Find in Files** from the Radiant software main window.
2. Type the text to find in the pop-up Find In Files dialog box.
3. Specify the search path and search filters.
4. Select the search parameters desired: **Search subdir**, **Include hidden files**, **Match case**, **Match whole word**, and **Regular expressions**.
5. Press **Find**. The results are displayed in the Find Results frame. Double-click any of the findings from the Find Results frame to open the associated source file in the associated editor. For example, if the finding is in a log file, the log file will be opened in the Reports view with the first finding appears on the first line.

You can search the Output log by clicking in the Output View (in the text, not on the tab) and then pressing **Ctrl**-**F**. This opens a basic text search dialog box at the top of the Output View. You can type the text in the Find text field and start a search.

**Find Results View** If the Find Results are not automatically displayed in your Radiant software main window, select **View > Show Views > Find Results** to display them.

The Find Results view can be detached from the main window by clicking the detach icon on the upper-right corner of the view. After detaching, you can double-click on the title bar of the view to re-attach it back to the main window.

**See Also** ▶"Managing Projects" on page 1

- "Viewing Logs and Reports" on page 54
- "Navigating Errors and Warnings" on page 55
- "Finding Text in Logs and Reports" on page 56

# Viewing Logs and Reports

The Radiant software automatically generates log files for all project activities. The log files contain processing information, as well as error and warning messages. If you run processes, reports are generated.

**Viewing Logs** A log file is displayed in the Output frame as a process is running. A scroll bar can be used to scroll up and down to view all log information.

Errors are displayed in red, while warnings are displayed in orange. A check mark indicates the view is displayed.

**Viewing Reports** The Reports view displays reports for the major processes.

There are two panes in the Reports view. The left pane lists the report types. The reports in detail are displayed in the right pane.

| Type of Report | Description |
|---|---|
| Project Summary | Lists the summary information of the project. |
| Synthesis Report | Lists LSE, LSE Timing Report, and Synthesis Resource Usage reports in HTML format. |
| Map Report | Lists Map, Map Timing Analysis, and Map Resource Usage reports in HTML format. |
| Place & Route Report | Lists Place & Route, Signal/Pad, and Place & Route Timing Analysis reports in HTML format. |
| Export Report | Lists Bitstream report in HTML format. |
| Misc Report | Lists Tcl Command Log in HTML format. |

**Other Reports** The Synthesize Design stage produces reports that do not appear in the Reports view. You can find these reports in the implementation folder by right-clicking the implementation name in the File List view and choosing **Open Containing Folder**. A window will open showing the contents of the folder. All of these reports can be read with a text editor.

One of the reports is a detailed description of the device resources that will be used by the design. This report is much more detailed than the synthesis report in the Reports view, as it includes the resources used by each module of the design. Similar information can also be found in the Hierarchy view. For Synplify Pro, look for <*top_module*>.areasrr; for Lattice Synthesis Engine, <*top_module*>.arearep.

**See Also** ▶"Managing Projects" on page 1

▶ "Finding Results" on page 53

# Navigating Errors and Warnings

If an error or a warning results from the specific line in an HDL source file, you can easily go to that line to edit the source file.

**To navigate errors and warnings:**

- In the Output View, located in the lower right of the Start Page, double-click the line describing the error or warning.
- Right-click the message and choose **Location in > Text Editor**. If the command is dimmed, there is no link to a source file. Depending on the message, more than one tool may be available to view the source.

Your default text editor opens with the appropriate HDL source file at the line number specified in the error or warning message. You can then modify the file to debug your design.

**See Also** "Managing Projects" on page 1

"Finding Results" on page 53

# Finding Text in Logs and Reports

You can search for text in the Output View.

**To search for text in the Output View:**

1. Right-click in the Output View and choose **Find in Text**.

   The Find toolbar appears.

2. In the **Find** text box, enter the text to search for. Check the **Match Case** or **Match whole word only** options as needed. As you type each character, the window displays any matches found.

3. Use the **Next** and **Previous** buttons in the Find toolbar to find other occurrences of the text.

**To search for text in the Reports view:**

1. Choose **Edit > Find**.

2. In the Find text box, enter the text to search for. Check the **Match Case** option if needed.

**See Also** "Managing Projects" on page 1

"Finding Results" on page 53

# Filtering Messages

The number of messages produced by the design implementation process can be very large. There are several ways to filter these messages to find the ones of most interest. Messages can be filtered by:

- Implementation process
- Severity (information, warning, and error)
- ID number
- Text

**To set filter options:**

As filters are added, the message list changes to show the latest results.

1. After running part or all of the implementation process, choose **View > Show Views > Message** if message view isn't displayed.

2. Select the Messages Tab at the bottom of the screen.

3. Click on one of the following icons: **Show Error** ( ), **Show Warning**( ), or **Show Information** ( ).

Any messages found appears in a spreadsheet-like display.

4. Specify any message ID to hide by right-clicking the desired message ID and choosing **Filter > Filter Messages with This ID**.
5. To hide a specific group of messages, right-click on a message and choose **Filter > Filter Messages Exactly Like This**, **Filter Messages from this Process**, or **Filter Messages with this Severity**. This hides all messages with the same text, process, and severity.

   To list only messages that have been hidden, right-click in the Message area and choose **Filter > Filter Details List**. The Detailed Filter List dialog box opens showing a list of the hidden messages.
6. To unhide messages, right-click in the Message area and choose **Filter > Filter Details List**. From the Detailed Filter List dialog box, uncheck the box in the Filter Enable column of the message(s) to unhide and click **Apply.**

# Changing Warnings to Errors

There may be warning messages in your design project that you want to place emphasis on. You can do so by changing these warnings to show the error severity level. You can also save these “promotions” to use in other design projects.

Promotions only show for messages with future actions. Promotions do not affect the display of messages from past actions.

**Note:**

After changing a warning to an error, the original flow will fail during the rerun process.

**To change a warning to an error in the Messages Tab:**

1. Right-click on the message to highlight and choose **Change Severity with with This ID to**.

   Not all warning messages can be highlighted in this way, so this command may be dimmed.

**To change a warning to an error without the Reports view:**

1. Choose **Project > Message Promotion/Demotion**.

   The Message Promotion dialog box opens.
2. In the IDs box, type the ID number of the message(s) that you want to promote. Separate multiple ID numbers with semi-colons (**;**).
3. Click **Promote**.

   Not all warning messages can be promoted, so some ID numbers may be rejected.

4. Click **Close** when finished.

**To restore a message ID to the warning level:**

1. Choose **Project > Message Promotion/Demotion**.

   The Message Promotion dialog box opens.

2. Select the message to restore.
3. Click **Remove**.
4. Click **Close** when finished.

**To save the list of promoted messages for another project:**

1. Choose **Project > Message Promotion/Demotion**.

   The Message Promotion dialog box opens.

2. Click **Export**.
3. In the Export Promotion dialog box, browse to where you want to save the list.
4. Enter a file name.
5. Click **Save**.

   The list is saved to a file with a ".pmt" extension.

6. Click **Close** when finished.

**To use an existing list of promoted messages:**

1. Choose **Project > Message Promotion/Demotion**.

   The Message Promotion dialog box opens.

2. Click **Import**.
3. In the Import Promotion dialog box, browse to the list you previously saved and want to import.
4. Click **Open**.

   The list of messages appears in the Message Promotion dialog box.

5. Click **Close** when finished.

**See Also** ▶"Managing Projects" on page 1

# Getting Help for Messages

Some messages offer a link to online Help that provides additional information. This information may help you better understand an error message or how to fix a specific problem.

**To check for help on a message:**

1. Right-click on the message in the Output View.

A menu appears. The last item in the menu is Help. If it is dimmed, there is no help link.

2. If the Help command is available, click on it.

   The Help opens with the appropriate topic.

**See Also** ▶"Managing Projects" on page 1

# Setting Tool and Environment Options

The Radiant software provides many environment control and tool view display options that enable you to customize your settings.

**To access these options:**

Choose **Tools > Options**.

The Options dialog box is organized into functional folders.

Commonly configured items include:

- General settings -- allows you to set some common options, including.
  - Startup – enables you to configure the default action at startup and also to control the frequency of checking for software updates.
  - File Associations – allows you to set the programs to be associated with different file types based on the file extensions.
  - Directories – Set directory location for Synthesis and Simulation tools.
  - Network Settings – Apply a proxy server and specify a host and port.
- Color settings -- allows you to set colors for such GUI features as fonts and backgrounds for various Radiant software tools, including:
  - General
  - Device
  - Design
  - Group
  - Reveal Analyzer
  - Source Editor

  You can also change the Theme color of Radiant software (Dark or Light) using the Themes drop-down menu.
- Tools settings -- allows you to set options for various Radiant software tools, including the Device Constraint Editor, Netlist Analyzer, Timing Constraint Editor, and Source Editor. You can also to check Constraint design rule checking (DRC), prior to saving or when launching a tool.

# Entering the Design

You can enter your design, or describe the logic of our design, in several different ways and you can use the ways in almost any combination. Radiant software accepts, and helps create, Verilog, VHDL and SystemVerilog files. Design files can be created using Synplify Pro.

Setting specifications to control synthesis or simulation or to control how the design is implemented in hardware are treated as a separate topic. You will almost always want to at least start entering your design before setting constraints.

**Verilog and VHDL** The hardware design languages (HDL) Verilog and VHSIC Hardware Description Language (VHDL) are the usual ways to describe complex logic designs. You can create HDL files using you preferred HDL editor and then add them to your Radiant software design project, or you can create or edit them within the Radiant software.

**Modules** Modules are functional bits of design that can be re-used wherever that function is needed. To help your design along, the Radiant software provides a variety of modules for common functions. They are optimized for Radiant software device architectures and can be customized. Use these modules to speed your design work and to get the most effective results. See “Designing with Modules and PMI” on page 84.

**Structural Verilog** Lattice Synthesis flow will by default generate a netlist which is a Structural Verilog (.vm) file. This .vm file can be used by itself as a whole design, or it can be referenced as an IP module to be used in another design. Any Verilog file -- Structural or Behavioral (.v or .vhd) -- can be used for design entry.

# HDL Design Entry

The Radiant software supports HDL design including pure VHDL design, pure Verilog design, and mixed VHDL and Verilog HDL design. You can use the integrated Source Editor to create and edit your HDL source files and any text-based files.

Source Editor is a text entry tool built into the Radiant software. You can use this tool to create and edit text-based files, such as HDL files, preference files, script files, test files, and project documentation files. Like many other advanced editing tools, it supports multiple file editing, wheel mouse scrolling, as well as popup menus for cut, copy, and paste operations. If a data source changes since the view was initialized, the view will detect this condition and provide you an option to refresh it.

Designed particularly for HDL file editing, Source Editor highlights the std_logic keywords and displays different HDL syntax elements with different colors. The color scheme can be customized in the Options dialog (**Tools > Options** from the main window). Also, Source Editor enables you to check correct pairing of parenthesis constructs and offers rich template features.

**Note**

Do not leave blank spaces in file names when you save source files in Source Editor, otherwise the files could fail to generate the database.

If you are going to use Lattice Synthesis Engine (LSE) to synthesize the design, there are helpful Verilog and VHDL coding tips in "Coding Tips for LSE" on page 68.

**See Also** "Running Source Editor" on page 61

"Viewing Logs and Reports" on page 54

## Running Source Editor

You can run Source Editor in several ways.

**To run Source Editor, do one of the following:**

- From the File List view, double-click a language file—for example, filename.vhd
- From the Radiant software main window, choose **File > New > File**. In the New File dialog, choose a text-based source, for example Verilog Files. Fill in the File name and Location, click **New**.

# Creating HDL Source Files in Source Editor

You can create an HDL source file in Source Editor.

**To create an HDL source in Source Editor:**

1. From the Radiant software main window, choose **File > New > File**. In the New File dialog, choose Verilog Files or VHDL Files from the Source Files list.
2. In the pop up New File dialog, fill in the Name and Location, and choose the file extension in the Ext field.
3. Check the **Add to Implementation** option if you want to add this source to the current project.
4. Click **New**.
5. In the pop up Source Editor, you can enter the text. When finished editing, click **File > Save** from the Radiant software main window.

**Tip**

You can detach Source Editor from the Radiant software main window by clicking the Detach Tool icon on the upper right corner of Source Editor. If you want to attach Source Editor back to the main window, click the Attach Window icon on the upper right corner of Source Editor window, or choose **Window > Attach Window** from Source Editor.

# Editing Text Files

This section describes the editing functions in Source Editor.

## Editing a Column of Text

You can cut, copy, and paste a column of selected text in Source Editor. You can also use the Tab key to increase the left indention of the selected column.

**To cut, copy, and paste a column of text:**

1. Hold down **Alt** and then press and hold the left mouse button. Move the mouse over the text you want to copy. The selected text becomes highlighted in blue.

   **IMPORTANT!**

   This feature does not work on a Linux platform, because the **Alt** key is occupied by the Linux system.

2. Use the right-click commands or the commands on the Edit menu to cut, copy, and paste the selected text.

**To indent a column of text:**

1. Place the cursor in front of the text you want to indent.
2. Choose **Edit > Advanced > Indent**. Or, press **Tab**.

   Source Editor increases the indent of the selected text column by one tab size. By "Matching Braces, Parenthesis, and Brackets" on page 63, you can change the position of the indent.

**To uppercase a selection:**

1. Highlight the text you want to change to uppercase.
2. Choose **Edit > Advanced > Uppercase Selection**. You will see the highlighted text become uppercase.

**To lowercase a selection:**

1. Highlight the text you want to change to lowercase.
2. Choose **Edit > Advanced > Lowercase Selection**. You will see the highlighted text become lowercase.

**To comment text:**

1. Highlight the text you want to add comment, or place the cursor in front of the text.
2. Choose **Edit > Advanced > Comment/Uncomment**. You will see "--" put ahead of the text (VHDL file only). You can add comment after "--" (VHDL file only).

### Matching Braces, Parenthesis, and Brackets

Source Editor monitors brace ( {} ), parenthesis ( () ), and bracket ( [] ) constructs. You can match an opening brace (or bracket or parenthesis) with a closing one, and vice versa.

**To jump to the matching brace, parenthesis, or bracket:**

1. In Source Editor, place the editing cursor adjacent to an opening or closing parenthesis (or bracket or brace).
2. Choose **Edit > Advanced > Match Brace**.

   The cursor will move to the corresponding opening or closing parenthesis (or bracket or brace). This allows you the check for the balanced braces around functions and statements.

You can use the **Edit > Advanced > Select to Brace** command to force the matching brace, parenthesis, or brackets to be highlighted along with the text in between the two braces, parenthesis, or brackets,

### Using Bookmarks

You can insert bookmarks into the source file to mark places where you need to enter variables. Afterwards, each time you load the file, you can use the **Edit > Bookmarks** command to jump to those marks to enter or edit your variables.

**To insert bookmarks in a file:**

1. In Source Editor, open the file you want to insert bookmarks into.
2. Click the location in the file you want to insert a bookmark.
3. Choose **Edit > Bookmarks > Toggle Bookmark**.

   A mark is inserted at the beginning of the line where the cursor points.
4. Repeat step 2 and 3 to add more marks, if desired.

**To find marks in a file:**

1. In Source Editor, open the file where the bookmarks you want to find reside.
2. Choose **Edit > Bookmarks > Previous Bookmark**, or **Edit > Bookmarks > Next Bookmark**.

   The cursor jumps to the bookmark in the direction selected. You can start typing from the desired marked place to enter or edit the variable.
3. Repeat step 2 to find, enter, or edit more bookmarks, if desired.

### Using AutoComplete

When you type a keyword into Source Editor, choose **Edit > Advanced > AutoComplete**. A list of the matching values are presented. You can then:

- Choose the first matching value by pressing **Enter** directly.
- Choose an alternate matching value by using the up/down arrow keys and then pressing **Enter**.
- Continue typing the rest of the value. If no matching value is found, nothing will be entered.

# Using Templates

Source Editor provides templates for creating VHDL, Verilog, and constraint files, or you can create your own. Templates increase the speed and accuracy of design entry.

**Note:**

Remember to replace dummy variables in the template with your own logic.

You can use Source Editor's **View > Template Editor** menu to locate and access template files. After clicking this menu, the Template Editor pane appears in the upper left of Source Editor. Templates are available for:

- Verilog
- VHDL
- LDC (Constraints for LSE.)

The following templates are available for each of the above categories:

**Table 4: Templates Available in Source Editor**

| Category | Template Folders |
|---|---|
| Verilogy/VHDL | ▶ Common Templates<br>▶ PMI Templates<br>▶ Primitive Templates<br>▶ User Templates |
| LDC | ▶ System Templates<br>▶ User Templates |

For LDC, System Templates holds templates for the supported constraint statements. User Templates allows you to create your own templates for future use.

**See Also** ▶"Lattice Module Reference Guide" on page 483

▶ "Lattice Synthesis Engine Constraints" on page 416

## Inserting a Template into Your Source File

You can insert templates into a source file you are editing with Source Editor.

**To insert a template into your source file:**

1. Open the source file in Source Editor. See "Running Source Editor" on page 61.
2. Choose **View > Template Editor**.

   Two additional panes open in Source Editor. The upper pane shows the list of template folders. The lower pane is blank.
3. Place the cursor in your source document where you want to insert a template. This should be in a blank line.
4. Select the template you want to use in the template list.

   The code of the template appears in the lower pane.
5. Right-click the template name and choose **Insert to text**.

   The selected template is inserted into the file after the cursor.
6. Be sure to replace all placeholders with real values. Placeholders are terms between angle brackets such as "<event_expression>."
7. If needed, make further changes such as adding code to execute within a case statement.

**See Also** ▶"Instantiating a PMI Module" on page 93

## Creating a New Template

You can create new templates for your own use and save them into the User Templates folders. The new templates will be available for all implementations in the design project.

You can also create sub-folders to organize the templates that you create. If you plan on creating many templates, think about how to organize them before you start. There is no easy way to move templates once they have been created.

**To create a new template:**

1. Open a source file in Source Editor. This can be a file that you will use the new template in or a blank file that you will delete after creating the template. See "Running Source Editor" on page 61.
2. Choose **View > Template Editor**.

   Two additional panes open in Source Editor. The upper pane shows the list of template folders. The lower pane is blank.
3. In Template Editor, expand the appropriate folder so that its User Templates folder is showing. Every top level folder has a User Templates folder. If the User Templates folder already has contents, expand it so you can see the contents.
4. If you want to create a sub-folder, right-click **User Templates** or one of its sub-folders and choose **New Folder**. Then right-click the new folder and choose **Rename**. Type the desired folder name and press **Enter**.
5. Right-click **User Templates** or one of its sub-folders and choose **New Template**.

   A new template icon appears under the folder.
6. Right-click the new template icon and choose **Rename**. Type in the desired template name and press **Enter**.
7. Type the code for your template in the lower pane of Template Editor.

   The new template is automatically saved.

## Deleting a User Template

You can only delete templates from a **User Templates** folder.

There is no undo command, so once a template is deleted it is permanently gone. So ensure you have chosen the correct template before deleting it.

**To delete a template:**

1. Open a source file in Source Editor. This can be a file that you are using or a blank file that you will delete after creating the template. See "Running Source Editor" on page 61.
2. Choose **View > Template Editor**.

Two additional panes open in Source Editor. The upper pane shows the list of template folders. The lower pane is blank.

3. In Template Editor, expand the folders so that the desired folder or template is showing.

**Caution**

Deleting a folder also deletes all templates under the folder.

4. Right-click that template or folder and choose **Delete**.

# Using Structural Verilog (.vm) in the Design Flow

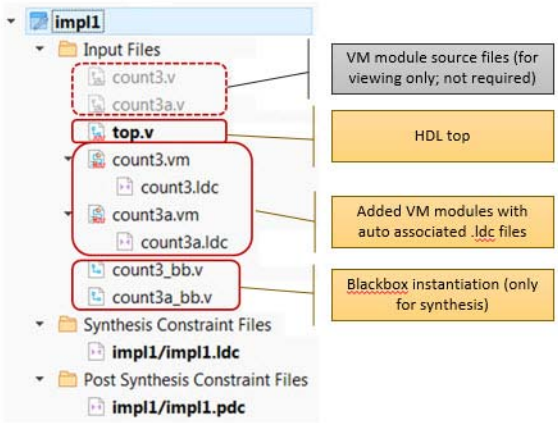
The Radiant software supports structural Verilog files (.vm) as an input module source for both LSE and Synplify Pro. Structural Verilog design entry is similar to Electronic Design Interchange Format (EDIF) design entry. Benefits of Structural Verilog over EDIF include:

- Allows individual module-level constraints.
- Allows simulation.

The following are Structural Verilog flow requirements:

- The top module must be HDL.
- The .vm and its associated .ldc files must have the same name.
- I/Os cannot be inserted insertion during .vm module synthesis.
- A Verilog Black Box instantiation module (_bb.v) is required for each .vm source file.

**Figure 3:**




**How VM Flow Works** The following provides an example of creating and using Structural Verilog (.vm) in the design flow.

1. Create a Radiant software project.
2. Add HDL sources.
3. Add timing constraints and save them to an .ldc file.
4. Synthesize the project without I/O insertion to create the .vm file.
5. Rename .vm and .ldc file to the module name. Repeat steps 2. through 5. until all .vm modules and .ldc files are created.
6. Add .vm files to the main project. The .ldc files are auto-associated.
7. Prepare a Verilog Black Box instantiation module for each .vm file.
8. Run the processes as normal after instantiating them from a top level. If processed properly, the Post-Synthesis Timing Constraint Editor will show all constraints for the .vm files.

# Coding Tips for LSE

The following are some recommendations if you choose to use LSE to synthesize your design. Tips include how to write code to infer different types of I/O ports, avoiding conflicts between VHDL packages, using VHDL coding, and preparing Verilog blocking assignments.

## About VHDL Coding

LSE typically applies the VHDL specification strictly, sometimes more strictly than other synthesis tools. The following are some coding practices that can cause problems with LSE:

**ieee.std_logic_signed or unsigned** When preparing VHDL code for LSE, you can include either:

USE ieee.std_logic_signed.ALL;

or:

USE ieee.std_logic_unsigned.ALL;

DO NOT include both statements. Code with both signed and unsigned packages may fail to synthesize because operators would have multiple definitions. Some synthesis tools may allow this but LSE does not.

**Strict Variable Typing** LSE is more strict about variable type requirements than most synthesis tools. A std_logic_vector signal cannot be assigned to a std_logic signal and an unsigned type cannot be assigned to a std_logic_vector signal. For example:

```
din : in  unsigned (data_width - 1 downto 0);
dout : out std_logic_vector (data_width - 1 downto 0));
...
dout <= din; -- Illegal, mismatched assignment.
```

Mismatched assignments like this will generate errors that stop synthesis.

## About Inferring Memory

To produce RAM and ROM in a design, you can write code for the design so that the synthesis tool infers the memory.

Inferring memory means that LSE, based on aspects of the code, implements a block of memory using programmable function units (PFU) or embedded block RAM (EBR) instead of registers. PFU is used for small memories and EBR for large. LSE can infer synchronous RAM, that is:

- Single-port, pseudo dual-port, or true dual-port.
- With or without asynchronous reset of the output.
- With or without write enables.
- With or without clock enables.

LSE can also infer synchronous ROM.

One of the advantages of inferring memory is that the design is portable to almost any FPGA architecture.

The following sections describe how to write code to infer different kinds of memory with LSE.

**See Also**

- “Inferring RAM” on page 69
- “Inferring ROM” on page 79

## Inferring RAM

The basic inferred RAM is synchronous. It can have synchronous or asynchronous reads and can be either single- or dual-port. You can also set initial values. Other features, such as resets and clock enables, can be added when needed. The following lists the rules for coding inferred RAM. Figure 4 on page 70 (Verilog) and Figure 5 on page 71 (VHDL) show the code for a simple, single-port RAM with asynchronous read.

To code RAM to be inferred, do the following:

- Define the RAM as an indexed array of registers.
- To control how the RAM is implemented (with distributed or block RAM), consider adding the syn_ramstyle attribute. See “syn_ramstyle” on page 460.
- Control the RAM with a clock edge and a write enable signal.
- For synchronous reads, see “Inferring RAM with Synchronous Read” on page 72.
- For single-port RAM, use the same address bus for reading and writing.

- For dual-port RAM, pseudo and true, see “Inferring Dual-Port RAM” on page 74.
- If desired, assign initial values to the RAM as described in “Initializing Inferred RAM” on page 78.

**Figure 4: Simple, Single-Port RAM in Verilog**

```
module ram (din, addr, write_en, clk, dout);
  parameter addr_width = 8;
  parameter data_width = 8;
  input [addr_width-1:0] addr;
  input [data_width-1:0] din;
  input write_en, clk;
  reg [data_width-1:0] mem [(1<<addr_width)-1:0];
    // Define RAM as an indexed memory array.

  always @(posedge clk) // Control with a clock edge.
  begin
    if (write_en) // And control with a write enable.
      mem[(addr)] <= din;
  end
  assign dout = mem[addr];
endmodule
```

### See Also

- “About Verilog Blocking Assignments” on page 80

**Figure 5: Simple, Single-Port RAM in VHDL**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ram is
generic (
  addr_width : natural := 8;
  data_width : natural := 8);
port (
  addr : in  std_logic_vector (addr_width - 1 downto 0);
  write_en : in  std_logic;
  clk : in  std_logic;
  din : in  std_logic_vector (data_width - 1 downto 0);
  dout : out std_logic_vector (data_width - 1 downto 0));
end ram;

architecture rtl of ram is
  type mem_type is array ((2** addr_width) - 1 downto 0) of
    std_logic_vector(data_width - 1 downto 0);
  signal mem : mem_type;
    -- Define RAM as an indexed memory array.
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then --Control with clock edge
      if (write_en = '1') then -- Control with a write enable.
        mem(conv_integer(addr)) <= din;
      end if;
    end if;
  end process;
  dout <= mem(conv_integer(addr));
end rtl;
```

## Inferring RAM with Synchronous Read

For synchronous reads, add a register for the read address or for the data output. Load the register inside the procedure or process that is controlled by the clock. The following examples show the simple RAM for Verilog and VHDL modified for synchronous reads. Changes are in bold text.

### Verilog Examples

**Figure 6: RAM with Registered Output in Verilog**

```
module ram (din, addr, write_en, clk, dout);
  parameter addr_width = 8;
  parameter data_width = 8;
  input [addr_width-1:0] addr;
  input [data_width-1:0] din;
  input write_en, clk;
  output [data_width-1:0] dout;
  reg [data_width-1:0] dout; // Register for output.
  reg [data_width-1:0] mem [(1<<addr_width)-1:0];

  always @(posedge clk)
  begin
    if (write_en)
      mem[(addr)] <= din;
    dout = mem[addr]; // Output register controlled by clock.
  end
endmodule
```

**Figure 7: RAM with Registered Read Address in Verilog**

```
module ram (din, addr, write_en, clk, dout);
  parameter addr_width = 8;
  parameter data_width = 8;
  input [addr_width-1:0] addr;
  input [data_width-1:0] din;
  input write_en, clk;
  output [data_width-1:0] dout;
  reg [data_width-1:0] raddr; // Register for read address.
  reg [data_width-1:0] mem [(1<<addr_width)-1:0];

  always @(posedge clk)
  begin
    if (write_en)
    begin
      mem[(addr)] <= din;
    end
    raddr <= addr; // Read addr. register controlled by clock.
  end
  assign dout = mem[raddr];
endmodule
```

### VHDL Examples

**Figure 8: RAM with Registered Output in VHDL**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ram is
generic (
  addr_width : natural := 8;
  data_width : natural := 8);
port (
  addr : in  std_logic_vector (addr_width - 1 downto 0);
  write_en : in  std_logic;
  clk : in  std_logic;
  din : in  std_logic_vector (data_width - 1 downto 0);
  dout : out std_logic_vector (data_width - 1 downto 0));
end ram;

architecture rtl of ram is
  type mem_type is array ((2** addr_width) - 1 downto 0) of
    std_logic_vector(data_width - 1 downto 0);
  signal mem : mem_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (write_en = '1') then
        mem(conv_integer(addr)) <= din;
      end if;
    end if;
    dout <= mem(conv_integer(addr));
      -- Output register controlled by clock.
  end process;
end rtl;
```

**Figure 9: RAM with Registered Read Address in VHDL**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ram is
generic (
  addr_width : natural := 8;
  data_width : natural := 8);
port (
  addr : in  std_logic_vector (addr_width - 1 downto 0);
  write_en : in  std_logic;
  clk : in  std_logic;
  din : in  std_logic_vector (data_width - 1 downto 0);
  dout : out std_logic_vector (data_width - 1 downto 0));
end ram;

architecture rtl of ram is
  type mem_type is array ((2** addr_width) - 1 downto 0) of
    std_logic_vector(data_width - 1 downto 0);
  signal mem : mem_type;
begin
  process (clk)
  begin
    if (clk'event and clk = '1') then
      if (write_en = '1') then
        mem(conv_integer(addr)) <= din;
      end if;
      raddr <= addr;
        -- Read address register controlled by clock.
    end if;
  end process;
  dout <= mem(conv_integer(raddr));
end rtl;
```

**See Also**

- "Inferring RAM" on page 69

## Inferring Dual-Port RAM

The following applies for dual-port, pseudo or true RAM.

- When using two address buses:
  - If the design does not simultaneously read and write the same address, add the syn_ramstyle attribute with the no_rw_check value to minimize overhead logic.
  - If writing in Verilog, use non-blocking assignments as described in "About Verilog Blocking Assignments" on page 80.

The examples below are based on the simple RAM of Figure 4 on page 70 (for Verilog) and Figure 5 on page 71 (for VHDL).

### Verilog Examples

**Figure 10: Pseudo Dual-Port RAM in Verilog**

```
module ram (din, write_en, waddr, wclk, raddr, rclk, dout);
  parameter addr_width = 8;
  parameter data_width = 8;
  input [addr_width-1:0] waddr, raddr;
  input [data_width-1:0] din;
  input write_en, wclk, rclk;
  output [data_width-1:0] dout;
  reg [data_width-1:0] dout;
  reg [data_width-1:0] mem [(1<<addr_width)-1:0]
    /* synthesis syn_ramstyle = "no_rw_check" */ ;

  always @(posedge wclk) // Write memory.
  begin
    if (write_en)
      mem[waddr] <= din; // Using write address bus.
  end
  always @(posedge rclk) // Read memory.
  begin
    dout <= mem[raddr]; // Using read address bus.
  end
endmodule
```

**Figure 11: True Dual-Port RAM in Verilog**

```
module ram (dina, write_ena, addra, clka, douta,
  dinb, write_enb, addrb, clkb, doutb);
  parameter addr_width = 8;
  parameter data_width = 8;
  input [addr_width-1:0] addra, addrb;
  input [data_width-1:0] dina, dinb;
  input write_ena, clka, write_enb, clkb;
  output [data_width-1:0] douta, doutb;
  reg [data_width-1:0] douta, doutb;
  reg [data_width-1:0] mem [(1<<addr_width)-1:0]
    /* synthesis syn_ramstyle = "no_rw_check" */ ;

  always @(posedge clka) // Using port a.
  begin
    if (write_ena)
      mem[addra] <= dina; // Using address bus a.
    douta <= mem[addra];
  end
  always @(posedge clkb) // Using port b.
  begin
    if (write_enb)
      mem[addrb] <= dinb; // Using address bus b.
    doutb <= mem[addrb];
  end
endmodule
```

### VHDL Examples

**Figure 12: Pseudo Dual-Port RAM in VHDL**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ram is
generic (
  addr_width : natural := 8;
  data_width : natural := 8);
port (
  write_en : in  std_logic;
  waddr : in  std_logic_vector (addr_width - 1 downto 0);
  wclk : in  std_logic;
  raddr : in  std_logic_vector (addr_width - 1 downto 0);
  rclk : in  std_logic;
  din : in  std_logic_vector (data_width - 1 downto 0);
  dout : out std_logic_vector (data_width - 1 downto 0));
end ram;

architecture rtl of ram is
  type mem_type is array ((2** addr_width) - 1 downto 0) of
    std_logic_vector(data_width - 1 downto 0);
  signal mem : mem_type;
  attribute syn_ramstyle: string;
  attribute syn_ramstyle of mem: signal is "no_rw_check";
begin
  process (wclk) -- Write memory.
  begin
    if (wclk'event and wclk = '1') then
      if (write_en = '1') then
        mem(conv_integer(waddr)) <= din;
          -- Using write address bus.
      end if;
    end if;
  end process;
  process (rclk) -- Read memory.
  begin
    if (rclk'event and rclk = '1') then
      dout <= mem(conv_integer(raddr));
        -- Using read address bus.
    end if;
  end process;
end rtl;
```

**Figure 13: True Dual-Port RAM in VHDL**

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity ram is
generic (
  addr_width : natural := 8;
  data_width : natural := 8);
port (
  addra : in  std_logic_vector (addr_width - 1 downto 0);
  write_ena : in  std_logic;
  clka : in  std_logic;
  dina : in  std_logic_vector (data_width - 1 downto 0);
  douta : out std_logic_vector (data_width - 1 downto 0);
  addrb : in  std_logic_vector (addr_width - 1 downto 0);
  write_enb : in  std_logic;
  clkb : in  std_logic;
  dinb : in  std_logic_vector (data_width - 1 downto 0);
  doutb : out std_logic_vector (data_width - 1 downto 0));
end ram;

architecture rtl of ram is
  type mem_type is array ((2** addr_width) - 1 downto 0) of
    std_logic_vector(data_width - 1 downto 0);
  signal mem : mem_type;
  attribute syn_ramstyle: string;
  attribute syn_ramstyle of mem: signal is "no_rw_check";
begin
  process (clka) -- Using port a.
  begin
    if (clka'event and clka = '1') then
      if (write_ena = '1') then
        mem(conv_integer(addra)) <= dina;
          -- Using address bus a.
      end if;
      douta <= mem(conv_integer(addra));
    end if;
  end process;
  process (clkb) -- Using port b.
  begin
    if (clkb'event and clkb = '1') then
      if (write_enb = '1') then
        mem(conv_integer(addrb)) <= dinb;
          -- Using address bus b.
      end if;
      doutb <= mem(conv_integer(addrb));
    end if;
  end process;
end rtl;
```

## See Also

- “Inferring RAM” on page 69

## Initializing Inferred RAM

Create initial values for inferred RAM to initialize memory.

**Verilog** Initialize RAM with the standard $readmemb or $readmemh tasks in an initial block. Create a separate file with the initial values in either binary or hexadecimal form. For example, to initialize a RAM block named “ram”:

```
reg [7:0] ram [0:255];
initial
begin
  $readmemh ("ram.ini", ram);
end
```

The data file has one word of data on each line. The data needs to be in the same order in which the array was defined. That is, for “ram [0:255]” the data starts with address 0; for “ram [255:0]” the data starts with address 255. The ram.ini file might start like this:

```
0A /* Address 0 */
23
5C
...
```

**VHDL** Initialize RAM with either signal declarations or variable declarations. Define an entity with the same ports and architecture as the memory. Use this entity in either a signal or variable statement with the initial values as shown below.

To initialize a RAM block named “ram,” define an entity such as:

```
entity ram_init is
port (
  clk : in std_logic;
  addr : in std_logic_vector(7 downto 0);
  din : in std_logic_vector(7 downto 0);
  we : in std_logic;
  dout : out std_logic_vector(7 downto 0));
end;
architecture arch of ram_init is
  type ram_init_arch is array(0 to 255)
  of std_logic_vector (7 downto 0);
```

Then use the entity in a signal statement:

```
signal ram : ram_init_arch := (
"00001010",
"00100011",
"01011100",
...
others => (others => '0'));
```

Or use the entity in a variable statement:

```
variable ram : ram_init_arch := (
1 => "00001010",
```

```
...
others => (1=>'1', others => '0'));
```

**See Also**

- "Inferring RAM" on page 69

## Inferring ROM

To code the ROM to be inferred, do the following:

- Define the ROM with a case statement or equivalent IF statements.
- Assign constant values, all with the same width.
- Assign values for at least 16 addresses or half of the address space, whichever is greater. For example, if the address has 6 bits, the address space is 64 words, and at least 32 of them must be assigned values.
- To control how the ROM is implemented (with distributed or block ROM), consider adding the syn_romstyle attribute. See "syn_romstyle" on page 464.

**Figure 14: ROM Inferred with Case Statement in Verilog**

```
module rom(data, addr);
  output [3:0] data;
  input [4:0] addr;
  always @(addr) begin
    case (addr)
       0 : data = 'h4;
       1 : data = 'h9;
       2 : data = 'h1;
       ...
      15 : data = 'h8;
      16 : data = 'h1;
      17 : data = 'h0;
      default : data = 'h0;
    endcase
  end
endmodule
```

**Figure 15: ROM Inferred with If Statement in VHDL**

```
entity rom is
port (addr : in std_logic_vector(4 downto 0);
  data : out std_logic_vector(3 downto 0) );
end rom;

architecture behave of rom is
begin
  process(addr)
  begin
    if    addr =  0 then data <= "0100";
    elsif addr =  1 then data <= "1001";
    elsif addr =  2 then data <= "0001";
    ...
    elsif addr = 15 then data <= "1000";
    elsif addr = 16 then data <= "0001";
    elsif addr = 17 then data <= "0000";
    else                 data <= "0000";
    end if;
  end process;
end behave;
```

## About Verilog Blocking Assignments

LSE support for Verilog blocking assignments to inferred RAM and ROM, such as “ram[(addr)] = data;,” is limited to a single assignment. Multiple blocking assignments, such as you might use for true dual-port RAM (see Figure 16 on page 80), or a mix of blocking and non-blocking assignments are not supported. Use non-blocking assignments instead (<=). See Figure 17 on page 81.

**Figure 16: Example of RAM with Multiple Blocking Assignments (Wrong)**

```
always @(posedge clka)
begin
  if (write_ena)
    ram[addra] = dina; // Blocking assignment A
  douta = ram[addra];
end
always @(posedge clkb)
begin
  if (write_enb)
    ram[addrb] = dinb; // Blocking assignment B
  doutb = ram[addrb];
end
```

**Figure 17: Example Rewritten with Non-blocking Assignments (Right)**

```
always @(posedge clka)
begin
  if (write_ena)
    ram[addra] <= dina;
  douta <= ram[addra];
end
always @(posedge clkb)
begin
  if (write_enb)
    ram[addrb] <= dinb;
  doutb <= ram[addrb];
end
```

## About Verilog Generate Blocks

If a module has multiple generate blocks, LSE requires that each block have a different name. Assign names with begin statements. See Figure 18 on page 81.

**Figure 18: Generate Blocks with Assigned Names**

```
generate
begin: R1_gen // Name first generate block.
  genvar i1;
  for (i1=0; i1<=15; i1=i1+1)
    regset R1 (.clk(clk[i1]),.q(q[i1]),.d(d[i1]));
end
endgenerate

generate
begin: IR0_gen // Name second generate block.
  genvar i;
  for (i=0; i<=7; i=i+1)
    ioreg IR0 (.clk(clk[0]),.reset(rst[i]),.q(q[i]),.d(d[i]));
end
endgenerate
```

## Inferring I/O

To specify a type of I/O port, follow these models.

### Verilog

Open Drain:

```
output <port>;
wire <output_enable>;
assign <port> = <output_enable> ? 1'b0 : 1'bz;
```

Bidirectional:

```
inout <port>;
wire <output_enable>;
```

```
wire <output_driver>;
wire <input_signal>;
assign <port> = <output_enable> ? <output_driver> : 1'bz;
assign <input_signal> = <port>;
```

**VHDL**

Tristate:

```
library ieee;
use ieee.std_logic_1164.all;
entity <tbuf> is
port (
  <enable> : std_logic;
  <input_sig> : in std_logic_vector (1 downto 0);
  <output_sig> : out std_logic_vector (1 downto 0));
end tbuf2;
architecture <port> of <tbuf> is
begin
  <output_sig> <= <input_sig> when <enable> = '1' else "ZZ";
end;
```

Open Drain:

```
library ieee;
use ieee.std_logic_1164.all;
entity <od> is
port (
  <enable> : std_logic;
  <output_sig> : out std_logic_vector (1 downto 0));
end od2;
architecture <port> of <od> is
begin
  <output_sig> <= "00" when <enable> = '1' else "ZZ";
end;
```

Bidirectional:

```
library ieee;
use ieee.std_logic_1164.all;
entity <bidir> is
port (
  <direction> : std_logic;
  <input_sig> : in std_logic_vector (1 downto 0);
  <output_sig> : out std_logic_vector (1 downto 0);
  <bidir_sig> : inout std_logic_vector (1 downto 0));
end bidir2;
architecture <port> of <bidir> is
begin
  <bidir_sig> <= <input_sig> when <direction> = '0' else "ZZ";
  <output_sig> <= <bidir_sig>;
end;
```

# Customizing Source Editor

The following can be performed in Source Editor:

- Select a default file language
- Choose window modes (attached or detached).
- Set encoding.
- Configure line number showing.
- Configure text wrapping.
- Set printing options.

## Setting File and Window Modes

Source Editor offers you the flexibility to configure file and window modes, such as line number showing, text wrapping, and printing options. The configuration can be applied to the current file, a specified type of file, or any new files created in Source Editor.

**To select a default file language:**

1. In Source Editor, choose **View > Language**. The following options are available: C++, Hypertext, Perl, Python, TCL, Verilog, VHDL, EDIF, SystemVerilog, and Preference.
2. Choose the desired language from the list. The language selected becomes the default language in Source Editor.

**To attach or detach Source Editor from the Radiant software:**

1. To detach Source Editor from the Radiant software main window, click the Detach Tool icon in the upper-right corner of Source Editor.
2. To attach Source Editor to the Radiant software main window, choose **Window > Attach Window**, or click on the Attach Tool icon in the upper-right corner of Source Editor.

## Customizing Source Editor from the Radiant software Main Window

**To customize Source Editor settings in the Radiant software main window:**

1. In the Radiant software main window, choose **Tools > Options**.
2. In the Tools window, you can set settings in the each related section.

   **General**

   Show Line Number: Displays line numbers of the files (on the left side of the Editor).

   Show Indicator Margin: Displays the side bar allowing you to select the whole row.

   Show Folder Margin: Displays the side bar (on the left side of the Editor) that is used to fold/unfold the syntax.

Tab Size: Specifies how many spaces are entered when a tab character is pressed.

Restore Returns to the default setting.

**Font**

Lists the current font used in Source Editor. To change the current font, click the drop-down menu next to Font Family and select the desired font. To change the font size, click the drop-down menu next to Font Size and select the desired size. Click **OK** to exit. The selected font name, font type, and font size will be displayed in Source Editor.

Restore Returns all selections to their default settings.

3. In the Color window, you can set the default colors for the text, comments, numbers, keywords, and a variety of other options under the Source Editor tab.

   **Colors**

   Click in the Themes drop-down menu to select either a light or dark overall color theme for Source Editor.

   To select a different color, double-click the color box next to the desired element to open the Select Color window. Click on the new color to use.

   Restore Returns all selections to their default settings.

If you want to change to use your own text editor other than Source Editor, you can customize to add your own text editor and associated files to your own text editor program in the Options dialog. Select **Tools > Options** from the Radiant software main window, click on **General**, and then select the **File Associations** tab.

## References

This section includes the following references.

### Verilog HDL Language Reference

If you have installed Synplify Pro for Lattice, refer to the *Synplify and Synplify Pro for Lattice Reference Manual* in the Synplify Pro installation directory.

### VHDL Language Reference

If you have installed Synplify Pro for Lattice, refer to the *Synplify and Synplify Pro for Lattice Reference Manual* in the Synplify Pro installation directory.

# Designing with Modules and PMI

Modules are functional bits of design that can be re-used wherever that function is needed. Creating such modules with hardware design languages is common practice. To help your design along, the Radiant software provides

a variety of modules for common functions. They are optimized for Radiant software device architectures and can be customized. Use these modules to speed up your design work and to get the most effective results.

Radiant software modules come in a variety of forms:

- IP Catalog provides a variety of functions ranging from the most basic, such as arithmetic and memory, to much more complex functions. With IP Catalog these modules can be extensively customized and created as part of a specific project or as a library for multiple projects. See “Creating IP Catalog Modules and IP” on page 86.

  However, many of these modules can also be used with PMI (see next item). To decide which method to use, see “PMI or IP Catalog?” on page 85.
- PMI (Parameterized Module Instantiation) is an alternate way to use some of the modules that come with IP Catalog. Instead of using IP Catalog, PMI can directly instantiate a module into your HDL and customize it by setting parameters in the HDL. You may find this easier than using IP Catalog if your design requires many variations of the same module. To decide which method to use, see “PMI or IP Catalog?” on page 85.
- Reference designs provide you with a starting point on creating your own modules. Lattice Reference Designs are available in Verilog and VHDL, and can be downloaded from the Lattice website: www.latticesemi.com/ip.
- Lattice library primitives are very basic functions, such as logic gates and flip-flops. They can be directly instantiated as HDL into designs. But this is an advanced technique and should usually be avoided. For more information, see “Designing with Radiant software Library Primitives” on page 94.

# PMI or IP Catalog?

Many IP Catalog modules are also available as PMI modules, but PMI doesn’t offer error checking. This and the next two sections discuss the pros and cons of both options to help you decide which is best for your project.

PMI is a convenient way to use modules of the same type but that vary from instance to instance. This eliminates the need to create a separate module for each instance using IP Catalog.

For example, a design might require dozens of FIFOs that are functionally the same but require different address depths. With PMI, you could insert the same FIFO instantiation command wherever it’s needed in the HDL and just change the address depth parameter as you go. The alternative would be to use IP Catalog to generate different modules for each variation and then insert the instantiation template for each of them. In this situation, you might find the PMI method easier and faster.

Before deciding whether to use a PMI module, compare it to the equivalent IP Catalog module. Often IP Catalog provides more options than PMI does. IP Catalog also assures that all of your option selections and parameter settings are legal. PMI offers no error checking.

**See Also**

- For a list of PMI modules, see Table 38 on page 484.
- “Creating IP Catalog Modules and IP” on page 86
- “Using PMI” on page 91

# Creating IP Catalog Modules and IP

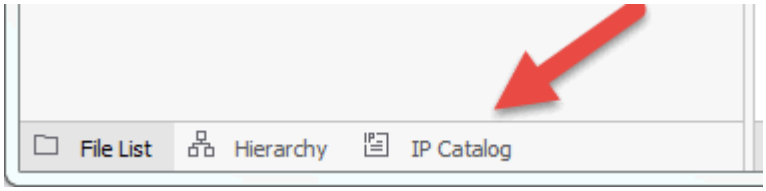
IP Catalog is an easy way to use a collection of functional blocks from the Radiant software. There are two types of functional blocks available through IP Catalog: modules and IP. IP Catalog enables you to extensively customize these blocks. They can be created as part of a specific project or as a library for multiple projects.

Modules: These basic, configurable blocks come with IP Catalog. They provide a variety of functions including I/O, arithmetic, memory, and more. Open IP Catalog to see the full list of what’s available. Also see the Lattice Module Reference Guide.

IP: Intellectual property (IP) are more complex, configurable blocks. They are accessible through IP Catalog, but they do not come with the tool. They must first be downloaded and installed as a separate step before they can be accessed from IP Catalog. To see all that’s available and to learn about licensing and other vendors of IP, go to the Lattice website: www.latticesemi.com/ip.

**Overview of the IP Catalog Process** Below are the basic steps of using IP Catalog modules and IP. For details of performing these steps, see the following topics.

1. Open IP Catalog. IP Catalog is accessed via a tab at the lower left of the Radiant software. Click the tab to view the list of available modules and IP.




2. Customize the module/IP. These modules and IP can be extensively customized for your design. The options range from setting the width of a data bus to selecting features in a communications protocol. At a minimum you need to specify the design language to use for the output.
3. Generate the module/IP and bring its .ipx file into your project. Prior to generating the module/IP, select the option “Insert to project.” This will automatically bring the .ipx file into your project after the generation step completes. If you do not select this option, add the .ipx file to your project as you would with any other source file (such as a Verilog or VHDL file) after the generation is complete.
4. Instantiate the module/IP into the project’s design. An HDL instance template is generated during the generation step to simplify this step.

5. IP Catalog modules and IP can be further modified or updated later. After the .ipx file has been added to the Radiant software project, it is visible in the project's file list. Double-clicking the .ipx file brings up the module/IP's configuration dialog box where changes can be made and the generation process repeated.

**See Also**

- "Generating a Module or IP with IP Catalog" on page 87
- "IP Catalog Output Files for Modules" on page 88
- "Importing an IP Catalog Module or IP into a Project" on page 88
- "Instantiating an IP Catalog Module or IP" on page 89
- "Downloading and Installing Soft IP" on page 89

## Generating a Module or IP with IP Catalog

Use IP Catalog to generate a customized functional block from any module or installed IP.

**To generate an IP or module:**

1. In the Module/IP tree, select the module or IP that you want to generate.
2. To get more information about the module or IP, click the **IP Information** tab.
3. Double-click the module or IP, or click the Generate Module/IP Instance  button, located at the upper-left of the tool the In the Configuration tab, specify general project information and the base file name for the module or IP.
   - Instance Name is the base name for the module's files (that is, with no extension).
   - Create In is the location for the customized module's files.
4. Click **Next.**

   The module's dialog box opens.
5. In the dialog box, select your desired options.Click **Generate**.
6. To automatically import the .ipx file into your project when the module/IP is generated, select **Insert to project** in the Check Generating Result dialog box.
7. Click **Finish**.

**See Also**

- "IP Catalog Output Files for Modules" on page 88

## IP Catalog Output Files for Modules

IP Catalog creates the following output files for modules under the specified Project Path. The <*file_name*> comes from the File Name specified in the Configuration tab.

IP Catalog creates some different files for IP. These are documented in the IP's associated user guide.

**Table 5:**

| File Name | Description |
|---|---|
| <*file_name*>.ipx | Manifest file. This file is loaded into IP Catalog so that modifications can be made to the module. |
| <*file_name*>.tmpl.v | Instantiation template for Verilog netlist. |
| <*file_name*>.v | Verilog HDL file for both synthesis and simulation. Verilog output files declare implicit wire types. |

## Importing an IP Catalog Module or IP into a Project

After generating module source files using IP Catalog, you can import the module by importing the IP Catalog manifest file (.ipx). Modules and IP have several files of different types, but you only need to import the .ipx. The .ipx file identifies the components needed to make up the module or IP.

Importing the module may not be necessary if the "Import IPX to  project" option was selected when the module was generated.

**To import a module:**

1. In the File List view, right-click the implementation folder ( ) and choose **Add > Existing File**.
2. Browse for the customized module's .ipx file, <*file_name*>**.ipx**, and select it.
3. Click **Add**.

   The .ipx file is added to the File List view.
4. Check the Output Panel for error messages. If the module is not targeted for the current device, try double-clicking the file to regenerate the module.

After importing the module, you can further customize it for your design project including changing the device type, design language, and any of the options specific to the module. You can also update older modules or IP to the latest version. See "Importing an IP Catalog Module or IP into a Project" on page 88.

## Instantiating an IP Catalog Module or IP

IP Catalog modules and IP are instantiated the same way other modules are in your HDL. When you generate modules and IP in IP Catalog, the tool also produces a Verilog or VHDL file with the necessary instantiation commands. You can copy and paste the contents of this file into any of your source files.

Before instantiating any IP Catalog module, check that it is compatible with your design project's device. When they were created, the modules were optimized for a specific device and may depend on that device's architecture. A module may not work with a different type of device and may need to be regenerated. If so, see "Importing an IP Catalog Module or IP into a Project" on page 88.

The instantiation file is located in the folder that the module was created in. The file name is based on the module's name: <*module name*>_tmpl.v or <*module name*>_tmpl.vhd.

Copy the instantiation command and paste it into one of your source files. Then add an instance name and signal names to the module ports.

Before running any processes on the design, make sure that the IP Catalog module or IP has been imported into the project. See "Importing an IP Catalog Module or IP into a Project" on page 88.

## Downloading and Installing Soft IP

Soft IP can be downloaded from the Lattice website. Use the **Install a User IP** feature of IP Catalog to install a Soft IP.

**To download and install a Soft IP:**

1. Using a web browser, download the Soft IP installer from the Lattice website.
2. In the Radiant IP Catalog, click on the **Install a User IP** button.
3. In the **Select user IP package file to install** dialog box, browse to the Radiant Software IP Package (.ipk) file, and click **Open**.

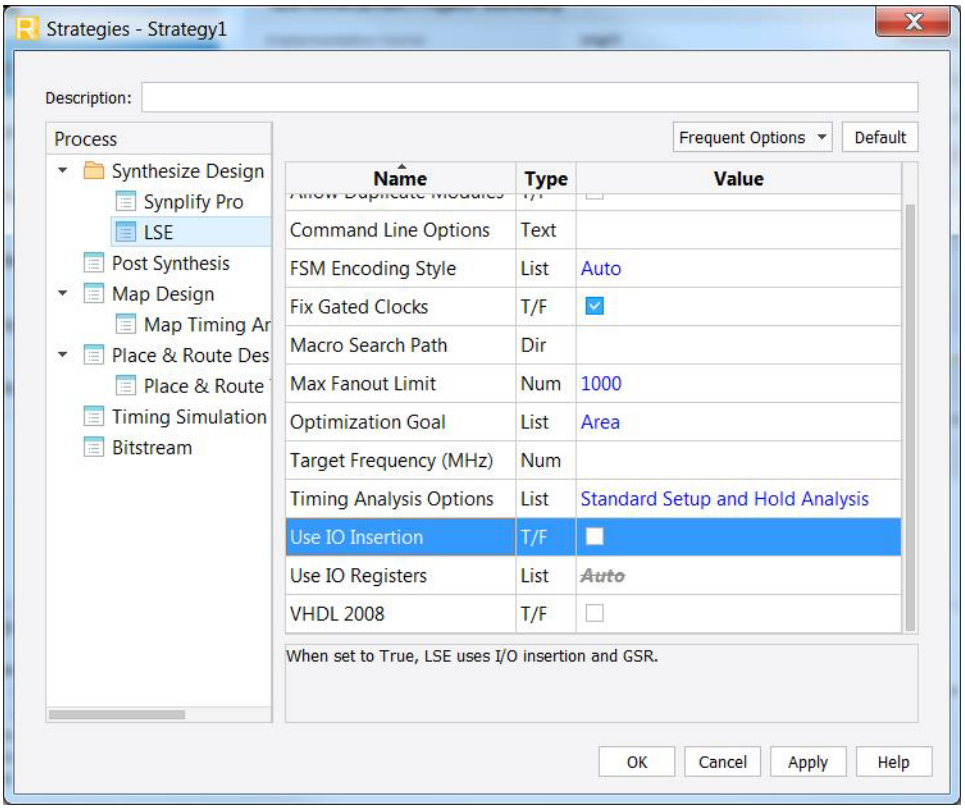
- The Soft IP will be installed into a folder in the user's personal directory. For example: C:/Users/<*username*>/RadiantIPLocal/<*IP_name*>.
- The Soft IP will be added into the IP Catalog.

## Creating an IP with a Pre-Generated I/O

This section assumes that the black box you wish to compile already exists. In this example, the black box is called bidirec.v. If bidirec.v contains your top level module in the Radiant software, you must make sure it "knows" that you don't intend to use this directly.

To do that, choose **Projects > Active Strategy > LSE Settings**, and disable **Use IO Insertion**.

**Figure 19:**




Make sure that the module defined in your bidirec.v file is set as your top level module in the project. That is all the setup required for a basic case.

Next, there may be a reason for you to want to force an I/O to be inserted into the module. In order to force I/O insertion on a per-pad basis, you can use the syn_insert_pad attribute. The following is an example of how this is used.

```
module bidirec (oe, clk, inp, outp, bidir);
inout   bidir /* synthesis syn_insert_pad = 1 */ ;
[rest of your code]
endmodule
```

Once this setup is complete and attributes have been added, the module is ready to be created.

Run Lattice Synthesis Engine (**Synthesize Design**) to generate a black box for your module. Depending on your project settings and names, the UDB file may need to be renamed to bidirec_bb.udb to ensure that it's consistent with the rest of this project.

Now that you have a black box, you'll need to create a Verilog wrapper file so users and LSE can "see" the module. In this case, bidirec_bb.v.

The following example contains the entirety of a wrapper for the bidirec defined in bidirec.v.

```
module bidirec (oe, clk, inp, outp, bidir) /* synthesis
syn_black_box black_box_pad_pin="bidir" */;
   input   oe;
   input   clk;
   input   inp;
```

## Using IP with Pre-Generated I/O

Before continuing, be sure you've completed the steps in "Creating an IP with a Pre-Generated I/O" on page 89.

Make sure that the bidirec_bb.v file has attributes for its pad pins set. If it does not, then Synthesis will assume there are no inserted I/Os. If you forget to do this for a pin that has a pad, you will encounter an error in the Radiant software when it determines that you are attempting to assign two pads to a single pin.

Refer to the example at the end of in "Creating an IP with a Pre-Generated I/O" on page 89.

Now create the top file. To do this, you must create a project that has a top.v file that uses your bidirec module defined in bidirec_bb.v and the corresponding black box. Because the intention is for this to be a top level module, you need to make certain that Synthesis knows to insert I/Os. To do that, choose **Projects > Active Strategy > LSE Settings**, and enable **Use IO Insertion**.

Next, make sure that Post-Synthesis knows where the black box file is located. To do this, go to your Post Synthesis settings and add the .udb file for the black box to your list of IPs. In this case, point to the bidirec_bb.udb file that was created in Creating an IP with a Pre-Generated I/O.

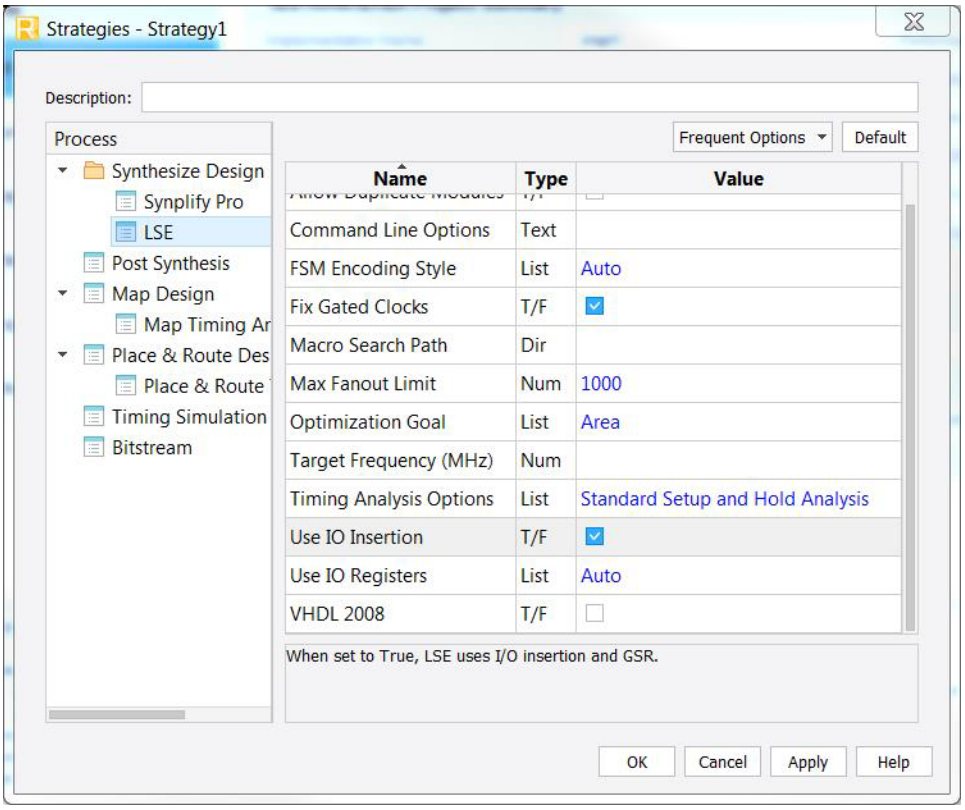
Before compiling your project, be sure to resolve any possible double pad situations as it's not possible to see inside the black box. To do this, consult with your bidirec_bb.v file. If you skipped Creating an IP with a Pre-Generated I/O, the following is a copy for reference.

```
module bidirec (oe, clk, inp, outp, bidir) /* synthesis
syn_black_box black_box_pad_pin="bidir" */;
   input   oe;
   input   clk;
   input   inp;
   output  outp;
   inout   bidir;
endmodule
```

# Using PMI

As mentioned earlier, PMI is an alternate way to use some of the modules that come with IP Catalog. Instead of using IP Catalog, you directly instantiate a module into your HDL and customize it by setting parameters in the HDL using PMI. You may find this easier than using IP Catalog if your design requires many variations of the same module.

**Figure 20:**




**See Also**

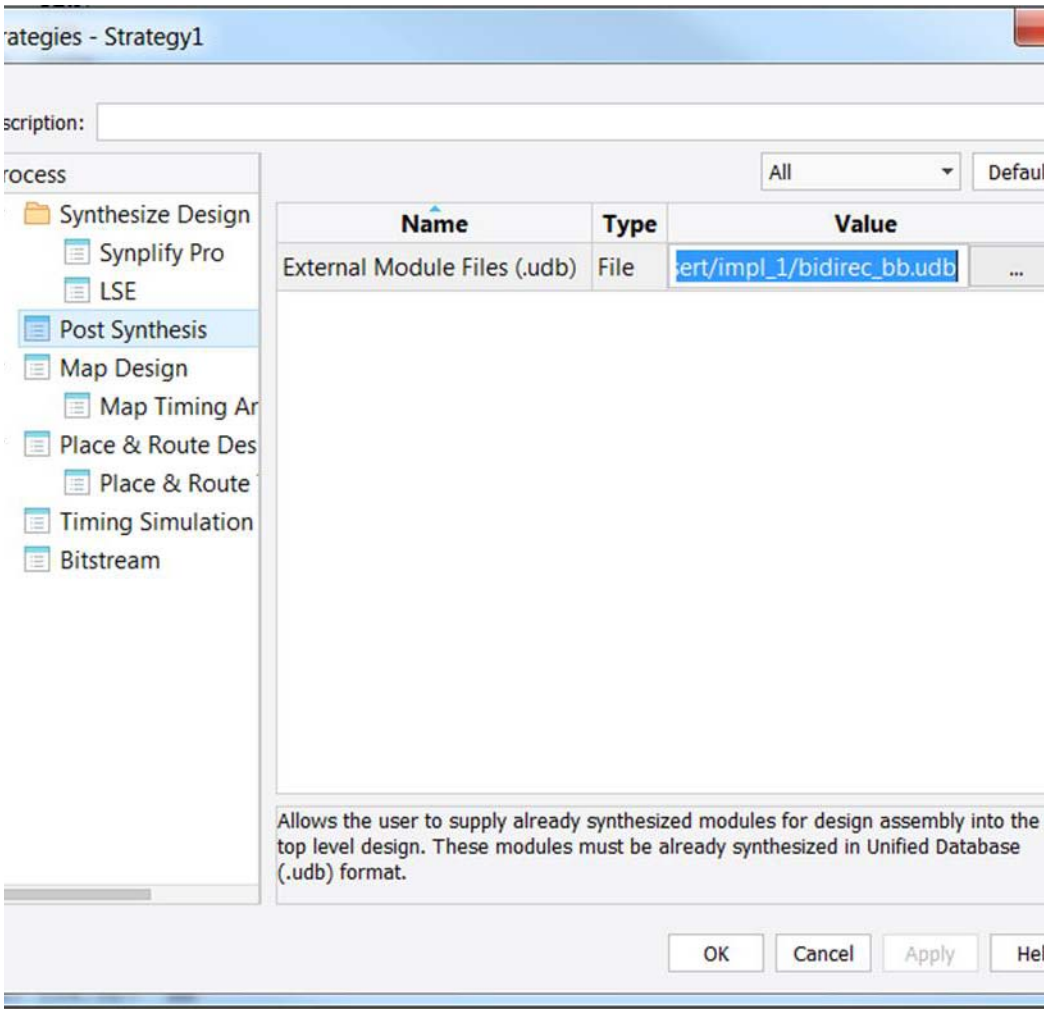
- "PMI or IP Catalog?" on page 85
- "PMI Module Lookup (In Alphabetical Order)" on page 484
- "PMI Process Steps" on page 92
- "Instantiating a PMI Module" on page 93

## PMI Process Steps

This section provides a brief listing of the process steps taken when using PMI.

1. Using Source Editor, insert PMI modules in the HDL source files. This is the recommended method because it includes the PMI instance templates, which require minimal editing.
2. If using stand-alone synthesis and simulation tools, add the PMI soft IP from the <install_path>/ip/pmi/ directory.
3. Perform the remaining implementation steps as you would normally.

**Figure 21:**




## Instantiating a PMI Module

PMI modules are instantiated the same way other modules are in your HDL. The Radiant software provides a template for the Verilog or VHDL instantiation command that specifies the customized module's ports and parameters. Customize the module by changing its parameters.

**To instantiate a PMI module:**

1. With Source Editor, open the source file that will receive the module. See "Running Source Editor" on page 61.
2. Place the cursor in the desired spot for the instantiation command.
3. Choose **View > Template Editor**.

   The Template Editor view appears.

4. In Template Editor, open the **Verilog > PMI Templates** or **VHDL > PMI Templates** folder and select the module.

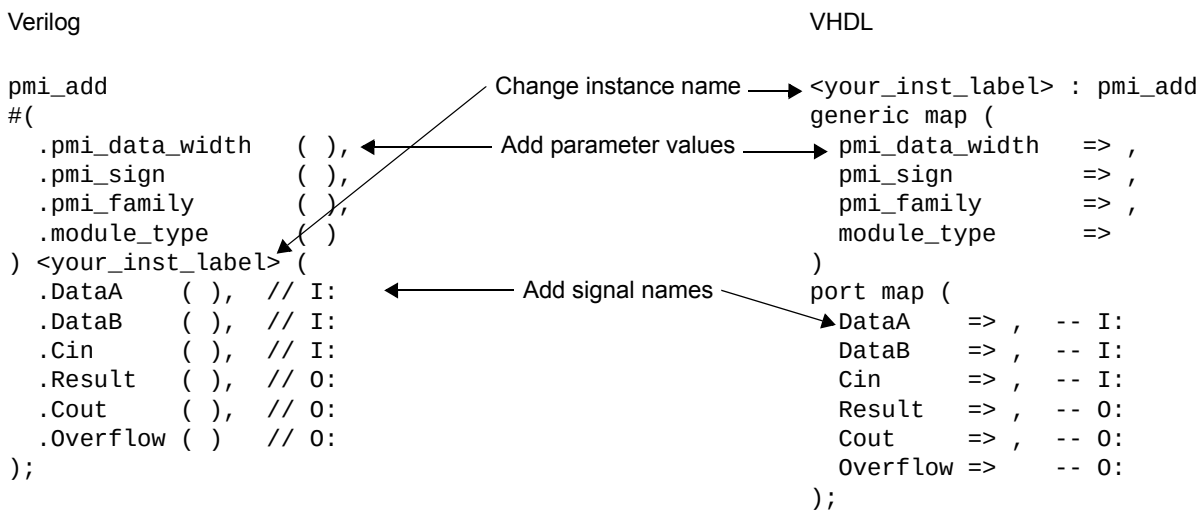
   The code of the template appears in the lower box of the Template Editor view.

5. Right-click the template and choose **Insert to text**.

   The instantiation command is copied into your source file.

6. Add an instance name, set parameter values, and add signal names to the corresponding module ports in the instantiation command, as shown in Figure 22. For information about allowed parameter values, see the section on the module in the "Lattice Module Reference Guide" on page 483.

**Figure 22: PMI Instantiation Command Example**




7. Save and close the source file.

# Designing with Radiant software Library Primitives

Any Radiant software library primitive described in the "Primitive Library - iCE40 UltraPlus" on page 490 can be instantiated as a Verilog module or VHDL component in your RTL design. This "gate-level" design approach can be error-prone and should be limited to a small number of primitives if attempted at all. In general, Lattice recommends you rely on IP Catalog to generate modules that are built with Radiant software library primitives.

To minimize the amount of code overhead required to design using a library primitive, the Radiant software provides a Verilog and VHDL synthesis header library file for each major FPGA device family. Refer to the "Lattice Synthesis Header Libraries" on page 163 topic for details. The module is generally treated as a "black box", which causes the synthesis tool to pass instances of the library primitive into the target netlist untouched.

The “Primitive Library - iCE40 UltraPlus” on page 490 contains descriptions, pin outs, and schematic diagrams of all library primitives for Radiant software FPGA libraries.

**See Also** ▶“Primitive Library - iCE40 UltraPlus” on page 490

- “Lattice Synthesis Header Libraries” on page 163
- “Simulation Library Files” on page 101

# Simulating the Design

This topic explains how to perform functional and timing simulation by using the Radiant software tools and third-party simulators.

**See Also**
- "Functional Simulation" on page 97
- "Simulation Environment" on page 97
- "Design Source Files" on page 97
- "Creating a New Simulation Project in the Radiant Software" on page 98
- "Exporting Gate-Level Simulation Files" on page 99
- "Verifying Designs with Timing Simulation" on page 100
- "Third-Party Simulators" on page 100

## Simulation in the Radiant Software

The Radiant software provides you with an interface to create a new simulation project file that you can import into a standalone simulator. The Radiant software supports Active-HDL and ModelSim/Questa simulation file for file exports. To create test benches for simulation in the Radiant software, generate all test benches in your third-party tool using an imported simulation project (.spf) file.

**See Also**
- "Functional Simulation" on page 97
- "Simulation Environment" on page 97
- "Design Source Files" on page 97
- "Creating a New Simulation Project in the Radiant Software" on page 98
- "Exporting Gate-Level Simulation Files" on page 99

# Functional Simulation

Functional simulation is the process of verifying that the model of the design matches the functional specification. Functional simulation helps identify logic errors in a design before it is programmed into a device.

To perform functional simulation on designs modeled with Verilog HDL in the Radiant software design flow, perform pre-synthesis functional simulation on your source HDL or netlist using a third party simulator. In the Radiant software, use the subprocesses **Gate-Level Simulation File** to produce a simulation file for functional simulation of primitive gate-level logic during that stage in the flow.

**HDL Simulators** The Radiant software and model libraries are qualified with a variety of popular HDL simulators, including Aldec Riviera Pro®, Aldec Active-HDL®, Cadence® NC-Verilog®, Cadence® NC-VHDL®, Cadence® NCSim®, Mentor Graphics® ModelSim/Questa®, Mentor Graphics Questa®,and Synopsys® VCS®.

**See Also** ▶"Simulation in the Radiant Software" on page 96

# Simulation Environment

The Radiant software environment provides library model resources in the Verilog HDL format to support stand-alone simulation with Aldec, Cadence, Mentor Graphics, and Synopsys simulators. Using the simulation wizard inside of the Radiant software, you can create simulation project files that can be exported and brought into standalone simulators.

In addition, you can use the Radiant software's back annotation tools to extract gate-level models and the timing data. This type of simulation is convenient if you want to simulate a design file outside the current project. Stand-alone operation enables you to choose your own settings and preferences.

You can customize the simulation by creating a different script file (.do) that contains commands equivalent to the ModelSim/Questa or Active-HDL graphical user interface (GUI) commands. For information about creating your own simulation scripts, see Aldec's Active-HDL online Help.

**See Also** ▶"Simulation in the Radiant Software" on page 96

# Design Source Files

The third-party simulators enable you to simulate the operation of your design in the following design entry formats:

- VHDL format (<*design*>.vhd) – Describes the circuit in Very High-Speed IC hardware description language (VHDL) format.
- Verilog HDL format (<*design*>.v) – Describes the circuit in Verilog format, which is an industry-standard hardware description language used to

describe the behavior of hardware that can be implemented directly by logic synthesis tools.

**See Also** ▶"Simulation in the Radiant Software" on page 96

# Creating a New Simulation Project in the Radiant Software

This topic describes how to use the Radiant Simulation Wizard to create a simulation project (.spf) file so you can import it into a standalone simulator.

1. In the Radiant software, click **Tools > Simulation Wizard** or click the  icon in the toolbar. The Simulation Wizard opens.
2. In the Preparing the Simulator Interface page click **Next**.
3. In the Simulator Project Name page, enter the name of your project in the Project Name text box and browse to the file path location where you want to save your simulation project.

   When you designate a project name in this wizard page, a corresponding folder will be created in the file path you choose. If you add a subdirectory to the default file path the wizard will prompt you to create a new directory. Click **Yes** in the popup dialog box that appears.
4. Click either the **Active-HDL** or **ModelSim/Questa** simulator check box and click **Next**. Note that if ModelSim/Questa is not installed, it will appear grayed out and not be selectable.
5. In the Process Stage page choose what type of Process Stage of simulation project you wish to create. Valid types are RTL and Post-Route Gate-Level+Timing. Only those process stages that are available are activated. Click **Next**.

   Note that you can make a new selection for the current Strategy if you have more than one defined in your project.

   The software supports multiple strategies per project implementation which allows you to experiment with alternative optimization options across a common set of source files. Since each strategy may have been processed to different stages, this dialog allows you to specify which stage you wish to load.
6. In the Add and Reorder Source page, select from the source files listed in the Source Files list box or use the browse button to choose a source file not listed.

   - By default, the **Automatically Set Simulation Compilation File Order** option is checked.
   - If you wish to keep the source files in the local simulation project directory you just created, check the **Copy Source to Simulation Directory** option.
7. Click **Next**. If the **Automatically Set Simulation Compilation File Order** option was checked in the previous step, you'll need to specify a top level

simulation test bench in the **Simulation Top Module** box in the Parse HTL and Test Bench page.

8. Click **Next**. A Summary page appears that provides information on the project selections, including the simulation libraries. By default the **Run simulator**, **Add top-level signals to waveform display**, and **Run simulation** check boxes are enabled. Their functionality is as follows:
   - The **Run simulator** option launches the simulation tool you chose earlier in the wizard.
   - The **Add top-level signals to waveform display** option can only be selected if the **Run simulator** box is checked. This option adds top-level signals to the waveform in the simulation tool.
   - The **Run simulation** option can only be selected if the **Add top-level signals to waveform display** option is checked. This option causes your simulation tool to run the simulation automatically.
9. Click **Finish**. After the simulation is complete, the Simulation Wizard Project (.spf) file and a simulation script DO file are generated and the simulation is displayed. You can import the DO file into your current project, if desired. If you are using Active-HDL, the wizard will generate an .ado file and if you are using ModelSim/Questa, it creates an .mdo file as well.

**See Also** ▶"Simulation in the Radiant Software" on page 96

## Exporting Gate-Level Simulation Files

In the Radiant software, you can export gate-level project simulation source files into your third-party vendor simulation tool for timing simulation.

**To export a Gate-Level simulation files in the Radiant software:**

1. From the Quick Run Toolbar, click Task Detail View and select the **Gate-Level Simulation File** subprocess under **Export Files**.
2. Double click **Export Files** to run the export feature. This will generate a back-annotated version of your design file in a Verilog netlist (.vo) simulation file and an SDF timing file.

**Note:**
You can also use Tcl Radiant commands in the Tcl Console to accomplish this task, as shown below:

```
prj_run Export -task TimingSimFileVlg
```

**See Also** ▶"Simulation in the Radiant Software" on page 96

# Timing Simulation

The Radiant software supports two types of timing verification:

- Dynamic timing simulation

- Static timing analysis

See the Static Timing Analysis topic and related information for more information on that type of design verification.

To facilitate timing simulation, the Radiant software allows you to export simulation files that you can import into a simulator for design verification using dynamic timing simulation. In the Process view under **Export Files** use **Gate-Level Simulation File** to export a simulation project file to a standalone Aldec Active-HDL® or Mentor Graphics® ModelSim/Questa® hardware design language (HDL) simulator. Some simulators allow a combination of both languages to be simulated concurrently. This scenario is common when mixing IP cores.

**See Also** "Verifying Designs with Timing Simulation" on page 100

## Verifying Designs with Timing Simulation

Timing simulation is based on an event-driven simulator and requires you to specify a test vector (waveform). Where timing analysis returns partial timing information, dynamic timing simulation (timing simulation) gives you detailed information about gate delays and worst-case circuit conditions. Because the total delay of a complete circuit depends on the number of gates the signal sees and on the way the gates have been placed in the device, you can only perform timing simulation after you have implemented the design. Timing simulation also requires several input files to run.

The Radiant software simulation files can be imported into HDL simulators (e.g., Aldec's Active-HDL or Mentor Graphics' ModelSim/Questa) for dynamic timing simulation. You can simulate FPGA devices by using any popular third-party HDL simulator. Dynamic simulation analyzers have considerably longer run times than a static timing analysis tool like the Radiant software TRACE program.

**Note**

You cannot generate text fixtures in the Radiant software. All simulation must be done outside of the Radiant software using simulation files in a supported simulator. For guidelines using third-party EDA simulators with Radiant software, see your vendor documentation or the following references.

**See Also** "Timing Simulation" on page 99

# Third-Party Simulators

This section explains how to use Aldec Active-HDL, Aldec Riviera Pro, Cadence NC-Verilog, Cadence NC-VHDL, Cadence NCSim, Mentor Graphics ModelSim/Questa, and Synopsys VCS software to simulate designs that

target Lattice Semiconductor FPGAs. It shows you how to use these simulators to perform functional register-transfer-level (RTL) simulation and place-and-route gate-level simulation with and without timing simulation.

**Note**

Lattice Semiconductor does not supply the Synopsys VCS, Cadence NC-Verilog, Cadence NC-VHDL, Cadence NCSim, Aldec Riviera Pro, or Mentor Graphics ModelSim/Questa simulators. You must obtain them independently.

**See Also** ▶"Simulation Library Files" on page 101

- "Performing Simulation with Aldec Riviera Pro" on page 102
- "Performing Simulation with Aldec Active-HDL" on page 105
- "Performing Simulation with Cadence NC-Verilog" on page 106
- "Performing Simulation with Cadence NC-VHDL" on page 108
- "Performing Mixed-Language (Verilog and VHDL) Simulation with Cadence NCSim" on page 110
- "Performing Simulation with Mentor Graphics ModelSim/Questa" on page 111
- "Performing Simulation with Synopsys VCS" on page 112
- "Performing Mixed-Language (Verilog and VHDL) Simulation with Synopsys VCS-MX" on page 112
- "Performing Mixed-Language (Verilog and SystemVerilog) Simulation with Synopsys VCS-MX" on page 113
- "" on page 114

# Simulation Library Files

This section describes where to find the source simulation library files for FPGA device families. In addition to source files, compiled versions of the simulation library files are available for Aldec Active-HDL Lattice Edition (LE). When you install Active-HDL LE, pre-compiled libraries are automatically installed in the Radiant software installation directory.

*There are no pre-compiled library files available for ModelSim/Questa*. If you choose to associate your Radiant software project with the ModelSim/Questa simulator in the Simulation Wizard, you need to precompile all of the source library files in ModelSim/Questa before using the Simulation wizard.

Verilog library source files are also installed in the cae_library directory. These source files are always installed, whether or not you install Active-HDL LE.

The sources of some of the library models are not provided, so Lattice delivers them as black-boxes. This means for those models, Lattice provides either pre-compiled libraries (ModelSim/Questa) or encrypted HDL sources (Riviera Pro, NCSim, and VCS). For further details, refer to "" on page 114.

**Verilog Source Library Files** The Verilog functional and behavioral simulation library files are installed with the Radiant software in the locations shown in Table 6.

**Table 6: Verilog Functional and Behavioral Simulation Library File Locations**

| FPGA Family | Library Location |
|---|---|
| iCE40 UltraPlus | <*install_dir*>/cae_library/simulation/verilog/ice40UP |

**Compiled Verilog Libraries for Aldec Active-HDL** Refer to Table 7, which shows the location of the compiled Verilog simulation libraries for Aldec Active-HDL.

**Table 7: Aldec Active-HDL Compiled Verilog Simulation Libraries**

| Library Name | Directory Tree | Files | Devices |
|---|---|---|---|
| ice40UP<br>ice40UP_sp | <*install_dir*>/active-hdl/Vlib/src/ice40UP | *.v | ice40 UltraPlus |

**Compiled Mixed-Language Libraries for Aldec Active-HDL** Refer to Table 8, which shows the location of the compiled mixed-language simulation libraries for Aldec Active-HDL.

**Table 8: Aldec Active-HDL Compiled Mixed-Language Simulation Libraries**

| | | | |
|---|---|---|---|
| pmi_work | <*install_dir*>/active-hdl/vlib/pmi_work | *.v | Not device-specific. These are for parameterized module Instantiation. |

**See Also** "Third-Party Simulators" on page 100

# Performing Simulation with Aldec Riviera Pro

This section explains how to perform simulation with the Aldec Riviera Pro simulator.

**Creating or Updating Lattice Semiconductor's FPGA Vendor Libraries**
Create VHDL or Verilog libraries by following the procedures in this section.

**Creating VHDL Libraries** If you do not have Lattice Semiconductor's FPGA VHDL libraries as pre-compiled vendor libraries, the following steps are required to create them:

1. Create Lattice Semiconductor's FPGA VHDL libraries:

```
# create new VHDL libraries
alib ice40UP
```

**Note**

The VHDL library names are strict.

2. For each of the above VHDL libraries compile the source files by using the `acom` command:

   ```
   acom -work <library_name> -reorder <vhdl_lib_src_folder>/
   *.vhd
   ```

   where:

   - *<vhdl_lib_src_folder>* is the folder containing VHDL source files for the appropriate Lattice Semiconductor library *<library_name>*.

3. Set the status of the newly created library as read-only to prevent accidental overwriting:

   ```
   setlibrarymode -ro <library_name>
   ```

**Creating Verilog Libraries** If you do not have Lattice Semiconductor's FPGA Verilog libraries as pre-compiled vendor libraries, the following steps are required to create them:

1. Create Lattice Semiconductor's FPGA Verilog libraries:

   ```
   # create new Verilog libraries
   alib ovi_ice40UP
   ```

   **Note**

   The Verilog library names are not strict, but they conform to Aldec's naming convention for Verilog vendor libraries.

2. For each of the above Verilog libraries, compile the source files by using the alog command:

   ```
   alog -quiet -work <library_name> <verilog_lib_src_folder>/
   *.v
   ```

   where <*verilog_lib_src_folder*> is the folder containing the Verilog source files for the <*library_name*> library.

   If you want to place the debugging information in the library, use the –dbg switch. Debugging may slow down simulation.

   **Note**

   Ignore the warning messages.

3. Set the status of the newly created library as read-only to prevent accidental overwriting:

   ```
   setlibrarymode -ro <library_name>
   ```

**Updating Lattice Semiconductor's Vendor Libraries** If you obtained the updated source code of Lattice Semiconductor's FPGA libraries, you can update the VHDL libraries, Verilog libraries, or both by using the following procedure (for each one of the VHDL or Verilog libraries):

1. Using the cd command, navigate to the directory where your vendor library resides. The vendor libraries are usually located in the <*install_dir*>/vlib folder when pre-compiled by Aldec.
2. Set the library mode to read-write by using the setlibrarymode command:

   ```
   setlibrarymode -rw <library_name>
   ```

3. Compile the source code into the library. Use the acom command for VHDL source files and the alog command for Verilog source files.

   ```
   acom -work <library_name> -reorder <library_src_folder>/
   *.vhd
   ```

   or

   ```
   alog -quiet -work <library_name> <library_src_folder>/*.v
   ```

   **Note**

   For the iCE40up VHDL library, you must perform both of the above steps.

4. Set the library to the read-only mode to prevent accidental overwriting:

   ```
   setlibrarymode -ro <library_name>
   ```

   **Note**

   You can enter the commands either in command-line mode or in the graphical console. Alternatively, you can write a macro that will prepare and compile the libraries, as follows:

   1. Include in the macro the appropriate commands for the libraries that you want to create or update.
   2. Run the runvsimsa script (Linux) or the runvsimsa.bat batch file (Windows) command with the macro as a command-line argument.

**Performing Verilog Simulation** Use the procedures described in this section to perform a Verilog simulation.

**Functional RTL Simulation** Use the following commands to perform a functional RTL simulation with one of the libraries shown in Table 6:

```
alib work
alog -v2k -work work <design_name>.v <test_bench>.v
asim work.<test_bench>
```

**Note**

If you have a pre-existing work library, you can clear its contents by using the following command:

```
adel -lib work -all
```

**See Also** "Third-Party Simulators" on page 100

# Performing Simulation with Aldec Active-HDL

You can use the same procedure as in "Performing Simulation with Aldec Riviera Pro" on page 102 for creating or updating Lattice Semiconductor's FPGA (vendor) libraries and simulating Verilog or VHDL designs in Active-HDL, with a few modifications. Refer to the *Active-HDL On-line Documentation* for more details.

Another way to update the Lattice Semiconductor FPGA libraries with Active-HDL is to obtain the pre-compiled Lattice Semiconductor FPGA libraries directly from the following directory:

`<Radiant_installation_folder>/active-hdl/Vlib`

You can either copy these libraries, along with the Library.cfg file, to your Active-HDL Vlib folder and overwrite any old libraries, or you can modify the Active-HDL Vlib/Library.cfg file to map to the Lattice Semiconductor FPGA libraries installed with the Radiant software. Alternatively, you can perform mapping with the amap command.

**Running Aldec Active-HDL in Stand-Alone Mode** If you are running Active-HDL in stand-alone mode—that is, you are not accessing it through the Radiant software graphical user interface—you should do the following to avoid elaboration (ELBREAD) errors when selecting top-level unit or compiling files from the design pop-up menu:

1. From the top-level menu, select **Design > Settings**.
2. Under Category in the left pane of the Design Settings dialog box, select **General > Compilation > Verilog**.
3. In the right (Verilog) pane under Verilog libraries, click the **Add Library** button  .
4. Select the reference Verilog libraries that are needed in your design. You can select multiple libraries by holding the CTRL key.
5. Click **OK**.
6. Click **OK** in the Design Settings dialog box.

This design setting also adds the -l <*verilog_library_name*> option to the vlog command, which then recognizes (or allows you to set) the top-level unit.

After setting the top-level unit (or testbench), you can start the simulation by clicking **Simulation > Initialize Simulation** from the top-level menu.

**Note**

If you are compiling and simulating your design by using a .do file, the steps just given are not needed. Instead, you should add -l with vlog, add -L with vsim, or both in your .do file.

If you are running Active-HDL in stand-alone mode and you are using at least one of the ASIC blocks, such as PCS or SYSBUS in your design, you must do the following:

1. Change the directory to the design folder where the PCS or SYSBUS configuration file resides:

   ```
   cd <design_folder>
   ```

2. Set the simulation directory to the design (current) folder:

   ```
   set SIM_WORKING_FOLDER .
   ```

Setting the SIM_WORKING_FOLDER option forces the Active-HDL simulator to look for the configuration or input files in the design folder (by default Active-HDL looks for input files in the <*work*> and <*work*>/src folders only, where <*work*> is the Active-HDL working library folder). If you do not set this option, the simulation will fail and Active-HDL will issue an error about a missing configuration file.

This option is also necessary if your simulation is expecting any input files in the design folder rather than in the <*work*> or <*work*>/src folders, unless you place the input file path in your HDL code.

**See Also** "Third-Party Simulators" on page 100

# Performing Simulation with Cadence NC-Verilog

This section explains how to perform simulation with Cadence NC-Verilog.

**Setting Lattice Semiconductor Libraries** Before simulating Lattice Semiconductor FPGA designs in the Cadence NC-Verilog simulator, the following must be done to specify the Lattice Semiconductor simulation libraries.

**Creating Library Definition Files** Before using the NC-Verilog simulator to simulate your design project, you must first create two library definition files named hdl.var and cds.lib in your project folder. The hdl.var and cds.lib files define which libraries are accessible and where they are located. The hdl.var file contains optional statements that can be used to define variables such as the default work library, and the cds.lib file contains statements that map logical library names to their physical directory paths.

For example, your local hdl.var file may include the following:

```
DEFINE work work
```

Your local cds.lib file can include the following:

```
DEFINE scm_vlog <compile_dir>/ice40UP_vlog
```

```
DEFINE work ./work
```

**Note**

*<compile_dir>* refers to the location of the compiled device libraries.

Cadence provides a utility called NCLaunch to set up the necessary initialization files and to compile the Verilog source libraries. NCLaunch is available as part of the 2.1 and later releases. Otherwise, setting up the initialization files and compiling the Verilog source libraries is a manual process.

You can create the hdl.var and cds.lib files with any text editor. You must map the physical locations to the logical names before proceeding to the next step. On the Linux platform, you can create these names by using the mkdir command as follows:

```
mkdir -p <compile_dir>/ice40UP_vlog
```

If you want the logical library names to be available for all designs, use INCLUDE or SOFTINCLUDE in the location of your master hdl.var and cds.lib files.

For example, you can add the following line to your master cds.lib file:

```
INCLUDE $path/cds.lib
```

**Note**

The *path* variable specifies the location of your own cds.lib file.

Or, you can directly include the library definitions in the master hdl.var and cds.lib files.

The master hdl.var and cds.lib files now include the Lattice Semiconductor library definitions. The next time you want to simulate Lattice Semiconductor designs in the NC-Verilog simulator, you do not need to recreate your own hdl.var and cds.lib files.

**Parsing and Analyzing Lattice Semiconductor Simulation Libraries**
After creating your own hdl.var and cds.lib files, you must parse and analyze the Lattice Semiconductor simulation libraries by using the NC-Verilog simulator. These libraries must be parsed and analyzed only once.

To parse and analyze Lattice Semiconductor Verilog simulation libraries, run the following commands in the NC-Verilog simulator:

- iCE40 UltraPlus Verilog:

  ```
  ncvlog -messages -work ice40UP_vlog
  $Radiant_install_path/cae_library/simulation/verilog/
  ice40up/*.v
  ```

**Functional RTL Simulation** Use the following commands to perform a functional RTL simulation with one of the libraries shown in Table 6:

```
rm –rf work
mkdir work
ncvlog -work work <design_name>.v <test_bench>.v
ncelab -lib_binding –access +rwc work.<test_bench>
ncsim -GUI work.<test_bench>
```

**Note**

If you are using NCSim version 5.4 or older, remove the –lib_binding option from the ncelab command.

**See Also** "Third-Party Simulators" on page 100

# Performing Simulation with Cadence NC-VHDL

This section explains how to perform simulation with Cadence NC-VHDL.

**Setting Lattice Semiconductor Libraries** Before simulating Lattice Semiconductor FPGA designs in the Cadence NC-VHDL simulator, you must perform the following steps to set the Lattice Semiconductor simulation libraries.

**Creating Library Definition Files** Before using the Cadence NC-VHDL simulator to simulate your design project, you must first create two library definition files named hdl.var and cds.lib in your project folder. The hdl.var and cds.lib files define which libraries are accessible and where they are located. The hdl.var file contains optional statements that can be used to define variables such as the default work library. The cds.lib file contains statements that map logical library names to their physical directory paths.

For example, your local hdl.var file can include the following:

```
DEFINE work work
```

Your local cds.lib file may include the following:

```
DEFINE ice40up <compile_dir>/ice40UP_vhdl
DEFINE work ./work
```

**Note**

<*compile*_dir> refers to the location of the compiled device libraries.

Cadence provides a utility called NCLaunch to set up the necessary initialization files and to compile the VHDL source libraries. NCLaunch is available as part of the 2.1 and later releases. Otherwise, setting up the initialization files and compiling the VHDL source libraries is a manual process.

You can create the hdl.var and cds.lib files with any text editor. You must map the physical locations to the logical names before you proceed to the next

step. On the Linux platform, you can create these names by using the mkdir command as follows:

```
mkdir -p <compile_dir>/ice40UP_vhdl
```

If you want the logical library names to be available for all designs, use INCLUDE or SOFTINCLUDE in the location of your master hdl.var and cds.lib files.

For example, you can add the following line to your master cds.lib file:

```
INCLUDE $path/cds.lib
```

**Note**

The *path* variable specifies the location of your own cds.lib file.

Or, you can directly include the library definitions in the master hdl.var and cds.lib files.

The master hdl.var and cds.lib files now include the Lattice Semiconductor library definitions. The next time you want to simulate Lattice Semiconductor designs in the NC-VHDL simulator, you do not need to recreate you own hdl.var and cds.lib files.

In addition to defining Lattice Semiconductor VHDL libraries, you must map the vital2000 logical library to the IEEE library location by adding the following line to your cds.lib file:

```
DEFINE vital2000 <NC_VHDL_Libraries_Folder>/IEEE
```

**Note**

The <*NC_VHDL_libraries_folder*> variable specifies the folder containing the NC-VHDL libraries.

**Parsing and Analyzing Lattice Semiconductor Simulation Libraries**
After creating your own hdl.var and cds.lib files, you must parse and analyze the Lattice Semiconductor simulation libraries by using the NC-VHDL simulator. These libraries must be parsed and analyzed only once.

To parse and analyze Lattice Semiconductor VHDL simulation libraries, run the following commands in NC-VHDL simulator.

- iCE40 UltraPlus VHDL:

```
ncvhdl -messages -work ice40up -smartorder
$Radiant_install_path/cae_library/simulation/vhdl/ice40up/
src/*.vhd

ncvlog -messages -work ice40up $Radiant_install_path/
cae_library/simulation/vhdl/ice40up/src/*.v
```

**Functional RTL Simulation** Use the following commands to perform a functional RTL simulation with one of the libraries shown in Table 2:

```
rm –rf work
mkdir work
ncvhdl -work work <design_name>.vhd <test_bench>.vhd
ncelab -lib_binding –access +rwc work.<test_bench>
ncsim –GUI work.<test_bench>
```

**Note**

If you are using NCSim version 5.4 or older, remove the –lib_binding option from the ncelab command.

If you receive warning messages during compilation, see "Warning Messages Caused by Std_Logic Declarations" on page 110.

**Warning Messages Caused by Std_Logic Declarations** When you set vital_level1 as an attribute for the architecture, Cadence NC-VHDL issues several warning messages during compilation about an unapproved type in the vital2000 library models. These messages are caused by Cadence NC-VHDL's strict interpretation of std_logic signal declarations in the architecture as signal types. You can make this interpretation less strict by using the following option:

```
ncvhdl -relax
```

This option does not affect the actual simulation.

**See Also** "Third-Party Simulators" on page 100

# Performing Mixed-Language (Verilog and VHDL) Simulation with Cadence NCSim

The steps involved in performing Verilog/VHDL mixed-language simulation with Cadence NCSim are straightforward and use most of the information provided in the previous two sections ("Simulation Library Files" on page 101 and "Performing Simulation with Cadence NC-VHDL" on page 108):

1. Set up your local cds.lib and hdl.var files to include all the libraries that you use in your design. Each library can be Verilog only, VHDL only, or a mixed-language library containing both VHDL entities and Verilog modules. The libraries can be Lattice Semiconductor FPGA libraries, protected libraries (for example, special-block or IP libraries), or your own design libraries.
2. Compile your Verilog files, using the ncvlog command.
3. Compile your VHDL files, using the ncvhdl command.
4. Elaborate the top-level unit (or test bench) in your mixed-language design, using the ncelab command.
5. Simulate the top-level unit (or test bench), using the ncsim command.

# Performing Simulation with Mentor Graphics ModelSim/Questa

This section explains how to perform simulation with the Mentor Graphics ModelSim/Questa simulator.

**Setting Lattice Semiconductor Libraries** Before simulating Lattice Semiconductor FPGA designs in the Mentor Graphics ModelSim/Questa simulator, you must compile the Lattice Semiconductor simulation libraries using the cmpl_libs command available from the Radiant software's TCL console.

For details about using this command, refer to "Simulation Libraries Compilation Tcl Command" on page 618.

**To enable the Radiant software's batch interface to ModelSim/Questa, change the default simulation tool setup and directory reference as follows:**

1. Choose **Options > Environment Options**.
2. Click on the **Directories** tab.
3. Change the path setting to the ModelSim/Questa executable on the tab to the new version.
4. Click **OK**.

**Mapping to ModelSim/Questa Libraries** The library compilation target folder should also be your simulation working directory. Otherwise, you need to copy the library mappings from ModelSim/Questa.ini at the target folder (or the file itself) into your local or master ModelSim/Questa.ini.

After running simulation libraries compilation using the cmpl_libs TCL command, a ModelSim/Questa.ini file will be created at the target folder chosen for the compiled libraries. That file will have the proper mappings for these compiled libraries.

**Performing Verilog Simulation** Use the procedures described in this section to perform a Verilog standalone simulation with ModelSim/Questa.

**Functional RTL Simulation** Use the following commands to perform a functional RTL simulation with one of the libraries shown in Table 6:

```
vlib work
vlog -work work <design_name>.v <test_bench>.v
vsim work.<test_bench>
```

**Note**

If you have a pre-existing work library, you can clear its contents by using the following command:

```
vdel -lib work -all
```

**See Also** "Third-Party Simulators" on page 100

# Performing Simulation with Synopsys VCS

This section explains how to perform simulation with Synopsys VCS for Verilog-only designs.

For VHDL designs, refer to "Performing Mixed-Language (Verilog and VHDL) Simulation with Synopsys VCS-MX" on page 112.

For SystemVerilog designs, refer to "Performing Mixed-Language (Verilog and SystemVerilog) Simulation with Synopsys VCS-MX" on page 113.

**Functional RTL Simulation** Use the following vcs command to perform a functional RTL simulation with one of the libraries shown in Table 6:

```
vcs -RI <design_name>.v <test_bench>.v
-y <Lattice_verilog_library_location> +libext+.v +v2k
```

**See Also** "Third-Party Simulators" on page 100

# Performing Mixed-Language (Verilog and VHDL) Simulation with Synopsys VCS-MX

In order to run mixed-language simulation with VCS, you must have a special license for VCS-MX. Contact Synopsys if you do not have one.

"Performing Simulation with Synopsys VCS" on page 112 describes how to simulate a Verilog-only design by using the vcs command. In order to simulate a VHDL-only or a Verilog/VHDL mixed-language design, extra steps are required (with extra commands). In general, if your design contains VHDL source code, you must analyze it by using the vhdlan utility. And, if your VHDL design instantiates a Verilog design within it, you must use the vlogan utility. Mixed-language designs are categorized into either VHDL-top or Verilog-top designs, where each one has its own procedure for running the simulation with VCS-MX tools. For more details, refer to the *VCS MX/VCS MXi User Guide*.

Following is an example of the procedure used for simulating a Verilog/VHDL mixed-language design that consists of your own VHDL design files, the iCE40up VHDL library source files, and the iCE40 UltraPlus black boxes encrypted Verilog source file.

1. Create local folders for the iCE40up and the work libraries:

   ```
   mkdir ice40up work
   ```

2. Create (or modify) your local synopsys_sim.setup file, and add the following lines:

```
WORK > DEFAULT
DEFAULT : ./work
ice40up: ./ice40UP_vhdl
```

3. Analyze the iCE40up VHDL library:

```
vhdlan –work ice40UP <ice40UP_vhdl_src_folder>/
ice40UPCOMP.vhd \
<ice40UP_vhdl_src_folder>/ice40UP_SEQ.vhd \
<ice40UP_vhdl_src_folder>/ice40UP_IO.vhd \
<ice40UP_vhdl_src_folder>/ORCA_CMB.vhd \
<ice40UP_vhdl_src_folder>/ORCA_MEM.vhd \
<ice40UP_vhdl_src_folder>/ORCA_MISC.vhd \
<ice40UP_vhdl_src_folder>/ORCA_LUT.vhd \
<ice40UP_vhdl_src_folder>/gsr_pur_assign.vhd
vlogan –work ice40UP <ice40UP_vhdl_src_folder>/*.v
vhdlan –work ice40UP <ice40UP_vhdl_src_folder>/
ice40UP_SL.vhd
```

   where <*ice40UP_vhdl_src_folder*> refers to <*install_dir*>/cae_library/simulation/vhdl/ice40UP/src.

4. Analyze the Lattice black box modules from the encrypted Verilog source:

```
vlogan –work ice40UP <install_dir>/cae_library/simulation/
blackbox/ice40UP_black_boxes-vcs.vp
```

5. Analyze your VHDL design and test bench files:

```
vhdlan <design_name>.vhd <test_bench>.vhd
```

6. Compile the simulation executable from your top-level unit (test bench):

```
scs –mhdl <test_bench>
```

7. Simulate your test bench by running the simulation executable:

```
scsim
```

**See Also** "Third-Party Simulators" on page 100

# Performing Mixed-Language (Verilog and SystemVerilog) Simulation with Synopsys VCS-MX

In order to run mixed-language simulation with VCS, you must have a special license for VCS-MX. Contact Synopsys if you do not have one.

"Performing Simulation with Synopsys VCS" on page 112 describes how to simulate a Verilog-only design by simply using the vcs command. In order to simulate a Verilog/SystemVerilog mixed-language design, extra steps are required. In general, if your design contains SystemVerilog code, then you need to first compile the Lattice library sources and/or encrypted modules

using the vlogan utility. For more details about running mixed-language designs, refer to the *VCS MX/VCS MXi User Guide*.

Following is an example of the procedure used for simulating a Verilog/ SystemVerilog mixed-language design that consists of your own SystemVerilog design files, modules from the MachXO2 Verilog source files and the machxo2_black_boxes encrypted Verilog source file.

1. Create a local folder for the work library:

   ```
   mkdir work
   ```

2. Create (or modify) your local synopsys_sim.setup file, and add the following lines:

   ```
   WORK > DEFAULT
   DEFAULT : ./work
   ```

3. Analyze the machxo2_black_boxes modules from their encrypted Verilog source file:

   ```
   vlogan +v2k <install_dir>/cae_library/simulation/blackbox/
   machxo2_black_boxes-vcs.vp
   ```

4. Compile your SystemVerilog files and simulate your design with top-level unit (test bench):

   ```
   vcs -mhdl -RI <design_name>.v <test_bench>.v -y
   <install_dir>/cae_library/simulation/Verilog/machxo2
   +libext+.v
   ```

**See Also** "Third-Party Simulators" on page 100

# Applying Design Constraints

Constraints are instructions applied to design elements that guide the design toward desired results and performance goals. They are critical to achieving timing closure or managing reusable intellectual property (IP). The most common constraints are those for timing and pin assignments, but constraints are also available for placement, routing, and many other functions.

Constraints can include timing or physical constraints defined in Constraint Files (.ldc/.pdc/.sdc/.fdc) or HDL attributes (for physical pin locking) using the Radiant software and third party synthesis tools. Files .ldc/.pdc/.sdc are used for the synthesis tool and specify design goals. Synthesis, map, and place-and-route work to meet these goals. Post-synthesis constraints (.pdc) can also be specified. The flow combines these based on different entry points with in the Radiant software design flow. The timing analysis tool reports whether or not the goals were met.

**See Also** ▶"Constraints Reference Guide" on page 403

- "Multiple Entry Constraint Flow" on page 115
- "Integrated Synthesis" on page 166
- "Applying Lattice Pre-Synthesis Timing Constraints" on page 123

## Multiple Entry Constraint Flow

Constraints from multiple sources are used and modified in the Radiant software in multiple ways, including the following timing constraints:

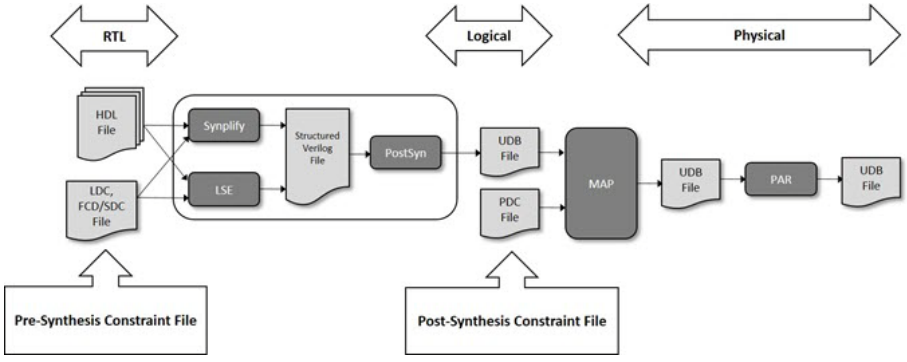
- .ldc (via Lattice Synthesis Engine (LSE))
- .sdc/.fdc (Synopsys Synplify Pro)
- .pdc constraint files

You can set constraints using one or more of the following methods:

- Assigning HDL attributes in the HDL source files. After synthesis and translation, these defined attributes can be viewed and modified in Radiant software views, and the modifications can be saved to the design constraint file.
- Using the LSE, you can assign pre-synthesis constraints using the Timing Constraint Editor and save them to an .ldc constraints file. If you are using Synplify Pro, assign .sdc constraints using the SCOPE editor tool and save them as an.sdc file. Any pre-synthesis constraints that are created by Synplify should be edited in Synplify Pro.
- Assigning physical constraint using a text editor or Radiant software editing views (Device Constraint Editor, and Floorplan View). Post-synthesis timing constraints are assigned via the Post-Synthesis Timing Constraint Editor. Both timing and physical constraints are stored in the .pdc file.

The following figure shows a high-level flow of how constraints from multiple sources can be used and modified in the Radiant software.

**Figure 23:**




**See Also** "Constraints Reference Guide" on page 403

- "Using Radiant Software Pre-Synthesis Constraints" on page 122
- "Device Constraint Editor" on page 127

# Constraint Implementation

The following steps illustrate how you might assign constraints and implement them at each stage of the design flow, using synthesis constraints and timing analysis.

1. Define the constraints in the pre-synthesis stage using one or both of the following methods:
   - Define constraints as attributes within the HDL.
   - Define timing constraints using Synplify Pro or LSE. Constraints are saved in .ldc file.

2. Synthesize the design.

   When synthesis is performed, the synthesis process synthesizes the design and stores it in a unified database.

   The timing constraints will be displayed in Radiant software Design Constraint Editor View and color-coded as synthesis values.

3. Open one or more of the following views in the Timing Constraint Editor or Device Constraint Editor to define new constraints:

   **(Pre/Post Synthesis) Timing Constraint Editor**– Timing constraints, exceptions, delays and synthesis attributes including physical pin locations can be defined in these two tools.

   **Device Constraint Editor**– Define signaling standards and make pin assignments. Assign clocks to primary or secondary routing resources. Define groups of ports only with current release. Other global settings such as temperature and voltage can be set. Run Constraint design rule checking.

   **Floorplan View** – View the device layout. Draw bounding boxes for GROUPs. Draw REGIONs for the assignment of groups or to reserve areas. Reserve sites and REGIONs that should be excluded from placement and routing. Run Constraint design rule checking.

4. Save the constraints to the .pdc file.
5. Run the **Map Design** process (map).

   This process reads all data from the unified design database and processes from there.

6. Run the **Place & Route Design** process (par).
7. Open one of the following views, as desired, to examine timing and placement.

   **Timing Analysis View** – Examine details of timing paths. Cross-probe selected paths to Floorplan View and Physical View.

8. Modify constraints or create new ones using any of the design / timing constraint editing views. Save the changes and rerun the **Place & Route Design** process.
9. Repeat Steps 3 through 8 until design performance objectives are met.

**See Also** ►"Analyzing Static Timing" on page 194

# Understanding Implications of Radiant Software Constraint Flows

In order to help understand the Radiant software constraints flow process, Figure 49 below details the entry mechanism for input files and its data flow through the constraints process. The blue boxes indicate design entry mechanism in the form of HDL and synthesized via our internal LSE or third party Synplify Pro synthesis tools. Other yellow boxes also indicate processes run by Radiant software. Green boxes are used to indicate input or output

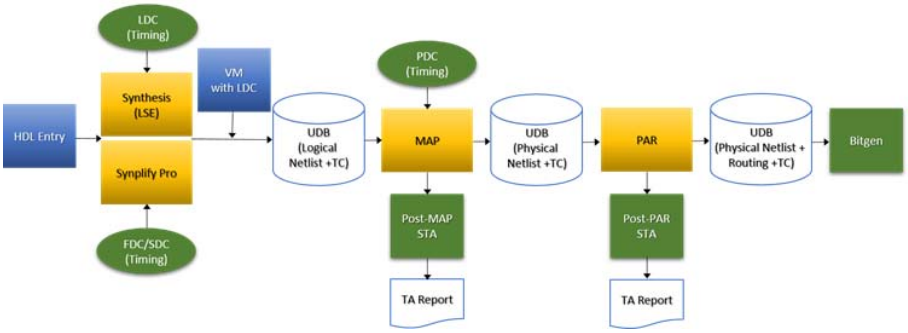
constraints files or related output report files such as Static Timing Analysis reports out of Map and PAR. Note that Radiant software also generates a post-synthesis timing report just for LSE.

The important process to note is that the Unified Database (.udb) used to store the processed data dictates that certain stages of the constraints need not be re-run in the tool saving processing time. For example, physical constraints on RTL entered as attributes in the HDL, or pre-synthesis constraints entered using the Pre-Synthesis Timing Constraint Editor/Synplify Pro SCOPE (or text editor) including .ldc/.fdc (.sdc converted to .fdc) timing constraints would initially run through synthesis and the data stored is in the .udb file. Subsequent timing and physical constraints entered using tools such as the Post-Synthesis Timing Constraint Editor, Floorplan View or Device Constraint Editor and stored in the .pdc file are processed via the mapping run without the need to re-synthesize the design since the pre-synthesis data resides in the .udb database. As the constraints flow proceeds with each process (i.e. synthesis, Map, PAR) the .udb successively stores a physical netlist, routing and timing constraint.

Transparent to the user, the storage of accumulating data throughout the constraints flow in the .udb and reduction in re-running processes is one reason for the improved speed and efficiency for Radiant software users.

In terms of the necessity of Pre versus Post synthesis timing constraints, usage depends on the complexity and desired performance of the design results. A simple design requiring few timing constraints with a relaxed FMax may require only pre-synthesis timing constraints and runs through the flow from synthesis to place and route. A complex design requiring a higher FMax performance may have initial pre-synthesis timing constraints entered for synthesis. Later in the flow, a user may then want to override or fine tune these timing constraints in addition to adding physical constraints to reach the desired performance by driving the place and route process. One example would be a user who initially specifies a higher FMax to reduce logic levels, then later in the post-synthesis flow lowers the Fmax constraint so that the place and route process is not strained when routing the design.

**Figure 24:**



## Managing Constraints: .ldc/.sdc/.fdc vs .pdc

The pre-synthesis timing constraints .ldc/.sdc/.fdc are passed to the .udb and shown in the Post-Synthesis Timing Constraint Editor as read only. If a user attempts to further re-constrain these constraints, the constraints are saved to the .pdc file.

If two identical constraints exist in the .ldc and .pdc files respectively, the constraint in the .pdc file takes precedence.

## Constraints Precedence

Regarding physical / timing constraints, constraints entered via any GUI tool such as Device Constraint Editor or Timing constraint editor will take precedence over the constraints hat were entered via a HDL attribute in the RTL or .ldc TCL sdc command.

Constraints entered later in the design flow such as Post-Synthesis Timing Constraint editor (.pdc) will override a constraint entered in the Pre-Synthesis Constraint (.ldc) editor tool

If there are conflicts in the constraint file such as a ldc_set_location TCL command conflicting with a ldc_prohibit on the same location, then the Radiant software will issue an error message in the design flow.

# Timing and Physical Constraints

There are three types of constraints that are supported in Radiant software for constraining your design for maximum performance: Lattice Design Constraints, Post-Synthesis Design Constraints, and Synopsis Design Constraints.

**Lattice Design Constraints (.ldc):** 1A combination of both Synopsys Design Constraints (.sdc version 2.0) and the Radiant software's synthesis attributes. Files have .ldc suffix.

Refer to **Constraints Reference Guide > Lattice Synthesis Engine Constraints > Synopsys Design Constraints** in the online help for more details.

**Post-Synthesis Design Constraints (.pdc):** The Radiant software design constraints contain both physical constraints and any post synthesis timing constraints. These files have .pdc suffix.

Below is a supported list:

▶ "ldc_create_group" on page 479

- “ldc_create_region” on page 479
- “ldc_create_vref” on page 479
- “ldc_set_location” on page 480
- “ldc_set_vcc” on page 480
- “ldc_set_port” on page 480
- “ldc_set_sysconfig” on page 481
- “ldc_set_attribute” on page 481
- “ldc_prohibit” on page 482

Refer to **Constraints Reference Guide > Lattice Synthesis Engine Constraints > Synopsys Design Constraints** in the online help for more details.

**Synopsys Design Constraints (.sdc):** Synopsys design constraints are industry standard timing constraints. Files with .sdc suffix. Its currently in its version 2.0 iteration.

Below is a supported list:

- “create_clock” on page 419
- “create_generated_clock” on page 421
- “set_clock_groups” on page 422
- “set_false_path” on page 425
- “set_input_delay” on page 425
- “set_max_delay” on page 426
- “set_min_delay” on page 427
- “set_multicycle_path” on page 428
- “set_output_delay” on page 429
- “set_clock_uncertainty” on page 424
- “Lattice Synthesis Engine-Supported HDL Attributes” on page 430

# Migrating from Former Lattice Diamond Preferences

New to the Radiant software flow are Lattice Design Constraints (.ldc/.pdc). These are a combination of both Synopsys Design Constraints (.sdc version 2.0) and the Radiant software proprietary physical constraints.

All Lattice Diamond constraints were known as Preferences and were defined as .ldc or .pdc files written as .sdc constraints. Lattice Diamond uses a combination of “preferences” and constraints, while Radiant software uses only .ldc constraints. All Lattice Diamond preferences have been converted to constraints in the Radiant software.

The use of .sdc constraints involve some object access commands as indicated below:

- get_cells
- get_pins
- get_nets
- get_ports
- all_inputs
- all_outputs
- get_clocks
- all_clocks

Figure 9 shows the most commonly used Lattice Diamond preferences for applying physical constraints and how to apply them with the new Radiant software.sdc commands. Listed in alphabetical order, examples are also shown on how to convert Lattice Diamond preferences to a Radiant software .ldc constraint.

**Table 9: Lattice Diamond Preferences Compared to Radiant Software Constraints**

| Lattice Diamond Preference Usage | Radiant Software .ldc Constraint Usage |
|---|---|
| BANK 1 VCCIO 3.3 V | ldc_set_vcc -bank 1 3.3 |
| BLOCK PATH FROM PORT “LOCAL_CMD0” TO PORT “RAM_RD”; | set_false_path -from [get_ports LOCAL_CMD0] -to [get_ports RAM_RD] |
| CLOCK_TO_OUT ALLPORTS OUTPUT_DELAY 10ns CLKPORT CKIN1; | set_output_delay 10 -clock [get_clocks CKIN1] [get_ports all_outputs] |
| DEFINE BUS “busA” NET “pen0” NET “pen1” NET “pen2” NET “pen3” NET “pen4” NET “pen5” NET “pen6” NET “pen7”; | ldc_create_group -name busA [get_nets pen0 pen1 pen2 pen3 pen4 pen5 pen6 pen7] |
| DEFINE PORT GROUP “grp1” “a1” “a2” “a3”; | ldc_create_group -name grp1 [get_ports a1 a2 a3] |
| FREQUENCY NET “clk1” 100 Mhz | create_clock -period 10 -name clk1 [get_nets clk1] |
| GROUP “rot_grp” BBOX 8 3 BLKNAME rotate_1; | ldc_create_group -name rot_grp -bbox 8 3_cells {rotate_1} |
| GSR_NET NET “rst_pcie”; | ldc_define_attribute -attr GSR [get_nets rst_pcie] |
| INPUT_SETUP PORT “in1” INPUT_DELAY 2 ns CLKPORT “clk”; | set_input_delay 2 -clock [get_clocks clk] [get_ports in1] |
| IOBUF PORT bchk IO_TYPE=LVCMOS33; | ldc_set_port -iobuf {IO_TYPE=LVCMOS33} [get_ports bchk] |
| LOCATE COMP “A” SITE “11”; | ldc_set_location -site 11 [get_ports A] |
| LOCATE GROUP “GROUP_0” SITE “R2C2D”; | ldc_set_location -site R2C2D [ldc_get_groups GROUP_0] |
| LOCATE GROUP “xyz_GROUP” REGION “my_region” | ldc_set_location -region my_region [ldc_get_groups GROUP_0] |

**Table 9: Lattice Diamond Preferences Compared to Radiant Software Constraints (Continued)**

| Lattice Diamond Preference Usage | Radiant Software .ldc Constraint Usage |
|---|---|
| LOCATE VREF “VREF1_BANK_3” SITE “n21”; | ldc_create_vref -name VREF1_BANK_3 -site N21 |
| MAXDELAY FROM PORT “LOCAL_CMD0” TO PORT “RAM_RD” 18 NS | set_max_delay 18 -from [get_ports LOCAL_CMD0] -to [get_ports RAM_RD] |
| MINDELAY FROM CELL “state_reg4” TO CELL “RAM_OE_N_reg” 15 NS; | set_min_delay 15 -from [get_cells state_reg4] -to [get_cells RAM_OE_N_reg] |
| MULTICYCLE FROM CLKNET “clk1” TO CLKNET “clk2” 2 X; | set_multicycle_path 2 -from [get_clocks clk1] -to [get_clocks clk2] |
| OUTPUT PRT “A’ LOAD 10 pF; | ldc_set_port -misc LOAD=10pF [get_ports A] |
| PERIOD NET “NetA” 150 NS HIGH 75 NS; | create_clock -period 100 -name clk0 -waveform {0.75 1.50} [get_nets NetA] |
| PERIOD PRT “Clk1” 30 NS; | create_clock -period 30 -name clk1 [get_ports Clk1] |
| PROHIBIT PRIMARY NET “bf_clk”; | ldc_set_attribute PRIMARY=FALSE [get_nets bf_clk] |
| PROHIBIT SITE “AB” | ldc_prohibit -site AB |
| REGION “Region_0” R16C2D” 11 30; | ldc_create_region -name Region_0 -site R16C2D -width 11 -height 30 |
| SYSCONFIG CONFIG_MODE=SLAVE_SERIAL PERSISTENT=OFF DONE_OD=ON; | ldc_set_sysconfig CONFIG_MODE=SLAVE_SERIAL PERSISTENT=OFF DONE_OD=ON |
| TEMPERATURE 45C; | ldc_set_attribute {TEMPERATURE=45C} |
| USERCODE HEX “53656D69”; | ldc_set_attribute {FORMAT=HEX CODE=53656D69} |
| VCC_DERATE PERCENT 5 | ldc_set_vcc -core -derate 5 |
| VCC_NOMINAL 1.2V; | ldc_set_vcc -core 1.2 |
| VCCIO_DERATE BANK 0 PERCENT 5; | ldc_set_vcc -bank 0 -derate 5 |
| VOLTAGE 1.5V; | ldc_set_vcc -core 1.5 |

# Using Radiant Software Pre-Synthesis Constraints

Enter constraints in the synthesis design constraints .sdc file using one of the following two synthesis tools:

- Pre/Post Synthesis Timing Constraint Editor (TCE)

  For more information about entering constraints with TCE, refer to “Applying Lattice Pre-Synthesis Timing Constraints” on page 123.

- Synplify Pro SCOPE Constraints Editor

  For more information about entering constraints with Synplify Pro SCOPE Constraints Editor, refer to “SCOPE Constraints Editor” help in the Synplify Pro online help.

The following Synopsis Design Constraints can be set in the .sdc:

- "create_clock" on page 419
- "create_generated_clock" on page 421
- "set_clock_groups" on page 422
- "set_false_path" on page 425
- "set_input_delay" on page 425
- "set_max_delay" on page 426
- "set_min_delay" on page 427
- "set_multicycle_path" on page 428
- "set_output_delay" on page 429
- "set_clock_uncertainty" on page 424
- "Lattice Synthesis Engine-Supported HDL Attributes" on page 430

.

**Note**

Some IPs such as PLL requires the user to only supply the input reference clock and the Radiant software tool will automatically write out the associated generated clock constraints based on the supplied PLL parameter division ratios.

**See Also** "Applying Lattice Pre-Synthesis Timing Constraints" on page 123

- "Device Constraint Editor" on page 127

# Applying Lattice Pre-Synthesis Timing Constraints

Radiant software LSE enables you to set pre-synthesis Synopsys® Design (formatted) Constraints, which are directly interpreted by the synthesis engine. When you use LSE, these .sdc constraints are saved to a Lattice Design Constraints file (.ldc). You can create several .ldc files and select one of them to serve as the active synthesis constraint file for an implementation.

The Source Editor and the Timing Constraint Editor are available for creating and editing .ldc files. Timing Constraint Editor provides a spreadsheet style user interface that enables you to quickly create and edit Synopsys Design Constraints.

Timing Constraint Editor runs a design rule check (DRC) to verify that the constraints you enter are legitimate. These checks are done in real time as you enter them in the GUI.

See "Synopsys Design Constraints" on page 416 for descriptions of the .sdc constraints that are supported by the LSE and Timing Constraint Editor.

**See Also** "Creating a New .ldc Synthesis Constraint File" on page 124

- "Integrated Synthesis" on page 166
- "Optimizing LSE for Area and Speed" on page 164

## Creating a New .ldc Synthesis Constraint File

Before you create a new .ldc file, verify that LSE has been selected as the synthesis tool. In the Processes tab, "Lattice Synthesis Engine" should be listed under "Synthesize Design." If it is not listed, choose **Project > Active Implementation > Select Synthesis Tool**, and select **Lattice LSE**.

To make sure that the .ldc file opens in Timing Constraint Editor instead of Source Editor, verify that Timing Constraint Editor has been selected as the default program in **Tool > Options > General > File Associations**.

**To create a new .ldc synthesis constraint file:**

1. Choose **File > New > File**.

   Alternatively, right-click the **Timing Constraint Files** folder and choose **Add > New File**.

2. In the New File dialog box, select **Source Files** from the Categories list. Select **Pre-Synthesis Constraint Files** from the Source Files list.
3. Type a name for the new .ldc file and specify the directory location.
4. Click in the **Add to Implementation** box if you want to use the file with the currently active implementation.
5. Click **New**.

   Timing Constraint Editor opens and displays the spreadsheet and three tabs for creating and editing synthesis constraints. If you selected the **Add to Implementation** option, the new .ldc file will appear in the Timing Constraint Files folder.

   Pulldown menus for source elements are provided when you double-click the Clock Source cells or the Inputs/Outputs Ports cells.

6. Enter or edit values for clocks, inputs and outputs, and delay paths:
   a. Double-click a cell and type a new value, or select a value from the pulldown list.
   b. Use the arrow keys or the Tab key to move to another cell and select it for editing.
   c. New rows are inserted automatically. You can delete existing rows with data entered by right clicking and selecting **Remove Row** from the pop-up menu.
7. Choose **File > Save**.

**See Also**
- "Defining Clocks Using Timing Constraint Editor" on page 138
- "Setting Input and Output Delays Using Timing Constraint Editor" on page 139
- "Defining Timing Exceptions Using Timing Constraint Editor" on page 140

- "Synopsys Design Constraints" on page 416
- "Managing Constraint Files" on page 13
- "Integrated Synthesis" on page 166
- "Optimizing LSE for Area and Speed" on page 164

## Using Source Editor for .ldc Files

Any .ldc file that you have created using the Timing Constraint Editor can be viewed and edited in Source Editor. Likewise, an .ldc file that you have created in Source Editor can be viewed and edited in the Timing Constraint Editor.

One advantage of using Source Editor is that it allows you to open multiple .ldc files at once, whereas Timing Constraint Editor will load only one .ldc file at a time.

**To open an .ldc file in Source Editor:**

1. In the File List view, expand the Timing Constraints Folder.
2. Right-click the name of the .ldc file you want to open and choose **Open With**.
3. In the dialog box, select **Source Editor** and click **OK**.

   The .ldc file opens in Source Editor window as an ASCII file.

**See Also** "Changing the Default Editor for .ldc Files" on page 125

## Changing the Default Editor for .ldc Files

The Radiant software gives you the option of setting Source Editor or Timing Constraint Editor as the default program for .ldc files.

**To change the default editor for .ldc files:**

1. Choose **Tools > Options**.
2. In the General section of the Options dialog box, select **File Associations**.
3. In the Extensions column of File Associations, scroll down to **ldc and/or ldc<active>**.
4. In the Default Programs column, choose **Source Editor** from the pulldown menu. The **ldc** option also allows for Add Program, an option to open with your choice of text editors.

   The next time you create or select an .ldc file, it will open in the editor you selected as the default.

**See Also** "Managing Constraint Files" on page 13

## Optimizing for Timing

The pre-synthesis constraints that you set in the .ldc file will drive optimization of the design if you set the optimization goal for timing in the active strategy file.

**To set the optimization goal for timing:**

1. From the File List view, expand the Strategies folder and double-click the name of the active strategy.
2. Select **LSE** from the Synthesis Design folder in the Process pane in the Strategies dialog box.
3. In the LSE pane on the right, scroll down to Optimization Goal and select **Timing** in the Value column.
4. Click **OK**.

**See Also** ▶"Synopsys Design Constraints" on page 416

▶ "Optimizing LSE for Area and Speed" on page 164

## Adding an .ldc File to an Implementation

You can add multiple .ldc files to an implementation and view each of them in Timing Constraint Editor.

**To add an existing .ldc file to an implementation:**

1. In the File List view, right-click the Timing Constraints Files folder and choose **Add > Existing File**.
2. Browse to the directory that contains the desired .ldc file, select the file, and click **Add**.

   The file is added to the implementation and is displayed in the Timing Constraint Files folder list.

**See Also** ▶"Managing Constraint Files" on page 13

## Activating an .ldc File

To apply the synthesis constraints, you need to identify an .ldc file as the active one for the implementation.

**To activate an .ldc file for an implementation:**

- Right-click the file name and choose **Set as Active .ldc**.

  By default, the active .ldc file is compiled when you synthesize the design.

**See Also** ▶"Managing Constraint Files" on page 13

# Using Radiant Software Tools

The Radiant software provides tools for editing design constraints. These tools, which are available from the Radiant software toolbar and Tools menu, include the following:

- "Device Constraint Editor" on page 127
- "Timing Constraint Editor" on page 136
- "Floorplan View" on page 143
- "Physical View" on page 148

The Radiant software constraints tools enable you to develop constraints that shorten turn-around time and achieve a design that conforms to critical circuit performance requirements.

This section describes these Radiant software constraints tools and how to use them.

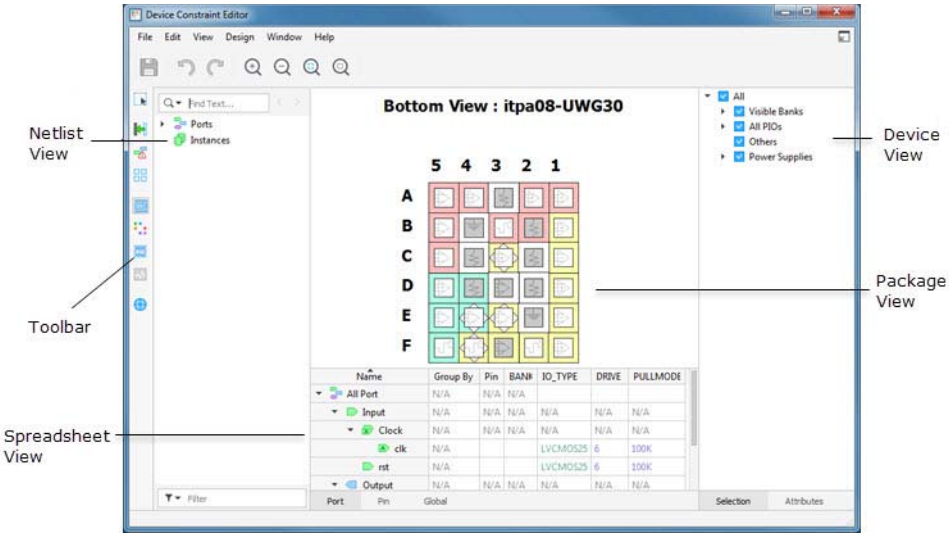
## Device Constraint Editor

Device Constraint Editor is used to edit .pdc constraints. These constraint editing views are available from the Radiant software toolbar and Tools menu.

All modified constraints will be saved to a .pdc file and the flow will return to Map. The Device Constraint Editor shows the pin layout of the device and displays the assignments of signals to device pins. This view allows the user to edit these assignments, reserve sites on the layout to exclude from placement and routing. The Device Constraint Editor is also the entry mechanism for physical constraints.

Device Constraint Editor views enable you to develop constraints that will shorten turn-around time and achieve a design that conforms to critical circuit performance requirements.

The following figure shows the Device Constraints Editor .




The Device Constraints Editor allows you to do the following tasks:

- Define physical constraints based on design signals and logic elements.
- Define sysIO Buffer properties based on design signals.
- Constrain clock signals to particular routing spines.
- View, assign, and validate design signals and package pins.
- Assign voltage reference pins.
- Perform DRC tests to ensure legal pin assignments or to detect incomplete or illegal conditions within the physical design database.
- Back annotate post-route pin assignments to a logical constraint file.

## Viewing Pin, Port and Global Assignments in Device Constraint Editor

The Device Constraint Editor has a spreadsheet view that provides a tabular format for viewing and assigning logical constraints. It displays all constraints, including those that have been assigned.

As soon as you have specified the target device, you can use this view to set global constraints. After synthesis and translation, you can use all of the following constraint sheets to create and modify constraints:

**Port** The Port tab provides a signal list of the design and shows any pin assignments that have been made.

By default, the ports are grouped by direction—Input, Output, and Bidirectional—plus an “Others” category that includes user-defined port groups. This allows you to quickly focus on Outputs, for example, by closing the display of Input ports or vice versa.

The Port tab enables you to assign or edit pin locations and other attributes by entering them directly on the spreadsheet. It also allows you to assign multiple signals, by right-clicking the selected signals and opening the Assign Ports dialog box.

**Pin** The Pin tab provides a pin list of the device and shows the signal assignments that have been made. In the Dual Function column, pins that can be used for I/O assignments or for another function, such as Vref, are identified.

You can show differential pin pairs by selecting **View > Show Differential Pairs**.

The Pin tab enables you to edit signal assignments or assign new signals by right-clicking selected pins and opening the Assign Pins dialog box.

**See Also** ▶"Assigning Signals" on page 149

▶ "Setting Port, Net, and Cell Attributes" on page 153

## Viewing Package View and Pin Layout in the Device Constraint Editor

The Device Constraint Editor has a Package View that shows the pin layout of the device and displays the assignments of signals to device pins. The Package View interacts with the Netlist View, which is a listed on the right side of the screen, to assign pins that enable you to drag selected signals to the desired locations on the pin layout.

Each pin that is assigned with a constraint is color-coded to indicate the port direction of the related signal port. The Package View allows you to edit these assignments, and it allows you to reserve sites on the layout that you want to exclude from placement and routing.

The Netlist view displays the RTL netlist objects including ports, instances and nets from RTL view. It will be docked in the package view for easy signal assignment. Selected signals from this view.

**See Also** ▶"Modifying Signal Assignments using the Device Constraint Editor" on page 152

▶ "Using the Zoom and Pan Tools in Device Constraint Editor" on page 133

## Using the "Show" Commands

When Device Constraint Editor is open, the View menu provides two groups of "Show" commands. The first group consists of commands that are mutually exclusive; when one command is activated, the others become inactive. The second group consists of commands that are not mutually exclusive; they can be activated at the same time that another "Show" command is active.

In addition to the View menu, the “Show” commands are available from the pop-up menu when you right-click a blank space on the Package View layout.

**Mutually Exclusive Commands** Only one command can be used at a time:

- Show Bank IO
- Show Port Group
- Show DQS Group

**Non-mutually Exclusive Commands** The following commands can be activated while another “Show” command is active:

- Display IO Placement
- Show Differential Pairs

## Displaying Bank I/O Assignments

By default, Package View displays any assigned I/Os within each color-coded bank. However, the bank I/O display is turned off when the display for Port groups or DQS groups is turned on.

**To display bank I/Os:**

- In Device Constraint Editor, choose **View > Show Bank IO**.

  The I/O assignments are displayed within each bank. Port groups and DQS groupsare hidden.

## Displaying Port Groups in Device Constraint Editor

Device Constraint Editor enables you to view the placement of assigned port groups on the pin layout. Each port group is color-highlighted with the same distinctive color that is displayed in the Group sheet of the Spreadsheet View.

**To display port groups:**

- In Device Constraint editor, choose **View > Show Port Group**.

  The color-coded port groups are displayed. Bank I/O assignments and DQS groups are hidden.

## Displaying DQS Groups in Device Constraint Editor

You can view the placement of DQS groups in color-coded format on Package View’s pin layout by turning on the “Show DQS Groups” command.

**To display DQS Groups:**

- In Device Constraint editor, choose **View > Show DQS Group**.

The color background of each DQS group is displayed. Bank I/O assignments and port groups are hidden.

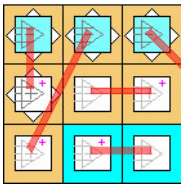
## Displaying Differential Pin Pairs in Device Constraint Editor

The “Show Differential Pairs” command enables you to identify all the differential pin pairs on a layout. When this command is activated, fly wires are displayed that connect the differential pin pairs. Each unplaced positive differential pin displays a small magenta-colored cross in the upper right corner. By default, these indicators are hidden.

**To display differential pin pairs:**

- In Device Constraint Editor, choose **View > Show Differential Pairs**.

  The layout displays the fly wires connecting differential pins and the cross indicator for each positive differential pin.



  The “Show Differential Pairs” command acts as a toggle key and displays a check mark when the differential pair fly wires are displayed. Select **View > Show Differential Pairs** to turn off the display.

**See Also** “Prohibiting Incompatible Pins” on page 154

## Displaying I/O Placement Assignments in Device Constraint Editor Package View

After placement and routing, Package View enables you to examine all pin assignments, including those made by Place & Route (PAR). A rotated square in the background of each placed pin serves as a placement indicator when the “Display IO Placement” command is turned on. The pin assignments are displayed with a color fill to distinguish them from PAR-assigned pins.

**To display I/O Placement assignments in Package View:**

- In Package View, choose **View > Display IO Placement** or click the Display IO Placement button on the toolbar. The border color around the box indicates the Bank info which is shown in the right hierarchy view.Pin assignments that originated from the .pdc file are displayed with a color fill. Blue is output signal; green is input signal.

The “Display IO Placement” command acts as a toggle key and displays a check mark when the assignments are displayed. Select the command again to turn off the display.

When you edit pin assignments after placement and routing and save them, the placement indicators are cleared from Device Constraint Editor. The “Display IO Placement” command is then unavailable until you rerun Place & Route.

## Changing Device Constraint Editor Default Color Coding

The color coding in Package View is used to differentiate each bank and each type of pin. You can easily view the significance of each color by opening the Color Legend by going to **Tool > Options, Color Tab** on the left and Device tab in the main pain. The Color Legend Tab also enables you to change the default color coding for banks and pin types.

**To change the default color coding for banks and pins:**

1. Go to **Tool > Options, Color** tab on the left and **Device** tab in the main pain.
2. Double-click the color box under the Color column of the desired bank or pin type listed in the Name column that you want to change.
3. In the Select Color dialog box, select a different color and click **OK**.
4. If you want to change more than one color, click **Apply** in the dialog box, and then repeat Steps 3 and 4. To revert to the default colors, click **Restore**.
5. When you have finished color editing, click **Close**.

## Filtering the Display of Banks and Pin Types in Device Constraint Editor

When you open Device Constraint editor, it shows all banks and pin types by default. In the Radiant software, the window to the right of the main device view shows the banks and pin types so that only selected banks and pin types are displayed via the Selection and Attributes tabs.

**To filter the display of banks and pin types in Package View:**

1. On the right hand side select Selection tab, you can check or uncheck the different types of IOs.

**See Also** ▶“Using the “Show” Commands” on page 129

## Reversing the Pin Layout View in Device Constraint Editor

You can reverse the view of Package View 's device pin layout from a bottom to top perspective, or vice versa.

**To reverse the pin layout view:**

- From the Device Constraint Editor, choose **View > Top View** or **View > Bottom View**.

  The View command acts as a toggle key and changes to its opposite, depending on the current view.

## Using the Zoom and Pan Tools in Device Constraint Editor

The Radiant software toolbar provides zoom tools for the Device Constraint Editor. In addition, the vertical toolbar included with Package View and Floorplan View provides commands that work together with the zoom tools to help you navigate to different objects and areas of the layout. Also, both Physical View and Logic Block View provide an area zoom that is controlled through the left mouse button. All of the views enable you to zoom in and out using the mouse wheel and function key combinations, and to pan using the arrow keys.

**Radiant Software Toolbar Zoom Commands** The following commands are available on the Radiant software toolbar and in the View menu for each layout view.

 Zoom In – enlarges the view of the entire layout.

 Zoom Out – reduces the view of the entire layout.

 Zoom Fit – reduces or enlarges the entire layout so that it fits inside the window.

 Zoom To– enlarges the view to only show a selected pin.

**Area Zoom with the Mouse in Device Constraint Editor and Floorplan View** In Physical View and Logic Block View, you can enlarge an area of the layout by holding the left mouse button and dragging to the right and downward. Afterwards, you can zoom out of the area by dragging your mouse upward and to the left. Area Selection Mode – enables you to select objects on the layout by clicking an individual object or by dragging around an area to select multiple objects. It also enables you to move an assignment to a different location by dragging it. After selecting objects, you can zoom in with the Zoom To command. The Select command is activated by default. When Select is active, the Pan and Zoom Area commands are disabled.

**Zooming with Function Key Shortcuts** The following key combinations, available for all the layout views, enable you to instantly zoom in or out from your keyboard:

- Zoom In – Ctrl++ (numeric keyboard)
- Zoom Out – Ctrl+- (numeric keyboard)

**Zooming with the Mouse Wheel** The mouse wheel gives you finer zoom control, enabling you to zoom in or out in small increments. While pressing the **Ctrl** key, move the mouse wheel forward to zoom in and backward to zoom out. The mouse wheel zoom is available for all the layout views.

**Panning with the Arrow Keys** After enlarging the layout with the zoom tools, you can use the keyboard arrow keys to pan horizontally or vertically to different areas of the layout. Panning with the arrow keys is available for all the layout views.

## Searching for Design Elements in Device Constraint Editor

Device Constraint Editor and Floorplan View enable you to search for design elements—either through a Find dialog box or through Find and Filter text boxes within the view.

Each provide Find and Filter text boxes at the top of the window. Radiant software highlights the found objects and enables you to navigate through them and cross-probe to other views.

Device Constraint Editor and Floorplan View each provide a Find dialog box. The Find dialog box in Floorplan View enables you to search for components, nets, and sites.

**To search for design elements in Device Constraint Editor:**

1. Open the desired view.
2. The Find text box appears at the top of the window.
3. To narrow the scope of the search, use wildcards in the following ways.
   - Use the asterisk to match one or more characters.
   - Use the question mark to match any single character.

   The view displays only those items that contain the characters you entered.
4. To locate a single design element from the list, type the name in the Find text box and press **Enter**.

   The found element is brought to the top of the view and highlighted.
5. To navigate through a list of elements that contain the same characters, use wildcards in the Find text box, and then press **Enter**.

   The first element containing the characters you entered is brought to the top of the view and highlighted. Press **Enter** again to go to the next element in the list.

## Using Drag-and-Drop to Make Assignments

Radiant software constraint-editing views provide a convenient drag-and-drop feature for making assignments or editing them. This feature is available between Device Constraint Editor and Floorplan View. The drag-and-drop feature is also available within Package View and within Floorplan View. This feature allows you to perform the following actions:

- Drag signals from Netlist View to Package View to assign them to pin locations all within Device Constraint Editor.
- Drag assigned signals within Device Constraint Editor to reassign them to other locations.
- Drag EBR, DSP, PCS, PLL, and ASIC block instances from Netlist View to Floorplan View all within Device Constraint Editor to set constraints.
- Drag non-SLICE logical components within Floorplan View to reassign them to other locations.
- Drag GROUPs and REGIONS within Floorplan View to reassign them to other locations.

## Undoing and Redoing Assignments

Radiant software **Undo** and **Redo** commands allow you to reverse constraint changes that were made in a selected constraint view. You can make multiple assignments, undo them all, and then redo them as desired. When all of your manual assignments have been undone or redone, the Undo or Redo command becomes dim. When you save your constraint changes, the undo and redo commands are disabled.

**Note**

The **Undo** command will not remove the in-memory constraint change indicators: the asterisk from a constraint view tab or the “constraints Modified” from the status bar. To remove these indicators after all assignments have been undone, use the **Save** command.

The commands are available from the Edit menu and from the Undo  and Redo  buttons on the Radiant software toolbar.

The standard Windows keyword shortcuts are also available:

- Press **Ctrl+Z** to undo an assignment.
- Press **Ctry+Y** to redo an assignment.

## Printing a Constraint View from Device Constraint Editor

You can print any from the Device Constraint Editor and Floorplan View. The Radiant software provides a preview for each view, enabling you to select options before printing.

**Printing a Layout View** You can print a copy of the layout from Floorplan View or Device Constraint Editor.

**To print a layout:**

1. Select the layout view that you want to print.
2. Choose **File > Print Preview**.
3. Select from the toolbar options at the top, and then click the **Printer** icon to print.

# Timing Constraint Editor

The Timing Constraint Editor tool is used to enter all the timing constraints in the Radiant software tool, as well as to edit .ldc/.pdc constraints. There are a total eight tabs to enter different constraint types:

1. Clock - define the clocking scheme of the design.
2. Generated Clock - define generated clocks such as from PLLs.
3. Clock Latency - define the latency in terms of Rise, Fall times.
4. Clock Uncertainty - define From and To constraints.
5. Clock Group - used for complex clocking schemes and groups in terms of being Logically / Physically Exclusive and Asynchronous groups.
6. Input/Output Delay - set Input and Output delays.
7. Timing Exception - set Min / Max, False and Multicycle path constraints.
8. Attribute - set synthesis attributes.

The above will be described in more detail below.

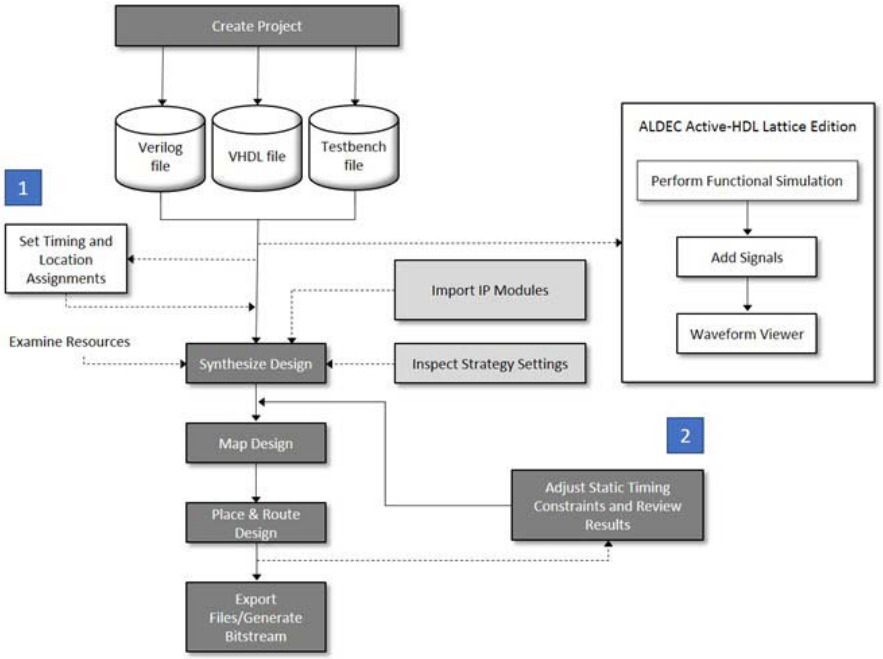
## Pre / Post Synthesis Timing Constraint Editor

You will notice under **Tools > Timing Constraint Editor**, there are there are two selections:

1. **Pre-Synthesis Timing Constraint Editor**
2. **Post-Synthesis Timing Constraint Editor**

The two GUIs are identical and the entry mechanism of the constraints also the same. The key difference as noted in the tool name is that pre-synthesis are (..ldc) constraints are entered pre-synthesis and will be synthesized by the chosen synthesis tools. Post-Synthesis constraints (..pdc) will populate pre-synthesis timing constraints and allow for entering physical as well as post-synthesis timing constraints.The post-synthesis Timing Constraint Editor is the same GUI with the varying constraints already synthesized and populated in each of the different constraints tab. A user cannot modify the constraints in the post synthesis constraints that were populated from pre-synthesis, but a

user can supply new constraints to either one, override an existing, or supply totally new constraints to better constrain the design for better performance upon analysis.




## Defining Synthesis Constraints Using Timing Constraint Editor

Timing Constraint Editor presents the contents of a Lattice Design Constraints File (.ldc) in a spreadsheet style format. You can use Timing Constraint Editor to open and edit a current .ldc file or create a new one. Individual sheets are provided for defining clocks, setting input and output delays, and defining delay paths for multicycle, minimum and maximum delays, and false paths.

By default, Timing Constraint Editor is launched when you create a new .ldc file. Each sheet of Timing Constraint Editor allows you to define synthesis constraints by double-clicking a cell and selecting or typing a value.

Each row includes an Enabled check box, which is selected by default. If you clear this check box, the constraint is ignored by LSE. This can be an easy way to test different settings.

**See Also** ▶"Creating a New .ldc Synthesis Constraint File" on page 124

- "Defining Clocks Using Timing Constraint Editor" on page 138
- "Setting Input and Output Delays Using Timing Constraint Editor" on page 139
- "Defining Timing Exceptions Using Timing Constraint Editor" on page 140
- "Defining Clock Groups in Timing Constraint Editor" on page 141

- "Defining Generated Clocks in Timing Constraint Editor" on page 141
- "Setting Attributes in Timing Constraint Editor" on page 142
- "Adding an .ldc File to an Implementation" on page 126
- "Synopsys Design Constraints" on page 416

## Running Timing Constraint Editor

Launch the Timing Constraint Editor by opening or creating an .ldc file for an implementation.

**Note**

In order to open or create an .ldc file, LSE must be the selected synthesis tool for the project. Choose **Project > Active Implementation > Select Synthesis Tools** and select **Lattice LSE**.

You can have multiple .ldc files for an implementation, but only one can be opened in Timing Constraint Editor at a time.

**To run Timing Constraint Editor, do one of the following:**

- Double-click an .ldc file name in the Timing Constraints Files folder in the File List view.
- If you have set Source Editor to be the default program for .ldc files, right-click the .ldc file name in the File List view and choose **Open With**. In the Open With dialog box, choose **Timing Constraint Editor** and click **OK**.
- If an .ldc file doesn't yet exist for the active implementation, create an .ldc file as described in Creating a New .ldc Synthesis Constraint File.

**See Also** "Managing Constraint Files" on page 13

- "Using Source Editor for .ldc Files" on page 125

## Defining Clocks Using Timing Constraint Editor

The Clock tab of Timing Constraint Editor enables you to define an alias to be associated with an existing clock port or net from the source file. You can also define clocks by dragging-and-dropping from Netlist Analyzer into Timing Constraint Editor.

**To define a clock using Timing Constraint Editor:**

1. Double-click the **Object Clock** cell to select an existing clock pin, port, or net. A CLOCKPIN is applied on instances and (CLOCKPORT) ports is applied to top level modules. Clocks coming from the outside should ideally be defined on a port.
2. If desired, enter an alias for the clock in the Clock Name cell.
3. Enter a clock period in nanoseconds in the Period(ns) cell.

4. If desired, you can specify the duty cycle. Double-click the Waveform cell and enter the length of the high portion followed by low portion of the cycle in nanosecond. The default duty cycle is 50%. The waveform option is used to control the arrival times of the positive and negative edges and the duty cycle. The waveform edges have to be within the period of the clock Enter the values with at least one decimal i.e. 5.0;2.0.

**To define a clock in Timing Constraint Editor using Netlist Analyzer:**

1. Start Netlist Analyzer and detach it by clicking the detach tool in the upper right corner. Refer to "About Netlist Analyzer" on page 30.
2. Select desired clock object (port or net) from Netlist Analyzer and drag-and-drop into Clock Object cell of the Timing Constraint Editor Clocks tab.
3. If desired, enter an alias for the clock in the Clock Name cell.
4. Enter a clock period in nanoseconds in the Period(ns) cell.
5. If desired, you can specify the duty cycle. Double-click the Waveform cell and enter the length of the high portion followed by low portion of the cycle in nanosecond. The default duty cycle is 50%. The waveform option is used to control the arrival times of the positive and negative edges and the duty cycle. The waveform edges have to be within the period of the clock Enter the values with at least one decimal i.e. 5.0;2.0.

**See Also**

- "create_clock" on page 419
- "Creating a New .ldc Synthesis Constraint File" on page 124
- "About Netlist Analyzer" on page 30


## Setting Input and Output Delays Using Timing Constraint Editor

The Inputs/Output Delay tab of Timing Constraint Editor enables you to specify input and output delays relative to a clock. You can also define input and output delays by dragging-and-dropping from Netlist Analyzer into Timing Constraint Editor.

**To set an input or output delay using Timing Constraint Editor:**

1. Double-click the **Constraint Type** cell to select the type of delay (input or output).
2. Select from the input or output ports in the Port cell.
3. In the Clock cell, select an existing clock or an alias name that has been defined for a clock.
4. Select the Clock Fall check box if you want to clock on negative edge.
5. Select either Max or Min or neither. Do not select both. You can also select Add Delay
   - Max means that the delay value refers to the longest path.
   - Min means that the delay value refers to the shortest path.

- Neither means that the maximum and minimum delays are assumed to be equal.
- Add Delay is used to define more than one input delay on a given port.

6. Enter a delay value in nanoseconds in the Value (ns) cell.

**To set an input or output delay in Timing Constraint Editor using Netlist Analyzer:**

1. Start Netlist Analyzer and detach it by clicking the detach tool in the upper right corner. Refer to "About Netlist Analyzer" on page 30.
2. Select from the input or output ports from Netlist Analyzer and drag-and-drop into the Port cell of the Timing Constraint Editor Input/Outputs tab.
3. In the Clock cell, select an existing clock or an alias name that has been defined for a clock.
4. Select the Clock Fall check box if you want to clock on negative edge.
5. Select either Max or Min or neither. Do not select both. You can also select Add Delay
6. Enter a delay value in nanoseconds in the Value(ns) cell.

**See Also**

- "Defining Clocks Using Timing Constraint Editor" on page 138
- "set_input_delay" on page 425
- "set_output_delay" on page 429
- "Creating a New .ldc Synthesis Constraint File" on page 124
- "About Netlist Analyzer" on page 30


## Defining Timing Exceptions Using Timing Constraint Editor

The Timing Exception tab of Timing Constraint Editor enables you to define a Multicycle path, specify a Max_Delay or a Min_Delay for a timing path, and identify a False path that is to be excluded from timing analysis. You can also perform the same functions by dragging-and-dropping from Netlist Analyzer into Timing Constraint Editor.

**To define a timing exception using Timing Constraint Editor:**

1. Double-click the **Constraint Type** cell and select the type of delay.
2. Specify the path information, delay, and cycles as appropriate for the selected delay type.

**To define a delay path in Timing Constraint Editor using Netlist Analyzer:**

1. Start Netlist Analyzer and detach it by clicking the detach tool in the upper right corner. Refer to "About Netlist Analyzer" on page 30.

2. Select object (port, net, or register) from Netlist Analyzer and drag-and-drop into the From or To cell of the Timing Constraint Editor timing exception tab.
3. Specify the path information, delay, and cycles as appropriate for the selected delay type.

**See Also** ▶"set_multicycle_path" on page 428

- "set_max_delay" on page 426
- "set_min_delay" on page 427
- "set_false_path" on page 425
- "Creating a New .ldc Synthesis Constraint File" on page 124
- "About Netlist Analyzer" on page 30

## Defining Clock Groups in Timing Constraint Editor

The Clock Group tab of Timing Constraint Editor enables you to define clock groups that are logically / physically exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during timing analysis.

**To define clock groups using Timing Constraint Editor:**

1. Double-click the **Group** cell and select the appropriate clock group.
2. Now you will be able to select logically / physically or asynchronous type.

**See Also** ▶"set_clock_groups" on page 422

- "Creating a New .ldc Synthesis Constraint File" on page 124

## Defining Generated Clocks in Timing Constraint Editor

The Generated Clock tab of Timing Constraint Editor enables you to define internally generated clocks. You can also perform the same functions by dragging-and-dropping from Netlist Analyzer into Timing Constraint Editor.

**To define generated clocks using Timing Constraint Editor:**

1. In the **Source** cell, select the source clock for the generated clock.
2. Specify the other values as appropriate for the generated clock.

**To define generated clocks in Timing Constraint Editor using Netlist Analyzer:**

1. Start Netlist Analyzer and detach it by clicking the detach tool in the upper right corner. Refer to "About Netlist Analyzer" on page 30.

2. Select the source clock from Netlist Analyzer and drag-and-drop into the Source cell of the Timing Constraint Editor Generated Clocks tab.
3. Specify the other values as appropriate for the generated clock. You can also specify the Master Clock and Object values by dragging from Netlist Analyzer.

**See Also** ▶"create_generated_clock" on page 421

▶ "Creating a New .ldc Synthesis Constraint File" on page 124

## Setting Attributes in Timing Constraint Editor

The Attribute tab of Timing Constraint Editor enables you to specify Synplify Lattice Attributes that are supported by the LSE. You can also define objects by dragging-and-dropping from Netlist Analyzer into Timing Constraint Editor.

**To set attributes using Timing Constraint Editor:**

1. Double-click the **Type** cell and select the desired attribute.
2. Specify object, value type, and value, as appropriate for the attribute.

**To set attributes in Timing Constraint Editor using Netlist Analyzer:**

1. Start Netlist Analyzer and detach it by clicking the detach tool in the upper right corner. Refer to "About Netlist Analyzer" on page 30.
2. In Timing Constraint Editor Attribute tab, double-click the **Type** cell and select the desired attribute.
3. Specify object type and value, as appropriate for the attribute.
4. Select object (port, net, or register) from Netlist Analyzer and drag-and-drop into the Object cell of the Timing Constraint Editor Attribute tab.

**Note**

You can also drag-and-drop an object from Netlist Analyzer to the Object cell of Timing Constraint Editor Attribute tab before you specify the attribute. Object type will then be selected automatically. You can then specify attribute and value.

**See Also** ▶"Lattice Synthesis Engine-Supported HDL Attributes" on page 430

▶ "About Netlist Analyzer" on page 30

## Compiling the Design Using Timing Constraint Editor

By default, Timing Constraint Editor is set to compile the design automatically. With automatic compilation, Timing Constraint Editor automatically performs checks as to whether the design data is outdated. This happens whenever the Timing Constraint Editor is launched or activated. If the design data is found to be outdated, the compilation will begin immediately.

# Floorplan View

Floorplan View provides a large-component layout of your design. All connections are displayed as fly-lines. Floorplan View allows you to create REGIONs and bounding boxes for GROUPS and specify the types of components and connections to be displayed. As you move your mouse pointer slowly over the floorplan layout, details are displayed in tool tips: the number of resources for each GROUP and REGION; the number of utilized slices for each PLC component; and the name and location of each component, port, net, and site.

Floorplan View is available as soon as the target device has been specified.

**See Also** ▶"Creating GROUPs" on page 154

▶ "Using the Zoom and Pan Tools in Device Constraint Editor" on page 133

## Displaying Assignments and Connections

The Floorplan View toolbar allows you to select the types of elements that will be displayed on the layout, including IOs, DQS GROUPS, placed components, and component connections. These commands are also available from the View menu.

 Area Selection Mode - used to drag and select sections of the floorplan.

 Display Placement – displays all placed components, including assignments originating from the logical constraint file (.pdc) and those made by Place & Route.

 Display Placement Groups – highlights the placed GROUP component sites with the designated color fill.

 Display Connections To/From Selection(s) – controls the two Display Connections buttons below it. When active, it enables you to activate one or both of these buttons to view the connections between selected components, outside of selected component, or both.

 Display Connections Between Selection(s) - shows the connections between selected components.

 Display Connections Outside of Selection(s) - shows the connections that extend outside of the selected components.

 Display Placement Constraints - shows placement assignments that originated from the ..pdc file. When this button is active, placed components assigned with a LOC constraint are distinguished with a cross-hatch pattern.

 Display Regions – displays any defined REGIONS.

 Display Groups – displays any defined GROUPS.

Display Logical Connections - shows the logical connections between GROUPs, including those assigned to REGIONs, and between GROUP and assigned Constraint sites. The connections are based on the logical instances within each GROUP.

Show World View– toggles the World View of DCE to On or Off

**See Also** ▶"Using the Zoom and Pan Tools in Device Constraint Editor" on page 133

▶ "Viewing Assignments in Floorplan View" on page 144

## Searching for Design Elements in Floorplan View

The Find Text... dialog box in Floorplan View enables you to search for components, nets, and sites.

**To search for components, nets, or sites in Floorplan View:**

1. Choose **Tools > Floorplan View**, and then choose **Edit > Find** or press **Ctrl+F**.
2. In the Find dialog box, type the name of the component, net, or site that you want to find. Use the asterisk and question mark characters to perform wildcard searches.
3. Select the type from the Find Type pulldown menu.
4. Select Match Whole Word and Match Case as desired, to narrow your search to a complete name or to match upper and lower case letters.
5. Click **Find Next** or **Select All**.

   The item or items are highlighted on the layout.

   Use the **Find Next** and **Find Previous** buttons to navigate from one found item to another. Each will be highlighted on the layout.

## Viewing Assignments in Floorplan View

Floorplan View displays assignments at any stage of the design flow. These can include assigned sites, GROUPS, REGIONs, and reserved resources. After routing, all placements assigned by the Place & Route process are displayed.

The "Display" commands on the toolbar or the View menu control the display of assignments. The types of assignments displayed will depend on whether the commands "Display Placement," "Display Placement Constraints," or both are activated.

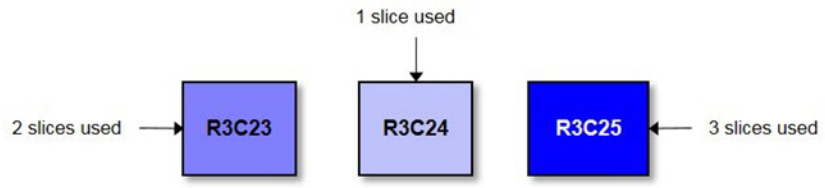
**Assigned Sites** After placement and routing, the site a component has been assigned appears on the layout with a solid fill.

If the assignment originated from the HDL source, the site will be filled and overlaid with a cross-hatch pattern when both “Display Placement” and “Display Placement Constraints” are activated.

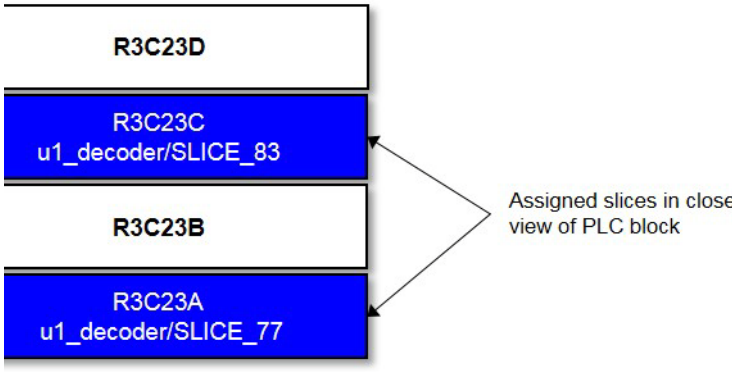
When only “Display Placement Constraints” is activated, an assignment originating from the logical constraint file or HDL source is displayed with the cross-hatch pattern but without the fill.

**Reserved Sites** Sites reserved are displayed with a gray pattern.

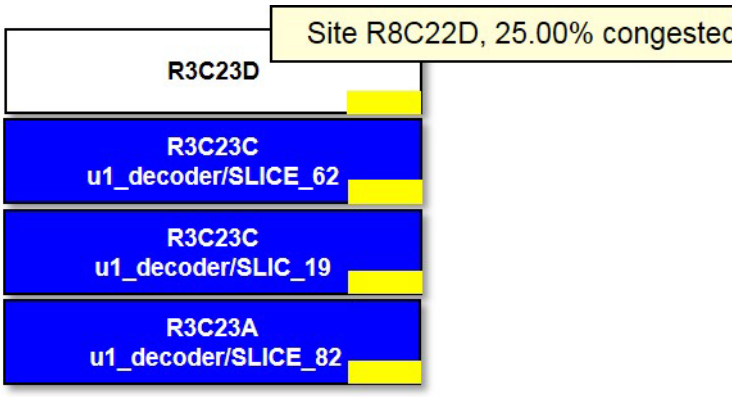
**PLC Blocks (PFU and PFF)** After placement and routing, a programmable logic cell (PLC) in which one or more logical components have been placed will appear on the layout with a solid fill. The color fill will be darker or lighter in shade, depending on how many sites are used.




As you zoom in so that individual slices are visible, only the slices that contain placed logical components are displayed with a color fill.



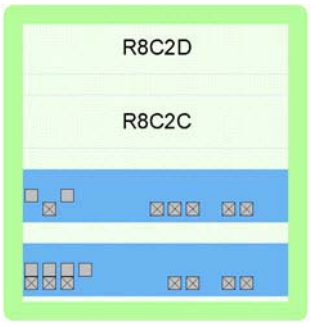

When Display Congestion is turned on, a highlighted rectangle is displayed that represents the amount of congestion for the row and column location. The tool tip shows the precise percentage of congestion.



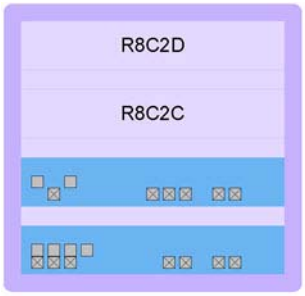

**GROUPS** Anchored GROUPS are displayed when both the “Display Placement Constraints” and “Display GROUPS” commands are activated. An anchored unbounded GROUP is indicated with a small square icon at the upper left of the assigned anchor location. The icon becomes visible as you zoom in.

A bounded GROUP is displayed with a solid border, corresponding to the assigned width and height, and a dotted pattern between blocks. GROUPS are distinguished from one another by their designated colors. After placement, the sites that contain the member elements of the GROUP are

filled with the designated color when the “Display Placement Groups” command is activated.




**REGIONs** REGIONs are displayed when both the “Display Placement Constraints” and “Display REGIONs” commands are activated. A REGION bounding box is shown with a solid border and a solid fill between blocks. It is distinguished from other REGIONs by its designated color. When a REGION is reserved with the ‘Prohibit” command, it is overlaid with a gray dotted pattern.




**See Also** ▶“Displaying Assignments and Connections” on page 143

- “Assigning Signals” on page 149
- “Creating GROUPs” on page 154

## Going to a Specific Site

Floorplan View provides a “Go To Location” dialog box for quick navigation to a specific column and row on the layout.

**To go to a specific site:**

1. In Floorplan View, right-click an empty space on the layout and choose **Go to Location**.
2. In the dialog box, enter the desired Row and Column.
3. Click **OK**.

   Floorplan View zooms in to the location you specified.

**See Also** ▶“Using the Zoom and Pan Tools in Device Constraint Editor” on page 133

- “Searching for Design Elements in Device Constraint Editor” on page 134

## Physical View

Physical View provides a read-only detailed layout of your design that includes switch boxes and physical wire connections. Routed connections are displayed as Manhattan-style lines, and unrouted connections are displayed as flylines. As you move your mouse slowly over the layout, the name and location of each REGION, group, component, port, net, and site are displayed as tool tips.

On the left hierarchy view in this tool, a user can do design exploration by the Logical tab whereby the netlist is hierarchy and signals exist at the module and instance level, or the Physical tab which displays a flattened netlist showing signals in the sub-module level.

**Displaying Components and Connections** The Physical View toolbar allows you to select the types of elements that will be displayed on the layout.

 Displays sites, unused or vacant blocks to which logic can be assigned.

 Displays components, blocks that have assigned logic.

 Displays routes, physical connections between resources on the chip after the design has been routed.

 Displays unrouted nets, those that PAR failed to route or those that could not be routed because a site associated with a net was unlocked by the user.

 Displays the latest timing path selected in Timing Analysis View.

**Highlighting Elements** Use the View menu's Highlight / Unhighlight commands to highlight or unhighlight selected nets or sites.

**Viewing Highlighted Items, Prohibited Sites, and Labels** Use the View menu's Layer commands to display highlighted items, prohibited sites, and the textual labels of sites and components.

**Note**

When the text display has been turned on, a label will sometimes overlap the edge of the symbol or even appear aligned with the symbol below it. However, when you hold the mouse pointer over the symbol, the correct label will appear in the tool tip.

**See Also** ▶"Using the Zoom and Pan Tools in Device Constraint Editor" on page 133

# Applying Radiant Software Physical Constraints

Radiant software constraint-editing views allow you to specify FPGA pre-synthesis constraints without having to write out logical constraints syntax

manually. These views include Spreadsheet View, Package View, Netlist View of Device Constraint Editor and Floorplan View.

# Setting Global Constraints

The Global tab in Spreadsheet View enables you to set constraints that affect the entire design, such as voltage, temperature, global BLOCK constraints, and configuration constraints. The list of these constraints varies by device family.

**To set global constraints:**

1. In Spreadsheet View, select the **Global** tab.
2. To perform a global edit, do one of the following:
   - Double-click the value and enter a new one.

     The table cell changes format. If the cell changes to a text box, type the new value. If it changes to a box with a pulldown menu, choose the value from the list.
   - Right-click the constraint value cell and select the desired value from the pop-up menu. For a text entry, choose **Edit Cell**, and then type the new value.
3. To return to the default value, highlight the value and press the **Delete** key or right-click the value and choose **Clear**.

# Assigning Signals

Radiant software Spreadsheet View and Netlist View enable you to assign signals to pin locations. You can also assign signals by dragging them from Netlist View to pin locations on the Package View layout. Signals can be assigned or reassigned at any stage of the design flow.

## Assigning Signals in Device Constraint Editor Spreadsheet View

Spreadsheet View provides two methods for assigning signals to pin locations:

- Use the Port Assignments sheet to assign pins to listed signals. The Port Assignments sheet displays signals by port type and signal names.
- Use the Pin Assignments sheet to assign signals to listed pins. The Pin Assignments sheet lists pins by pin, pad name, bank, dual function, and polarity.

Both views enable you to make single or multiple assignments.

## Assigning Pins Using the Port Assignments Sheet

For a single pin assignment, double-click the pin cell for a given signal and type the desired pin name. For multiple pin assignments, use the Assign Pins dialog box, as explained below.

**To assign pin locations:**

1. From the Pin column, select the pin cells of the signals to assign.

   To select multiple pin cells, do one of the following:

   - Drag down the pin column or use the **Ctrl+click** or **Shift+click** combination.
   - For a bus, open Device Constraint Editor View and detach it. Select the bus from the Ports list of Netlist View widget and drag it to the Port Assignments Sheet of Spreadsheet View in the lower half. All the rows for the bus signal names become highlighted.

2. Right-click the selected cells in the Pin column and choose **Assign Pins**.

   In the Assign Pins dialog box, the signals you selected are displayed in boldface in the left pane. In the pin list next to it, the same number of pins are displayed in bold at the top of the list. The software automatically interprets the design files and displays the pin filters that should be enabled by default. For example, if you selected an LVDS25 type of port in Spreadsheet View, the polarity section on the right will show the True P-side filter as enabled. If you selected a BLVDS type of port, it shows the Emulated P-side filter as enabled.

3. Clear or select the desired options under Pin Types to narrow or expand the list of available pins, and then select **Auto Sort** or **Manual Sort**.

   - If Auto Sort is selected, select the desired sorting option under “Sort by,” and then select **Ascending** or **Descending** order.
   - If Manual Sort is selected, use one of the following methods to drag pins and place them in the desired order:
     - Click one pin to select it, and then drag it to the desired location.
     - Use the **Shift** key to select several pins at once and release the mouse button. Next, drag the highlighted pins to the desired position.

4. From the pin list, select the pin location where you would like the assignment to begin.

   The dialog box highlights the selected pin and displays in bold type the sequence of pins following it—one for each signal.

   **Note**

   If you have used the “Incompatible Pins” dialog box to select a device for possible pin migration, the incompatible pins will be displayed in a dim font on the pin list. If you have disabled the incompatible pins, The Radiant software will assign the next available pin for each pin that has been disabled.

5. From the Assign Pin pulldown menu at the bottom left, select one of the following Assign Pin options: every, every second, every third, or every fourth pin.
6. To check the validity of the selected pin assignments, click **Check Pins**.

   A message box appears that lists any assignment errors. If errors are reported, repeat Steps 4–6 until the pin check is free of errors.
7. Click **Assign Pins**.

## Modifying Assignments in Device Constraint Editor Spreadsheet View

You can modify pin assignments in Spreadsheet View before or after placement and routing. After routing, you can also modify assignments that automatically made by the Place & Route Process.

**To modify assignments:**

1. From the Port Assignments sheet or Pin Assignments sheet, select the assignments that you want to modify, right-click, and choose **Assign Pins** or **Assign Signals**.
2. Follow the same procedure for assigning pins or signals in Spreadsheet View.

**To modify assignments made automatically by Place & Route:**

1. Choose **View > Display IO Placement**.

   All placed and routed assignments are displayed in parentheses, including user assignments. User assignments appear twice—once without parentheses and once in parentheses—and they are displayed in black font or the font color that has been designated for “Normal” in the options dialog box. Assignments that were made automatically by Place & Route appear only once. They are displayed in parentheses and are distinguished by blue font or the font color designated for “Default Values.”
2. Select the assignments made by Place & Route that you want to modify, right-click, and choose **Assign Pins** or **Assign Signals**.
3. Follow the same procedure for assigning pins or signals in Spreadsheet View.

   The assignments are now displayed in black font. Those that were assigned by Place & Route appear in parentheses, preceded by the new assignments.

**To remove assignments:**

- Select the assigned pins or signals that you want to delete, and press the **Delete** key. Optionally, right-click the selected assignments and choose **Clear**.

  **Note**

  Only user assignments can be deleted. Assignments made automatically by Place & Route can be modified but not deleted.

## Modifying Signal Assignments using the Device Constraint Editor

Device Constraint Editor enables you to use drag-and-drop to move one or more signal assignments to unlocked pins. It also allows you to move a single signal assignment to a pin that is already locked as well as swap the two assignments.

**To modify one or more assignments:**

1. Select the device pins on the layout that contains the assignments you want to modify.

   To select only one pin, click it to highlight it. Use the **Ctrl** or **Shift** key or to select more than one pin, or drag your mouse around multiple pins to select them.

2. Move the mouse toward the desired unlocked pin location.

   As you begin to drag, the cursor displays a crossed-out circle symbol. The cursor changes to an arrow as you move it over valid pins. Keep dragging until you have reached the targeted location.

3. When the cursor displays the arrow symbol over the targeted location, release the mouse button.

   **Note**

   If you are modifying a single assignment, make sure that the pin is highlighted before dragging it. If you try to drag a pin before it is highlighted, you will activate the multiple selection tool.

**To remove assignments:**

1. Select the pins that contain the assignments you want to remove.
2. Right-click the selected pins and choose **Unlock**.

**See Also**


- "Assigning Signals in Device Constraint Editor Spreadsheet View" on page 149
- "Modifying Assignments in Device Constraint Editor Spreadsheet View" on page 151
- "Prohibiting Incompatible Pins" on page 154

## Modifying Signal Assignments in Floorplan View

In Floorplan View, you can modify a signal assignment by dragging it from one Constraint to another. You can also modify an assignment that originated from the logical constraint file (.pdc) or one that was made automatically by Place & Route (PAR).

**To modify a signal assignment in Floorplan View:**

1. Choose **Tools > Floorplan View**.
2. Make sure that the “Display Placement” button  is turned on if you will be modifying a PAR-assigned I/O.
3. In Floorplan view, zoom in so that you can easily view the placed Constraint component that you want to reassign and the targeted location.

   Constraint components that were assigned automatically by PAR are displayed with a solid color fill. Constraint components whose assignments originated from the .pdc file are displayed with a color fill and a cross-hatch overlay.
4. Drag from the assigned site to the targeted location and release the mouse button.

   The previous location is now displayed with a fill and no cross-hatch, and the targeted location is displayed with the cross-hatch and no fill.

**See Also** ▶“IO_TYPE” on page 409

# Setting Port, Net, and Cell Attributes

Spreadsheet View's Port Assignments Sheet, Clock Resource sheet, Route Priority sheet, Cell Mapping sheet, and Global constraints sheet enable you to set attributes for ports, nets, and cells. They allow you to set I/O port attributes, assign net priorities, specify registers for flip-flops, and set junction temperature and voltage.

## Defining I/O Defaults

The Radiant software allows you to set values for all ports in your design at once. You can do this by using the ALLPORTS selection, which is available in the Spreadsheet view section of Device Constraint Editor.

### Defining I/O Defaults Using ALL PORTS

The ALL PORTS row in Device Constraint Editor’s Spreadsheet View's Port Assignments sheet enables you to quickly define a default setting for I/O attributes. It can be especially useful for setting the I/O type default.

**To define default settings Using ALL PORTS:**

1. In Spreadsheet View in the Device Constraint Editor, select the Port tab.

2. Double-click the desired attribute cell in the ALL PORTS row and select a value from the pulldown list. Optionally, right-click the cell and select a value from the pop-up list.

   The newly selected value is displayed in orange, or the font color designated for “Reference Value,” for all of the ports that do not have other values already assigned.

   **Note**

   The range of attribute values available with the ALL PORTS row reflect the intersection of valid values across all port types. To access all available values, use individual ports or port groups.

   You can return to the original default setting by selecting the attribute cell and pressing the **Delete** key.

## Prohibiting Incompatible Pins

Both Spreadsheet View and Package View of Device Constraint Editor enable you to create a PROHIBIT constraints for selected incompatible pins or all incompatible pins. Prohibiting incompatible pins will prevent you from assigning signals to them. It will also prevent the Map and Place & Route processes from using these incompatible pins. When you prohibit incompatible pins, none of the pins that have already been assigned will become unlocked.

**To prohibit one or several incompatible pins:**

- On the Pin Assignment sheet of Spreadsheet View or in Package View, select the incompatible pins that you want to prohibit. Right-click, and choose **Prohibit**.

**To release one or several incompatible pins:**

- On the Pin Assignment sheet of Spreadsheet View or in Package View, select the prohibited pins that you want to release. Right-click and choose **Release**.

## Creating GROUPs

The GRP constraint consists of logical components that are to be packed close together. GROUPs can include PFU, PFF, EBR, and DSP blocks. You can create GROUPs in Spreadsheet View, Netlist View and Floorplan View. After placement and routing, Floorplan View displays the group member elements with the GROUP color fill.

**Note**

A GROUP must be anchored if it includes a DSP block. If such a GROUP is not anchored, the Radiant software will issue a warning and ignore the DSP block.

**Creating GROUPs in Netlist View in Device Constraint Editor To create a GROUP in Spreadsheet View:**

1. Right click on Netlist View on Ports (Input or Output).
2. Click on **Create Port Group**.
3. A new Group dialog appears. You can now move available ports to selected ports.
4. After entering a Group name, this constraint will be saved in the ..pdc file as a ldc_create_group constraint.

**Creating GROUPs in Floorplan View** In Floorplan View, you can draw a bounding box for a new GROUP, and then select the instances to be included. For routed designs, you can create a GROUP from selected components displayed on the layout.

**To create a GROUP bounding box and add instances:**

1. Choose **Tools > Floorplan View**.
2. Zoom in, as needed, to the desired area on the floorplan layout.
3. Right-click and choose **Create GROUP BBox**.

   The mouse pointer changes from an arrow to a cross symbol.
4. Draw a rectangle around the sites where you want to place the GROUP. Make sure that the bounding box does not overlap the bounding box of another GROUP or REGION.

   When you release the mouse button, the Create New GROUP dialog box opens. In the Anchor and BBox sections, it shows the anchor site location and the bounding box size, based on the rectangle you drew on the floorplan. Under GROUP Name, the resources covered by the bounding box are shown: LUTs, registers, and EBRs.

   **Note**

   A GROUP must be anchored in order to be displayed in Floorplan View.
5. In the boxes at the top of the dialog box, specify a name for the GROUP and select a color.
6. From the list of Available Instances, select those that you want included in the GROUP.
   - Use a string and wildcards (asterisk or question mark) in the Filter text box to narrow the list.
   - Use the **Ctrl** or **Shift** key to select multiple instances from the full or filtered list.
7. Click the **>** button to move the selected instances to the Selected Instances list. Click the **>>** button to add all instances to the Selected Instances list.

   You can further modify the group membership by moving elements between the Available Instances and Selected Instances lists.
8. Click **Add**.

The new GROUP is displayed in Floorplan View with a bounding box in the color you selected. The new GROUP is added to the Group constraint sheet in Spreadsheet View.

**To create a GROUP from selected placed components in Floorplan View:**

1. In Floorplan View, zoom in to the desired area as needed.
2. Select the placed components that you want to include in the GROUP. These can include PFU or PFF slices, EBR blocks, or DSP blocks.

   **Note**

   To view the precise logical identifiers for a PFU or PFF slice, select it on the layout and cross-probe to Netlist View.

3. Right-click the selected components and choose **Create GROUP**.

   The Create New GROUP dialog box opens. In the Selected Instances list, the placed components from the blocks you selected in Floorplan View are shown.
4. In the boxes at the top of the dialog box, specify a name for the GROUP and select a color.
5. To select a different anchor location, select either **Region** or **Site**.
   - For Region, select a REGION from the pulldown list. Regions must already be defined in order for them to appear on the list.

     When Region is selected, the group will float inside the selected REGION.
   - For Site, specify the column and row for the anchor. The allowable range is shown in parentheses above each box.

     When Site is selected, the specified site will serve as the northwest corner anchor location.
6. To define a different size for the group's bounding box, specify the number of columns and rows that the GROUP's bounding box should cover. The allowable range for height and width is shown in parentheses above each box. Make sure that the bounding box does not overlap the bounding box of another GROUP or REGION.

   The group's revised resource coverage is now displayed at the top under GROUP Name, including the number of LUTs, registers, and EBRs.
7. Click **Add**.

   The new GROUP is displayed in Floorplan View with a bounding box in the color you selected, and it is added to the Group constraint sheet in Spreadsheet View.

## Grouping Components Along an Individual Signal

In Floorplan View, you can group components along an individual signal of a timing path by cross-probing a signal from Timing Analysis View to Floorplan View. From Floorplan View, you can then select the components along the displayed connection to create a new GROUP.

**To group components along an individual signal:**

1. After placement and routing, choose **Tools > Timing Analysis View**.
2. In the timing constraint list in the lower left pane, expand Analysis Results, and then select the desired timing constraint.

   The path table opens in the upper right pane.
3. Select a signal from the path table, right-click, and choose **Show in > Floorplan View**.

   The connection is highlighted in Floorplan View.
4. While holding the **Ctrl** key, click each component related to the net.
5. Right-click the selected components and choose **Create > GROUP**.

   The Create New GROUP dialog box displays the anchor location and bounding box size of the components you selected, and lists the instances in the Selected Instances pane.
6. In the top part of the dialog box, assign a name to the GROUP. Click the color box to assign a different color.
7. Click **Add**.

   The new GROUP is displayed in Floorplan View with a bounding box in the color you selected. The new GROUP is added to the Group constraint sheet in Spreadsheet View.

**See Also** ▶"Using Timing Analysis View" on page 212

## Modifying GROUPs

Use the Edit GROUP Property dialog box to quickly modify the name, color, and membership of a group constraint. You can access the dialog box from the Floorplan View. You can also use Floorplan View to delete a GROUP, or you can use the Group constraint sheet to delete a GROUP or GROUP elements.

**To modify a group constraint:**

1. To access the Edit GROUP Property dialog box, do one of the following:
   - In the Group sheet of Spreadsheet View, double-click the group you want to modify or right-click the group and choose **Modify**.
   - In Floorplan View, zoom in to the GROUP and click the border or an empty space inside so that the group's border is highlighted. Right-click and choose **Edit GROUP**.
2. In the dialog box, make the desired changes and click **Update**.

**To remove elements from a group on the Group constraint sheet:**

1. Select the Group constraint sheet and expand the desired GROUP.
2. Select the group elements you want to remove, right-click, and choose **Remove GROUP Elements**.

**To delete a group constraint on the Group sheet:**

- Right-click the group you want to delete and choose **Delete**, or press **Delete**.

**To delete a GROUP using Floorplan View:**

1. Zoom in to the GROUP and click the border or an empty space inside so that the group's border is highlighted.
2. Right-click and choose **Remove GROUP**.

**See Also** ▶"Creating GROUPs" on page 154

## Resizing or Moving a GROUP in Floorplan View

You can resize a GROUP or change its anchor location in Floorplan View.

**To resize a GROUP using Floorplan View:**

1. In Floorplan View, zoom in to the GROUP and select it by clicking the border or an empty space inside. The border becomes highlighted, and you should see small circles on the GROUP's border—one in each corner and one on each side.
2. Position your mouse pointer over one of the corner or side circles.

   The cursor symbol changes to a double-headed arrow.
3. Drag the border outward or inward until the bounding box contains the sites you want to include.

   Floorplan View displays the new bounding box dimensions. The Group sheet in Spreadsheet View shows the updated dimensions.

**To change the GROUP's anchor location:**

1. Zoom out until you can view the current location on the layout and the targeted location.
2. Click the GROUP's border or an empty space inside it so that the border is highlighted.

   The crossed arrows symbol should immediately appear. If it does not, move your mouse pointer very slightly until it does.
3. Hold down the left mouse button, drag the GROUP box to the targeted location, and then release the mouse button.

   Floorplan View displays the GROUP in its new location. The Group sheet in Spreadsheet View shows the updated location.

**See Also** ▶"Creating GROUPs" on page 154

# Checking Design Rules

The Radiant software provides DRC, which enables you to validate I/O placement, logic, and routing.

During load time and initialization, the Radiant software constraints views interpret the database and constraint files in a similar manner as the Map and Place & Route processes. Syntax or semantic violations within the ..ldc and ..pdc files are also treated in a manner similar to the Map and Place & Route processes. Illegal constraints are ignored but retained in the logical and physical constraint files. The Radiant software loads and applies the legal subset it detects and displays error and warning messages.

After the project is loaded, the Radiant software provides access to on-demand and real-time Constraint DRC Check. All DRC reports are presented in the Output window for the current session.

**See Also** ▶"Constraint DRC" on page 159

## Constraint DRC

Constraint DRC can catch many common errors and prevent wrong pin usage at the early stage of the design process, and helps you ensure that design I/Os are assigned correctly before the design is mapped, placed, and routed. This can dramatically reduce design errors, help you meet timing requirements, optimize for both the FPGA and printed circuit design, and reduce the number of design iterations.

Constraint DRC performs logical and physical tests to validate I/O placement such as signal standard and VCCIO combinations bank by bank, true and emulated LVDS combinations, VREF requirements, and potential conflicts with user assignments made to multi-function pins like those of the sysCONFIG interface. It is available as real-time and on-demand DRC.

Constraint DRC now displays all of its messages in the Radiant software Output window.

**See Also** ▶"Performing On-Demand Constraint Design Rule Check" on page 159

## Performing On-Demand Constraint Design Rule Check

The Constraint DRC feature validates pin assignments to banks based on legal combinations of signal standards, including true and emulated LVDS combinations. The Radiant software performs immediate real-time Constraint

DRC on any constraint changes. But you can run Constraint DRC on the entire design after modifying constraints, by running on-demand Constraint DRC.

**To perform Constraint design rule checking:**

1. From the Radiant software Tools menu, select **Device Constraint Editor.** The following views are available:
   - Spreadsheet View
   - Package View
   - Device View
2. Choose **Design >**  **Constraint DRC**.

   The Radiant software checks the I/Os. A message box reports whether errors or warnings were encountered. A log of errors and warnings appears in the Output window.

**See Also** ▶"Constraint DRC" on page 159

- "Modifying Assignments in Device Constraint Editor Spreadsheet View" on page 151

# Post-Map DRC Check

In the post-map stage, DRC performs the following logical and physical tests:

- Block
- Net
- Pad
- Clock Buffer
- Name
- Primitive Pin

Reported errors from DRC indicate conditions where routing or component logic does not operate correctly. Warnings indicate conditions where routing or logic is incomplete or the condition is incorrect but not serious. Some conditions listed as warnings might be reported as errors when DRC is performed by the Generate Bitstream Data (bitgen) process.

Post-Map DRC Check is available in the post-map or post-PAR stage in the Spreadsheet View, Package View, Device View in Device Constraint Editor and Floorplan View.

**See Also** ▶"Performing On-Demand Constraint Design Rule Check" on page 159

# Implementing the Design

Design implementation includes the processes of:

- Synthesizing and translating the design to build the internal database.
- Mapping the design to device-specific components.
- Assigning the mapped components to specific locations.
- Establishing physical connections that will join the components in an electrical network.

The Radiant software environment enables you to set options for each of these processes to help optimize results, including placement effort level and guided placement and routing, among others. These options are available in the active strategy for the design. For descriptions of all design implementation process options, see the “Strategy Reference Guide” on page 380.

**See Also** “Using Strategies” on page 21

- “Synthesizing the Design” on page 161
- “Mapping” on page 168
- “Place and Route” on page 175
- “Applying Design Constraints” on page 115
- “Command Line Reference Guide” on page 561

## Synthesizing the Design

Synthesis is the process of translating a register-transfer-level design into a process-specific, gate-level netlist that is optimized for Lattice Semiconductor FPGAs.

In different ways, Radiant software can be used with almost any synthesis tool. The Radiant software comes with two tools fully integrated: Synopsys Synplify Pro for Lattice and Lattice Synthesis Engine (LSE). “Fully integrated” means that you can set options and run synthesis entirely from within the Radiant software. If you prefer, you can use other synthesis tools by running them independently of the Radiant software.

Synthesis tools can be run three ways with the Radiant software:

- Integrated synthesis is the simplest way. You work entirely within the Radiant software. See “Integrated Synthesis” on page 166.
- Interactive synthesis provides much greater control of how synthesis is done. You set up a design project in the Radiant software, but set up and run synthesis directly in the synthesis tool. See “Interactive Synthesis” on page 167.
- Stand-alone synthesis allows use of other, non-supported synthesis tools. You set up and run synthesis directly in the synthesis tool and then the Radiant software imports the result. See “Stand-Alone Synthesis” on page 167.

LSE can only be run in integrated synthesis.

Whichever method you use, see “Pre-Synthesis Check List” on page 162 to make sure your project is ready to synthesize.

**See Also**

- “Selecting a Synthesis Tool” on page 48
- TN1008, “HDL Synthesis Guidelines for Lattice Semiconductor FPGAs” for how coding style influences performance and area utilization
- Synopsys Synplify Pro for Lattice User Guide
- Synopsys Synplify Pro for Lattice Reference Manual

# Pre-Synthesis Check List

Following is a list of tasks to be done before running synthesis. Most of these are normally done as parts of other tasks, such as setting up the project and design entry.

In all cases, add or adjust the constraints, attributes, and directives used by the synthesis tool in the source code. See the synthesis tool’s user documentation. For LSE, see “Lattice Synthesis Engine Constraints” on page 416.

Integrated synthesis:

- Specify a synthesis tool. See “Selecting a Synthesis Tool” on page 48.

- Specify the top-level unit. See "Setting the Top-Level Unit for Your Project" on page 16. This is not always required but is a good practice. If not specified, you are relying on the defaults of the synthesis tool.

  **Note**

  In VHDL and mixed-language designs, LSE stops with an error if the top-level unit is not specified. LSE can find the top-level unit in pure Verilog designs.

- Specify a search path for files referenced by Verilog include directives. See "Specifying Search Path for Verilog Include Files" on page 50.
- Specify a VHDL library name. The default is "work." See "Specifying VHDL Library Name" on page 49.
- Order the source files for synthesis. The synthesis tool processes the files in the order showing in the File List frame.
- Specify the strategy settings for the synthesis tool. See "Using Strategies" on page 21. If using LSE, also see "Optimizing LSE for Area and Speed" on page 164.

Interactive and stand-alone synthesis:

- Reference the Lattice synthesis header library in the source code. See "Lattice Synthesis Header Libraries" on page 163.
- For interactive synthesis, if you want to use the Synplify Pro strategy settings instead of Synplify Pro's own defaults, change the "Export Radiant Settings to Synplify Pro GUI" option to **Yes** or **Only on First Launch**. See "Export Radiant Software Settings to Synplify Pro GUI" on page 385.

# Lattice Synthesis Header Libraries

The synthesis header libraries define primitives from the FPGA Libraries. A separate library is available for each device family and in Verilog and VHDL versions. The header libraries support Synplify synthesis tools. If the design has any primitives from the FPGA Libraries instantiated, the appropriate header library is required.

The integrated flow automatically includes the FPGA libraries, but if you are running synthesis outside of the Radiant software with interactive or stand-alone synthesis, you need to manually add a reference to the appropriate library. Look in Table 10 for the device family that you are using and add it to the source file list of your synthesis project. If your design is VHDL-based, add the .vhd file. If your design if Verilog-based, add the .v file.

The files are located at:

<*install_dir*>/cae_library/synthesis/verilog
<*install_dir*>/cae_library/synthesis/vhdl

If your design is in VHDL, also specify the library name for the header file.

**Table 10: Synthesis Header Files**

| Device Family | Header Library File | Library Name |
|---|---|---|
| iCE40 UltraPlus | ice40tp.v<br>ice40tp.vhd | ice40tp |

# Optimizing LSE for Area and Speed

The following strategy settings for Lattice Synthesis Engine (LSE) can help reduce the amount of FPGA resources that your design requires or increase the speed with which it runs. (For other synthesis tools, see those tools' documentation.) Use these methods along with other generic coding methods to optimize your design. Also, consider using the predefined Area or Timing strategies.

Minimizing area often produces larger delays, making it more difficult to meet timing requirements. Maximizing frequency often produces larger designs, making it more difficult to meet area requirements. Either goal, pushed to an extreme, may cause the place and route process to run longer or not complete routing.

To control the global performance of LSE, modify the strategy settings by choosing **Project > Active Strategy > LSE Settings**. In the Strategy dialog box, set the following options, which are found in **Synthesize Design > LSE**. See the following text for explanations and more details.

**Table 11: LSE Strategy Settings for Area and Speed**

| Option | Area | Speed |
|---|---|---|
| FSM Encoding Style | Binary or Gray | One-Hot |
| Max Fanout Limit | <*maximum*> | <*minimum*> |
| Optimization Goal | Area | Timing |
| Remove Duplicate Registers | True | False |
| Resource Sharing | True | False |
| Target Frequency | <*minimum*> | |
| Use IO Registers | Auto or True | Auto or False |

**FSM Encoding Style** If your design includes large finite state machines, the Binary or Gray style may use fewer resources than One-Hot. Which one is best depends on the design. One-Hot is usually the fastest style. However, if the finite state machine is followed by a large output decoder, the Gray style may be faster.

**Max Fanout Limit** A larger fanout limit means less duplicated logic and fewer buffers. A lower fanout limit may reduce delays. The default is 1000,

which is essentially unlimited fanout. Select a balanced fanout constraint. A large constraint creates nets with large fanouts, and a low fanout constraint results in replicated logic. You can use this in conjunction with the syn_replicate attribute. See “syn_replicate” on page 462. To minimize area, don’t lower this value any more than needed to meet other requirements. To minimize speed, try much lower values, such as 50.

You can change the fanout limit for portions of the design by using the syn_maxfan attribute. See “syn_maxfan” on page 451. Set Max Fanout Limit to meet your most demanding requirement. Then add syn_maxfan to help other requirements.

**Optimization Goal** If set to Area, LSE will choose smaller design forms over faster whenever possible.

If set to Timing, LSE will choose faster design forms over smaller whenever possible. If a create_clock constraint is available in an .ldc file, LSE ignores the Target Frequency setting and uses the value from the create_clock constraint instead.

If you are having trouble meeting one requirement (area or speed) while optimizing for the other, try setting this option to Balanced.

**Remove Duplicate Registers** Removing duplicate registers reduces area, but keeping duplicate registers may reduce delays.

**Resource Sharing** If set to True (box checked), LSE will share arithmetic components such as adders, multipliers, and counters whenever possible.

If the critical path includes such resources, turning this option off may reduce delays. However, it may also increase delays elsewhere, possibly reducing the overall frequency.

**Target Frequency (MHz)** A lower frequency target means LSE can focus more on area. A higher frequency target may force LSE to increase area. Try setting this value to about 10% higher than your minimum requirement. If Optimization Goal is set to Timing and a create_clock constraint is available in an .ldc file, LSE will use the value from the create_clock constraint instead.

**Use IO Registers** If set to True, LSE will pack all input and output registers into I/O pad cells. Register packing reduces area but adds delays.

Auto, the default setting, enables this register packing if Optimization Goal is set to Area. If Optimization Goal is Timing or Balanced, Auto disables register packing.

You can also control packing on individual registers. See “syn_useioff” on page 477. Set Use IO Registers to meet your most demanding requirement. Then add syn_useioff to help other requirements.

**See Also**

- “Using Strategies” on page 21
- “LSE Options” on page 388

# Integrated Synthesis

In integrated synthesis, you create, synthesize, and implement a design completely within the Radiant software environment. This is the simplest method, but it limits your control of synthesis to tools and features directly supported by the Radiant software.

### To synthesize your design in the Radiant software:

If you haven't already, check the Pre-Synthesis Check List before running synthesis.

1. If desired, change the strategy settings by choosing **Project > Active Strategy >** <*Synthesis Tool*> **Settings**. See also "Using Strategies" on page 21.

   **Note**

   If you are switching from interactive synthesis to integrated, you need to reset all the options in the Radiant software. Options set directly in the synthesis tool have no effect in the Radiant software.

2. In the Process Toolbar, click **Synthesize Design**. Alternatively, right-click **Synthesize Design** and choose **Run** from the pop-up menu.

   If Synthesize Design has already been run and does not recognize any changes in the design, it won't run again. You can force Synthesize Design to run by right-clicking it and choosing **Rerun** from the pop-up menu.

3. When finished, check the icon next to Synthesize Design in the Process frame. A green check mark indicates success; a red X indicates failure.

4. For more information on how the synthesis ran, open the **Reports** view in the Tools area (choose **View > Reports**). Under Project Summary, choose **Synthesis Reports >** <*Synthesis Tool*>. This report lists actions taken by the synthesis tool, warnings, and errors. The report also provides a list of FPGA resources used and a timing report based on estimated place-and-route data. See "Viewing Logs and Reports" on page 54.

   You can find a more detailed area report in the folder of the active implementation. The report is named with an.areasrr extension (for Synplify Pro) or with an .arearep extension (for LSE). The report includes the resources used by each module of the design. Similar information can also be found in the Hierarchy view.

   If you are using LSE, you can also find a more detailed timing report in the folder of the active implementation. The report is named <*top_module*>_lse.twr. It is similar to the report described in "Timing Analysis Report File" on page 198.

### See Also

- "Optimizing LSE for Area and Speed" on page 164
- "Running Processes" on page 42

# Interactive Synthesis

In interactive synthesis, you set up a design project in the Radiant software, but set up and run synthesis directly in the synthesis tool. This gives you more complete control of synthesis than integrated synthesis and allows you to make full use of the synthesis tool's features. Note that LSE cannot be used in interactive synthesis.

**Note**

If you decide to use interactive synthesis, remember to use it every time you make a design change. If you try to repeat the implementation process by simply double-clicking an item in the Process frame, the Radiant software will synthesize the design with integrated synthesis, which may give unexpected and inferior results.

**To synthesize your design using interactive synthesis:**

If you haven't already, check the "Pre-Synthesis Check List" on page 162 before running synthesis.

1. Open the synthesis tool from the **Tools** menu.

   The synthesis tool opens with the device and source files for your design.
2. Set options in the tool as desired.
3. To re-use the settings, save the synthesis tool's project file as <*project_name*>_**syn.prj** (Synplify) or <*project_name*>**.psp** (Precision) in the project folder.

   Make sure the Radiant software strategy option, "Export Radiant Settings to Synplify Pro GUI" (under Synplify Pro in the Strategies dialog box) is set to **No** or **Only on First Launch**. See "Export Radiant Software Settings to Synplify Pro GUI" on page 385.
4. Synthesize the design.
5. Analyze the results in the synthesis tool.

# Stand-Alone Synthesis

In stand-alone synthesis, you can use a synthesis tool that is not directly supported by the Radiant software. In this flow you set up and run synthesis directly in the synthesis tool. Then create a design project in the Radiant software using the output of the synthesis tool as the source file. Note that LSE cannot be used in stand-alone synthesis.

If you haven't already, check the "Pre-Synthesis Check List" on page 162 before running synthesis.

**To set up a project using a stand-alone tool:**

1. Open the synthesis tool directly without using the Radiant software.
2. Using your synthesis tool, create a project, including specifying a device, source files, and options.

3. Synthesize the design.
4. Analyze the results in the synthesis tool.
5. When you are satisfied with the synthesis, create a design project in the Radiant software.

**To re-synthesize your design using the stand-alone tool:**

This procedure assumes that you have already set up the project to use stand-alone synthesis.

1. Open the synthesis tool directly without using the Radiant software.
2. In the synthesis tool, open the project.
3. If desired, change the options.
4. Analyze the results in the synthesis tool.

# Mapping

Mapping is the process of converting a design represented as a network of device-independent components, such as gates and flip-flops, into a network of device-specific components, such as PFUs and EBRs or configurable logic blocks.

You can use the Map Design process in the Radiant software environment or the MAP program from the command line.

The Map Design process generates physical descriptions of the logical configuration within the programmable device elements. These device elements include programmable function units (PFU), programmable I/O cells (PIC), embedded block RAM (EBR). They also include special function blocks: internal oscillator, global set/reset (GSRN), start-up logic, phase locked loop (PLL), delay locked loop (DLL), and physical coding sublayer (PCS) logic.

Depending upon the attributes specified in the input netlist, mapping includes absolute placement, logical partitioning (hierarchical netlists), component group placement, and regional group placement information in the physical description.

**See Also** ▶"Running MAP from the Command Line" on page 577

- "Setting Map Design Options" on page 168
- "Mapping Output Files" on page 169

## Setting Map Design Options

When you run the Map Design process in the Radiant software, the design is mapped based on design options that are in the active strategy. Mapping options can influence the performance and utilization of the design

implementation, ease incremental design changes, or allow you to over map in order to review how many resources are required for a particular implementation.

**To set Map Design options:**

1. Choose **Project > Active Strategy > Map Design Settings** to open the Strategies dialog.
2. For each option that you want to change, double-click a cell in the Value column. Select the desired value from the pulldown menu, enter text, or click the browse button, as appropriate.
3. Click **OK**.

**See Also** "Map Design Options" on page 395

- "Mapping" on page 168
- "Mapping Output Files" on page 169

# Mapping Output Files

The following files are created from the Map Design process:

- MAP Report File (.mrp) — The map report file provides details about how the design was mapped to physical elements and reports any errors. Mapping details can include such things as how attributes were interpreted, logic that was removed or added, and how signals and symbols in the logical design were mapped to components in the physical design.

**See Also** "Mapping" on page 168

- "Setting Map Design Options" on page 168

# Running Map Design

In the Radiant software environment, the Map Design process automatically maps the design based on the active strategy settings.

**To run Map Design in the Radiant software environment:**

- In the Process Toolbar, click **Map Design**.

**See Also** "Running MAP from the Command Line" on page 577

- "Running Processes" on page 42

# MAP Report File

The MAP Report (.mrp) file supplies statistics about component usage in the mapped design and shows the number and percentages of resources used out of the total resources in the device. The report is produced whether you have run Map in the Radiant software environment or the command line.

**To view the MAP Report file in the Radiant software:**

- Click the **Reports** tab to activate the Report view and select **Map Reports.** The Map report is displayed in the Reports window to the right.

  OR

- Using a text editor, open the .mrp ASCII file in your project directory.

Although detailed information varies depending on the device, the format of the .mrp file is the same. The report is divided into sections, which can include the following:

- Design Information – Shows the map command line, the device/ performance grade, and the time/date stamp of the run.
- Design Summary – Shows the number and percentage of resources used out of the total of the resources of the mapped device. The MAP Report lists the number of slice registers, I/O registers, LUT4s, and other logic. The number of SLICEs generated from MAP could exceed the device limitation. The Place and Route (PAR) process will determine whether or

not it is necessary to merge two SLICEs into one.

**Note**

The total resources reported represents the total resource count in the device and not the total accessible resources. Since the accessibility of resources can change, depending on the design and design revisions, there will sometimes be a gap between the number of resources that are available for use and the total number in the device.

```
Design Summary
--------------
   Number of slice registers: 2140 out of  5280 (41%)
   Number of I/O registers:     0 out of    56 (0%)
   Number of LUT4s:           3833 out of  5280 (73%)
      Number of logic LUT4s:             3026
      Number of inserted feedthru LUT4s:  39
      Number of ripple logic:            384 (768 LUT4s)
   Number of IO sites used:   22 out of 56 (39%)
      Number of IO sites used for general PIOs: 22
      Number of IO sites used for I3Cs: 0 out of 2 (0%)
      Number of IO sites used for PIOs+I3Cs: 22 out of 53 (42%)
      (note: If I3C is not used, its site can be used as
general PIO)
      Number of IO sites used for OD+RGB IO buffers: 0 out of 3
(0%)
   Number of DSPs:             4 out of 8 (50%)
   Number of I2Cs:             0 out of 2 (0%)
   Number of High Speed OSCs:  0 out of 1 (0%)
   Number of Low Speed OSCs:   0 out of 1 (0%)
   Number of RGB PWM:          0 out of 1 (0%)
   Number of RGB Drivers:      0 out of 1 (0%)
   Number of SCL FILTERs:      0 out of 2 (0%)
   Number of SRAMs:            0 out of 4 (0%)
   Number of WARMBOOTs:        0 out of 1 (0%)
   Number of SPIs:             0 out of 2 (0%)
   Number of EBRs:             0 out of 30 (0%)
   Number of PLLs:             0 out of 1 (0%)
   Number of Clocks:  1
      Net clk_c: 2022 loads, 2022 rising, 0 falling (Driver:
Port clk)
   Number of Clock Enables:  8
      Net n18836: 38 loads, 38 SLICEs
      Net n9306: 128 loads, 128 SLICEs
      Net n18680: 36 loads, 36 SLICEs
      Net n18773: 36 loads, 36 SLICEs
      Net n18139: 36 loads, 36 SLICEs
      Net n9301: 128 loads, 128 SLICEs
      Net n9296: 128 loads, 128 SLICEs
      Net n9311: 128 loads, 128 SLICEs
   Number of LSRs:  6
      Net maxfan_replicated_net_203: 169 loads, 169 SLICEs
      Net data_in_reg_63__N_501: 925 loads, 925 SLICEs
      Net mult32_2/n1818: 4 loads, 4 SLICEs
      Net n9303: 4 loads, 4 SLICEs
      Net n9298: 4 loads, 4 SLICEs
      Net mult32_3/n1821: 4 loads, 4 SLICEs
   Top 10 highest fanout non-clock nets:
      Net reset_n_c: 926 loads
      Net data_in_reg_63__N_501: 925 loads
```

The Number of SLICEs information in the Design Summary will vary.

The number and percentage of PIO sites used, out of the total number, is given in the Design Summary. Sometimes this is further broken down into

external and differential PIOs. The Number of external PIOs is based on the number of PIO pads programmed with a single-ended signal standard out of the total number of bonded pads on the chip die. The Number of differential PIOs counts those PIOs programmed with a differential signal standard.

- Design Errors/Warnings:
  - Shows any warnings or errors encountered by the mapping process. For example, the section will report a pad that is not connected to any logic, or a bidirectional pad that has signals passing only in one direction.
  - Shows any warnings or errors generated as a result of the design rule tests performed at the beginning of the mapping process. These warnings and errors do not depend on the device to which you are mapping.
  - Shows errors and warnings associated with problems in assigning components to sites. For example, an attribute on a component specifies that the component must be assigned to site AK, but there is no site AK on the part to which you are mapping. Another example would be a problem in fitting the design to the part where the design maps to 224 logic blocks, but the part only contains 192 logic block sites.
- IO (PIO) Attributes – Provides a table by input/output port of programmed direction, I/O type, and whether the PIO is registered. Shows any attributes (properties) specified on PIO logic elements.
- Removed Logic – Describes any logic that was removed when the design was mapped. Logic may be removed for the following reasons:
  - A design uses only part of the logic in a library macro.
  - The design has been mapped even though it is not yet complete.
  - The mapping process has optimized the design logic.
  - Unused logic has been created in error during design entry.

  This section also indicates which nets were merged when a component separating them was removed.

  The Removed Logic section also enumerates how many blocks and signals were removed from the design, including the following kinds of removed logic:
  - Blocks clipped – A "clipped" block is removed because it is along a path that has no driver or load. Clipping is recursive; that is, if Block A becomes unnecessary because logic to which it is connected has been clipped, then Block A is also clipped.
  - Blocks removed – A removed block is removed because it can be eliminated without changing the operation of the design. Removal is recursive; that is, if Block A becomes unnecessary because logic to which it is connected has been removed, then Block A is also removed.
  - Blocks optimized – An "optimized" block is removed because its output remains constant regardless of the state of the inputs—for

example, an AND gate with one input tied to ground. Logic generating an input to this optimized block (and to no other blocks) is also removed, and appears in this section.

- Signals removed – Signals were removed because they were attached only to removed blocks.
- Signals merged – Signals were combined because a component separating them was removed.

- ASIC Components – Lists any ASIC instances in the design by name and specifies the type used for each. These components mostly include IPs, such as Block RAM, DSP, and others.
- Symbol Cross Reference – Shows where symbols in the logical design were mapped in the physical design. By default, Map will not report symbol cross references unless you specify the -xref_sym Map command line option or enable the Report Symbol Cross Reference property for the Map Design process in the Radiant software environment.
- Signal Cross Reference – Shows where nets in the logical design were mapped in the physical design. By default, Map will not report symbol cross references unless you specify the -xref_sig Map command line option or enable the Report Signal Cross Reference property for the Map Design process in the Radiant software environment.
- PLL/DLL Summary – Provides instance name, type, clock frequency, pin/node values, and a listing of all settings for each PLL or DLL in the design.
- OSC Summary – Provides the oscillator's instance name, possible oscillator primitive type, output clock, and nominal frequency. This section is only included in the map report if the OSC primitive has been instantiated in the design.
- Run Time and Memory Usage – Lists total CPU time, real time, and peak memory usage related to the Map run.

# PFU Logic Mapping with HGROUP/UGROUP

When an HGROUP/UGROUP is defined, Map will prevent logical elements of different HGROUPs/UGROUPs to be mapped into the same Programmable Function Unit (PFU). By doing so, a natural partitioning of the mapped design is achieved.

Map then groups all PFUs of the same HGROUP and writes out PGROUP. The HULOC attribute translates to the LOCATE PGROUP. The LOC attribute translates to the relative location within a PGROUP.

## Packing of Duplicate Registers

Designers often duplicate registers in order to reduce fan-out and improve timing. Duplicated registers are registers with the same data and control paths. The software keeps duplicated registers from being mapped into the same PFU.

If the COMP= attribute is attached to duplicate registers, it will override duplicate register packing.

# Place and Route

After a design has undergone the necessary translation to bring it into the physical design format during mapping, it is ready for placement and routing. Placement is the process of assigning the device-specific components produced by the mapping process to specific locations on the device floorplan. After placement is complete, the route phase establishes physical connections to join components in an electrical network. The place and route process takes a mapped physical design and places and routes the design. Placement and routing of a design can be cost-based or timing driven.

In most cases, your design will require timing-driven placement and routing, where the timing criteria you specify influences the implementation of the design. Static timing analysis results will show how constrained nets meet or do not meet your timing.

In the Radiant software Task Detail View, there is a Place & Route Trace process available that runs static timing analysis. This process reports any timing errors and generates a report. The Place & Route Trace report can be accessed from the Place & Route Reports folder of the Reports window.

**See Also**

- "Setting Place & Route Design Options" on page 175
- "Options and Processes to Improve PAR Results" on page 176
- "Place & Route Output Files" on page 176
- "Running Place & Route Design" on page 177
- "PAR Report File" on page 180


## Setting Place & Route Design Options

Placement and routing options can influence the performance and utilization of the design implementation and ease incremental design changes. Some options, such as "Placement Sort Best Run," affect the way the results are reported. You can set Place & Route Design options in the Strategies dialog box in the Radiant software or use the par command-line program.

**To set Place & Route Design options in the Radiant software environment:**

1. Choose **Project > Active Strategy > Place and Route Design Settings** to open the Strategies dialog.
2. For each option that you want to change, double-click a cell in the Value column.
3. Select the desired value from the pulldown menu, enter text, or click the browse button, as appropriate.
4. Click **OK**.

**See Also** "Place & Route Design Options" on page 397

- "Running PAR from the Command Line" on page 579
- "PAR Report File" on page 180

# Options and Processes to Improve PAR Results

Experimenting with place and route settings in the Strategies dialog box can help improve your placement and routing results. Try changing certain strategy options to help meet your objectives, as in the following scenarios:

- Some number of connections are left unrouted.

  Increase the number of routing passes to help produce a fully routed design.

- Timing period/frequency objectives are not met.

  Try one or more of the following techniques:

  - Increase the number of routing passes.
  - Increase the effort level of the placer.
  - Increase the number of times the placer runs.

**See Also** "Running Place & Route Design" on page 177

- "Running PAR from the Command Line" on page 579
- "Place & Route Design Options" on page 397
- "Cost-Based Place & Route" on page 185
- "Place & Route Output Files" on page 176
- "Place & Route Output Files" on page 176

# Place & Route Output Files

The following files are the output files generated by the Place & Route Design process:

- PAR Report File (.par) — a PAR report including summary information of all placement and routing iterations.
- PAD Specification File (.pad) — a report file containing I/O pin assignments.
- Delay Report File (.dly) — an optional ASCII text file that includes a delay analysis of the 20 nets with the longest delay. PAR generates this file when the “Create Delay Statistic File” has been set to **True** in the Strategies dialog box or when the -y option is used in the command line.

  The option also appends a Delay Summary Report to the end of the Place & Route report. The first column of this summary gives the actual averages for the design. The figures in the second column, which are enclosed by parentheses, indicate a “worst case” scenario for the delay.

**See Also** “Place and Route” on page 175

# Running Place & Route Design

In the Radiant software environment, the Place & Route Design process automatically assigns device-specific components to locations and connects them.

**To run Place & Route Design in the Radiant software environment:**

- In the Process Toolbar, click **Place & Route Design**.
- Alternatively, right-click **Place & Route Design** and choose **Run PAR** from the pop-up menu. You can force run PAR from the pop-up menu as well.

**See Also** “Setting Place & Route Design Options” on page 175

- “Running PAR from the Command Line” on page 579
- “Running Processes” on page 42

# Multiple PAR Iterations

The “Place Iterations” option for Place & Route Design sets the maximum number of placement/routing passes.

If you specify options that produce a single output design file, your output consists of a single par file and .pad file. Optionally, you can output a Delay Statistics (.dly) file that will also, by default, assume the project name. See “Delay Statistics Report File” on page 184 for details.

If you run multiple placement and routing iterations, you produce a .par, .dly, and .pad file for each iteration. After each iteration, the program saves the physical design for every instance where the timing core has improved. This means that during long PAR runs, the current minimal timing score is always available. This behavior prevents a few rare cases where timing scores may have gone up rather than down when performing multiple PAR iterations. The

timing score is determined as a weighted sum of several parameters including the number of unrouted nets, number of timing constraints not met, amount by which the timing constraints were not met, maximum delay on nets, and maximum delay on the ten highest nets.

As the par command is performed, PAR records one .par file (a summary of all placement and routing iterations) at the same level as the directory you specified. It also places within the directory a .par, .dly and .pad file for each individual iteration.

The file names for the output files use the naming convention <*effort-level*>_<*cost-table-entry*>; for example, 1_1.par, 1_1.dly, and 1_1.pad. In this example, the effort level and cost table entries start at 1 (the default effort level is 5).

**Note**

If you wish to further optimize your place-and-route results beyond multiple PAR iterations, a process which uses timing as the sole criteria for optimization, see "Using MPARTRCE to Optimize PAR Iterations" on page 178. In addition to timing score, the MPARTRCE tool's methodology uses a performance score when it determines the best design run for your purposes.

**See Also** ▶"Place and Route" on page 175

▶ "Using MPARTRCE to Optimize PAR Iterations" on page 178

## Using MPARTRCE to Optimize PAR Iterations

MPARTRCE is a multi-PAR program that uses a performance score for selecting the best iteration. It is available from the command line.

**Multi-PAR** When PAR processes design iterations using different placement seeds to produce the best possible placed-and-routed design, it does so by performing and evaluating each design on a timing score basis. See "Multiple PAR Iterations" on page 177.

The timing score is a useful measure for algorithms to perform optimal place-and-route on the design.

In multiple PAR scenarios, you are interested in the implementation that results in the best timing performance. For example, in a single clock design, where the setup and clock-to-out times are not critical, the best iteration is the one that results in the highest $f_{MAX}$ in static timing analysis.

**MPARTRACE Multi-PAR** The multiple PAR of MPARTRCE continues to use the timing score for PAR algorithms in multiple PAR iterations, but it also employs a measure called performance score for selecting the best iteration.

**MPARTRCE attempts to do the following:**

- Use the existing method for running multiple PAR iterations, each using a different placement seed and using “timing score” criteria for optimizing place and route processes.
- Automatically run static timing analysis on the implementation after each PAR iteration.
- Establish a new selection criterion to pick the iteration that best satisfies user requirements.
- Provide options to specify special requirements and criteria for selecting the best result.

**Performance Score** The MPARTRCE method for selection of the best design uses a performance score in its determination. The performance score is calculated for each iteration’s implementation result and the one with the highest (algebraic maximum) performance score is selected as the best result.

**MPARTRCE Process** MPARTRCE executes the following operations:

1. Reads in the constraints and determines the weights and penalties based upon a specific .ini file and user overrides.
2. Runs PAR and result selection algorithms multiple times. During result selection, it parses the report for the values of interest, determines the frequency, and determines performance scores.
3. After all the iterations, copies the implementation and report files corresponding to the best result from the results directory into the project directory and creates the log file.

If you break the process after any iteration N, the results and reports correspond to the best implementation to that point. The outputs would be equivalent to what you would have obtained if you had run multiple PAR to N iterations.

**See Also** ▶“Running Timing from the Command Line” on page 584

- “Multiple PAR Iterations” on page 177

# Place & Route Report Files

The Reports view in the Radiant software provides immediate access to reports produced by the Place & Route Design process, including the Place & Route report (.par) and the Pad Specification file (.pad). These reports open in the Report view in HTML format.

If you have enabled the Create Delay Statistics File option for the active strategy, a Delay Statistics Report File (.dly) is included in your project directory. When you run multiple iterations of Place & Route, a .par file, .pad file, and .dly file are included in your project directory for each run. You can open these files with a text editor.

**See Also** "Multiple PAR Iterations" on page 177

"PAR Report File" on page 180

## PAR Report File

The PAR Report (.par) file contains execution information and shows the steps taken as the process converges on a placement and routing solution.

**To view the PAR Report file in the Radiant software:**

- In the Design Summary pane of the Reports window, select **Place & Route Reports** and select one of the three reports:
  - Place & Route
  - Signal/Pad
  - Place & Route Timing Analysis

  The chosen report will appear in the right pane.
- Alternatively, use a text editor to open the .par ASCII file from your project directory.

By default, the Cost Table Summary is sorted by worst slack for multiple placement seeds, as follows:

1. Worst slack (setup)
2. Timing score (setup)
3. Worst slack (hold)
4. Timing score (hold)
5. Seed #

If you would prefer the cost table be sorted by timing score for multiple placement seeds, select the Timing Score option for "Placement Sort Best Run" in the Place & Route Design section of the active strategy. The table will then be sorted as follows:

1. Timing score (setup)
2. Worst slack (setup)
3. Timing score (hold)
4. Worst slack (hold)
5. Seed #

**Device Utilization Summary** This section summarizes the number and percentage utilization of device resources, including I/O, logic, and global signals. For PIOs, twice the number of PIO SITE resources are required for differential I/Os. A single differential PIO COMP uses two PIO SITE resources, whereas non-differential PIO COMPs use one PIO SITE resource.

The PIO report summary counts the number of bonded and unbonded PIOs required to implement single-ended and differential signal standards. PIO utilization reflects the preliminary results reported by the design mapper (map). See the I/O Usage Summary section for details on final results. Utilization reported by Map might be lower, because some SITE resources, such as VREF assignments, are made automatically by PAR.

*Report Syntax*:

SITE TYPE <*# sites used*> / <**#** sites of type available on device> <x>% used

Consider the example below. The first column identifies the device type. The second column provides the actual number of available sites versus those used for that device type; and the third column provides the percentage of available sites used. For PIOs, you will see a breakdown of available versus used bonded pads and a percentage:

```
Device utilization summary:

   APIO              24/36           66% used
   GSR                1/1           100% used
   IOLOGIC          110/506          21% used
   PIO (prelim)     194/504          38% used
                    194/372          52% bonded
   DQSBUF             8/16           50% used
   EBR              135/225          60% used
   DQSDLL             1/2            50% used
   MULT18             1/88            1% used
   PCS                2/2           100% used
   SLICE          17827/23832        74% used

   PLL                3/8            37% used
```

For total PIO resources, a preliminary (prelim) report shows that out of a possible 504 total PIO resources on the device, the design uses 194 of them, or about 38 percent. The following line shows that 194 bonded out pads are being used out of a possible 372 bonded I/Os, or about 52 percent.

The term “used” means that these I/O are in the design and will be programmed on the device. This report is considered preliminary (prelim) because VREF assignments have not yet been placed. The difference in resource utilization is accounted for later in the PAR report in the I/O Usage Summary (final).

An I/O is considered “bonded” when the packaging of the chip connects the bond pad to a pin on the package.

**Note**

Bonded pad availability is based on packaging, which varies within a device family from part to part. See the Package Diagrams documentation in addition to the Product Selector Guide on the Lattice web site for details when selecting a device and package for your application.

The rest of the example report shows IOLOGIC, PLL, SLICE and other various resource type usage. As utilization percentage approaches 100 percent on logic or any specific key resource types reported in this section, you might want to consider a larger device that can accommodate better design performance.

**Placement** This section provides a log of messages produced during the placement phase.

**Clock Report** This section provides a complete listing of all of the clocks used in the design. Aside from the example, the Clock Summary report would reflect a scenario where only global clocking is used as follows:

```
   Quandrants All (TL, TR, BL, BR) - Global Clocks
      PRIMARY  : 1 out of 4 (25%)
        DCS    : 1 out of 2 (50%)
     SECONDARY : 0 out of 4 (0%)
```

**I/O Usage Summary (final)** This section Summarizes the final number and percentage utilization of device resources, including I/O, logic, and global signals.

*Report Syntax*:

```
I/O Bank Usage Summary (final):
   <#PIO sites used> out of <#PIO sites of device> (x%) PIO
   sites used.
<#PIO sites used> out of <#PIO sites of device/pkg> (x%) bonded
PIO sites
used.
Number of PIO comps:
Number of Vref pins used: <#VREF pins>
```

**Routing** This section provides a log of messages produced during the routing phase.

**Completion** This section shows statistics of the run, including time to route, number and percentage of successfully routed connections, errors and warnings, and final timing score.

Notes regarding the sample place and route report:

- Placer score is a rating of the relative “cost" of a placement. A lower score indicates a better, less costly placement.
- Timing score will always be 0 (zero) if all timing has been met. If not, the number will be other than 0.
- Underneath the “Completed router resource pre-assignment” line, warning messages might appear, indicating that a clock signal can suffer excessive delay or skew due to conflicts in clock routing resources.

  For example, the placer avoids placing two clock signals in the same PIC pair that share the same clock spine. If more than one clock signal are hard-located in the same PIC pair that share a common clock spine, the router will then write out a warning message in the PAR Report File (.par).
- The “Starting iterative routing” section contains figures in parentheses (125818). This represents the timing score for the design (not to be confused with the PAR score) at the end of the particular iteration. When the timing score reaches 0 (as it does in this example after iteration 2), all timing has been met. This timing score (0) also appears at the end of the Delay Summary Report section. Note that the Delay Summary Report is only generated when you enable the “Create Delay Statistic File” option in the Strategies dialog box, or use the **-y** option in the command line.

  The timing score at the end of the “Starting iterative routing” section might not agree with the timing score at the end of the Delay Summary Report. This can occur if MAXSKEW is scored and not met. The score shown in the Delay Summary Report section will always be the correct one.
- The design may be completely routed but the router continues to route in the attempt to adhere to timing.
- There is a section before the “Starting Constructive Placer” message that lists all selected primary clocks.
- When you specify the **-y** option on the command line, the last section of the .par file summarizes the delay information for the routed design. The first column of this section gives the actual averages for the design. The figures in the second column, which are enclosed by parentheses, indicate a "worst case" scenario for the delay. The PAR run also produces a .dly (delay) file that contains more detailed timing information.

**Note**

Timing scores can rise when attempting to meet timing constraints, because the router might move a signal to a less favorable path to make a resource available for another signal.

**See Also** ▶“Place & Route Output Files” on page 176

## PAD Specification File

The PAD specification (.pad) file is an ASCII report file that lists all Programmable I/O Cells (PICs) used in the design and their associated primary pins. The PICs are listed by port name and by pin names.

**To view the PAD Specification Report File in the Radiant software environment:**

- In the Design Summary pane of the Reports window, select **Signal/Pad**. Your report will appear in the pane to the right.
- Alternatively, use a text editor to open the .pad ASCII file from your project directory.

The .pad file may include the following information:

- Pinout by Port Name – This section lists the port names with primary pin designations. It also shows the buffer type and any associated attributes. Buffer type indicates the PIO mode. For example, a PIO in LVDS or LVPECL mode needs two bonded pads for differential signals that are both included in the .pad file.

  In most designs, the top-level HDL design will not include both true and complement sides of differential port signals. The designer assigns I/Os to pads (via LOCATE) using a single signal, assuming that it will represent the true (+) signal. Based on an IOBUF type of LVDS or LVPECL, the Radiant software will automatically infer the complement (-) signal and assign it to the appropriate site on the device package.

  Port signals are reported using the syntax: <port name>+ for positive differential and <port name>- for negative differential in the Pinout by Pin Number table.
- Vccio by Bank – This section lists the voltage by bank.
- Vref by Bank – This section shows the name and location of on-chip voltage references that have been set and the associated port groups or signals.
- Pinout by Pin Number – This section lists pin numbers, which includes the primary pin number, the component name or reference voltage type, and the buffer type. The Dual Function column identifies pins that can be used for I/O assignments or for another function, such as Vref. In the Pin info column, the status of any "unused" pin is reported as "unused, PULL:UP," because these pins have an internal pull-up.

## Delay Statistics Report File

When you use the -y option from the command line or the “Create Delay Statistics File” option in the Strategies dialog box, each PAR run will generate a delay file (.dly). This file contains delay information for each net in the design. It includes:

- A listing of the 20 nets with the longest delays.

  An "e" preceding a maximum delay indicates that the delay shown is only approximate.
- A delay analysis for each net, including the net name, followed by the driver pin and the load pin(s).

The “Create Delay File” or -y option also appends a Delay Summary Report to the end of the Place & Route report. The first column of this summary gives

the actual averages for the design. The figures in the second column, which are enclosed by parentheses, indicate a "worst case" scenario for the delay.

**See Also** "Running Place & Route Design" on page 177

- "Place & Route Design Options" on page 397
- "Cost-Based Place & Route" on page 185
- "Place and Route" on page 175
- "Place & Route Considerations" on page 186
- "Running PAR from the Command Line" on page 579

# Cost-Based Place & Route

The standard PAR package is a cost-based tool. This means that placement and routing are performed using various cost tables that assign weighted values to relevant factors such as constraints, length of connection, and available routing resources.

**Placement** The PAR process places the mapped physical design in two stages: a constructive placement and an optimizing placement. PAR writes the physical design after the completion of each of these two stages.

During constructive placement, PAR places components into sites based on factors such as the following:

- Constraints specified in the input file (for example, certain components must be in certain locations)
- Length of connections
- Available routing resources
- Cost tables that assign random weighted values to each of the relevant factors. There are 100 possible cost tables.

Constructive placement continues until all components are placed. Optimizing placement is a fine-tuning of the results of the constructive placement.

**Routing** Routing also is done in two stages: iterative routing and delay reduction routing, which is also called cleanup. PAR writes the physical design only after iterations where the routing score has improved.

During iterative routing, the router performs an iterative procedure to converge on a solution that routes the design to completion or minimizes the number of unrouted nets.

During reduction routing, the router takes the result of iterative routing and reroutes some connections to minimize the signal delays within the device. There are two types of reduction (cleanup) routing that you can perform:

- A faster cost-based cleanup routing. This type of routing makes decisions by assigning weighted values to factors, such as the type of routing resources used, that affect delay times between sources and loads.

- A more intensive delay-based cleanup routing. This type of routing makes decisions based on computed delay times between sources and loads on the routed nets.

**See Also** "Place and Route" on page 175

# Place & Route Considerations

In primary clock placement, if the primary clock is a PIO, the placer will automatically place it into a "sweet site," a site that can be routed easily to the center. However, if the primary clock is driven by a PFU (internally generated primary clock), the placer will attempt to place it on a sweet site or a proximal location. If you locate a primary clock driver on a non-sweet site, the placer will issue a warning. During pre-placement, all PIO constraints are observed, which means that the pre-placement is always legal.

- The placer prints out a list of selected primary clocks before the constructive placement. You should check that the primary clocks are selected properly.

# Running Multiple PAR Jobs in Parallel

The PAR Multi-tasking option allows you to use multiple machines (nodes) to run multiple place-and-route jobs at the same time instead of serially. You can run this feature in the Radiant software environment. Use the "Multi-tasking Node List" option in the Place and Route Settings of the Strategies dialog box, or use the PAR-m option from the command line.

The ability to run multiple jobs on different nodes simultaneously can significantly reduce the time it takes to complete these runs. Otherwise, it would take the cumulative time for each job to complete by itself in a consecutive manner, and this could mean the difference between one hour and 10 hours of total time.

The PAR multi-tasking option is supported three ways:

- Local multi-tasking, where multiple PAR runs are executed on a single machine.
- Networked multi-tasking, where multiple PAR runs are executed on multiple machines.
- A combination of local and networked.

**Creating a Node List File** For a PC or for UNIX/Linux, you must first create an ASCII-based node list file input that specifies the candidate machines that allow multiple PAR jobs to be run in parallel.

The node list file contains node description blocks of information on separate lines for each machine in the format shown below:

[<*node_name*>]

System = <*solaris* | *linux* | *pc*>

Corenum = <*number_of_cores*>

Env = <*file_name*>

Workdir = <*directory_name*>

[<*node_name2*>]

System = <*solaris* | *linux* | *pc*>

Corenum = <*number_of_cores*>

Env = <*file_name*>

Workdir = <*directory_name*>

*where*:

The [<*node_name*>] value contains the machine name in square brackets. For example, if your machine name is *jsmith*, this line would simply show [jsmith] on the first line.

**Note**

For UNIX/Linux, you can use many networked nodes to run the PAR job, so the node list will have a corresponding number of node description blocks in it. For PC, there will only be one node description block in the node list since you can only employ one PC that has multiple cores. This feature does not currently support a host of networked PCs.

The System node description line lists the platform the machine runs on. This line can only take Solaris, Linux, or PC as a value.

Corenum specifies how many CPU cores there are on the machine. This line can be omitted when there is only one CPU on the machine. Changing it to zero disables the node from running PAR tasks.

Env specifies an environment setup file to be run before the multiple PAR job is run on the remote machine. If the remote machine environment is ready, then this line can be omitted. To test if a remote machine is ready, run the following command to see if PAR can be run remotely:

ssh <*remote_machine*> <*par_cmd_options*>

Workdir specifies in which working directory on the remote machine the PAR multi-task job should be run. If account permissions allow you to get into that directory automatically after login, this line can also be omitted.

General Node List Rules:

- You must use the equal sign for each node description line and the same line and character spacing illustrated in the node list format example.

- Only the names of files and directories are case sensitive on UNIX and Linux. Node description line markers, for example, “Corenum =” can appear in upper, lower, or mixed case.
- The [<*node_name*>] and System lines are required for any node list file.
- The Corenum, Env, and Workdir node description lines are optional.
- For PC, the nodelist file should only include one node description block on the local machine. Remotely networked PCs are not supported. Any entries for remote nodes on a PC network will be discarded by the PAR multi-tasking feature.
- For UNIX /Linux, the nodelist file can have multiple machine nodes; but it should not include PC type nodes, which will be discarded.
- By beginning a line with double forward slashes, “//”, you can place a comment line anywhere in the node list file. The PAR multi-tasking feature will ignore its syntax.

**PAR Multi-Tasking Environment Setup** For PC, running PAR remotely is not supported, so multiple PAR processes can only run in parallel on one PC. Since the child PAR process takes the environment from the parent, no environment variable setup is necessary.

For UNIX/Linux, there are two ways to set up the environment:

- Use the account login profile and source files.
  - For the account to allow PAR to be run remotely, it should have an .rc file (.bashrc if the bash shell used) that runs automatically when a secure shell (ssh**)** call is received.
  - The following three environment variables have to be set properly for PAR to run: PATH, FOUNDRY, and LD_LIBRARY_PATH. To set the environment variables, see “Setting Up the Environment to Run Command Line” on page 566. You can test if the remote environment is set up correctly by running the following command locally:

    ssh <*remote_machine*> par
  - You will see the PAR help page if the environment setup is correct.
- Use Env and Workdir node description lines in the node list file.
  - After it reads the node list, the PAR Multi-tasking option will know which script to run and in which working directory to run the par command. The spawned task on the remote machine will run the “Env” file, then it will “cd” to the working directory before it runs PAR.

There is also UNIX/Linux environment setup information for this feature in the Using the PAR Multi-Tasking (-m) Option section of the topic “Running PAR from the Command Line” on page 579.

**Running Multiple PAR Jobs in Parallel in the Radiant software** When running the multi-tasking option on the PC, you should use a dual processor or multiple processor configuration, or the feature will provide no benefit. Currently, you cannot run the multi-tasking option using multiple networked PCs.

**To run the PAR Multi-tasking on the PC with multiple processors:**

1. Prepare a node list ASCII text file that identifies your machine (lpass4), system type (PC), and number of core processors (2). Include any desired comment lines.

   ```
   // This file contains a profile node listing for a multipar
   // PC job.
   [lpass4]
   SYSTEM = PC
   CORENUM = 2
   ```

   You must use the format above for the node list file and fill in all required parameters. Parameters are case insensitive. The node or machine names are given in square brackets on a single line.

   The System parameter can take the Solaris, Linux, or PC values, depending upon your platform. However, the PC value cannot be used with either Solaris or Linux because it is not possible to create a multiple computer farm with PCs. Corenum refers to the number of CPU cores or processors available on that single PC. Setting it to zero will disable the node from being used. Setting it to a greater number than the actual number of CPUs will cause PAR to run jobs on the same CPU, lengthening the run time. No further parameters are necessary on the PC.

2. Save your node list file and copy it to your top-level project directory. In our example, we named the node list my_nodelist.txt.

3. In the Radiant software, click **Project > Active Strategy > Place and Route Design Settings**.

4. Double click the **Value** cell for the **Multi-Tasking Node List** option and click the browse (···) button.

5. Navigate to the file name in your project directory. If the file is located elsewhere, you must use an absolute file and path name, for example, *C:/my_projects/my_nodelist.txt*. PAR will issue an error message if it cannot find your node list file, for example:

   ```
   ERROR - par: Node name file  does not exist.
   ```

   If it cannot find your node list, PAR will continue in a serial fashion without using the node list file to specify the use of multiple processors.

6. Click **OK**.

   In our example, we opted to change the number of placement iterations to 3, set the placement iteration starting point to 3, and limit the number of best saved runs to 2.

7. In the Process view in the Radiant software environment, right click **Place & Route Design** and choose **Run** from the pop-up menu to run the flow through the Place & Route process.

   Rerun will run the one process over again and nothing else. Rerun All will rerun the design flow up to the Place and Route process.

   In the Output view or Automake Log, the multipar run shows that it ran all jobs and completed successfully.

```
---- Multipar Tool ----
Running par. Please wait . . .
Starting job 5_3 on node lpass4
Starting job 5_4 on node lpass4
Finished job 5_4 on node lpass4
Starting job 5_5 on node lpass4
Finished job 5_3 on node lpass4
Finished job 5_5 on node lpass4
Exiting par with exit code 0
Exiting multipar with exit code 0
Done: completed successfully.
```

Note that we chose the starting point of 3, so the jobs start at 5_3 instead of 5_1.

After the run, PAR creates a <project_name>.dir directory in your top-level project directory.

To learn more about how to use this feature from the command line, see the subsection Using the PAR Multi-Tasking (-m) Option of the topic "Running PAR from the Command Line" on page 579. There is also information there about general usage, environment setup, screen output, interrupting jobs, and requirements.

**See Also** ▶"Running PAR from the Command Line" on page 579

- "Using the PAR Multi-Tasking (-m) Option" on page 581

# Bit Generation

The **Bitstream File** process in the Process view or bit generation (**bitgen**) program takes a fully routed physical design and produces a configuration bitstream (bit images). The bitstream file contains all of the configuration information from the physical design that define the internal logic and interconnections of the FPGA, as well as device-specific information from other files associated with the target device.

The data in the bitstream can then be downloaded directly into the FPGA's memory cells or used to generate files for PROM programming. You can run **bitgen** from the Radiant software window by double-clicking the **Export Files** process or from the command line.

**See Also** ▶"Generating Bitstream Files" on page 191

- "Bit Generation Considerations" on page 193
- "Bit Generation Output Files" on page 191
- "Bit Generation Options" on page 191
- "Running Bit Generation from the Command Line" on page 590

# Bit Generation Options

Bit generation options provide you with control over the bit generation process. Bit generation options are accessed from the Strategy dialog box or the **bitgen** program from the command line. These options allow you to control the format of the bitstream output.

# Bit Generation Output Files

The following files are possible output to the **Bitstream File** process or the **bitgen** program:

- **Bit File** (Binary) — binary (.bin) bitstream. Binary bitstream files are the default output of the bitstream process and contain the configuration information in bitstream (zeros and ones) that is represented in the physical design.
- **Hex File** (Hexadecimal) — Hex files are a file format that conveys binary information in ASCII text form.
- **NVCM File** (Non-Volatile Configuration Memory) — NVCM is a type of computer memory that can retrieve stored information even after having been power cycled (turned off and back on).
- **Raw Bit File** (ASCII) — ASCII (.rbt) bitstream. The Raw Bit File is a text file containing ASCII ones and zeros representing the bits in the bitstream file. If you are using a microprocessor to configure a single FPGA, you can include the Raw Bit file in the source code as a text file to represent the configuration data. The sequence of characters in the Raw Bit file is the same as the bit sequence that will be written into the FPGA. The .rbt file differs from the .bin file in that it contains design information in the first six lines.

**See Also** ▶"Bit Generation" on page 190

# Generating Bitstream Files

In the Radiant software, bitstream generation is automatically performed when you run the **Export Files** process for either a full or partial design flow. You can set bit generation options before running a design flow by setting a strategy.

**To generate bitstream files from the Radiant software:**

1. In the Radiant software File List view, double-click the target strategy.

   The Strategy dialog box opens.

2. Under Process, select **Bitstream**.
3. In the right-hand pane, double-click the Value box for the Bitstream option that you want to edit, enter the new value or select a value from the pulldown list, and click **Apply**.
4. When you finish, click **OK** to close the Strategies dialog box.

5. In the Radiant software Process view, double-click **Bitstream File** to generate the bitstream files.

**See Also** ▶"Bit Generation" on page 190

- "Bit Generation Output Files" on page 191
- "Bit Generation Considerations" on page 193
- "Running Bit Generation from the Command Line" on page 590

# JTAG Setup

Using the **bitgen** program from the command line, you can generate setup bitstreams (.jbt) files to set JTAG port read and write for the FPGA device using the -J option. Downloading these setup bitstreams requires the serial cable with a JTAG cable connector.

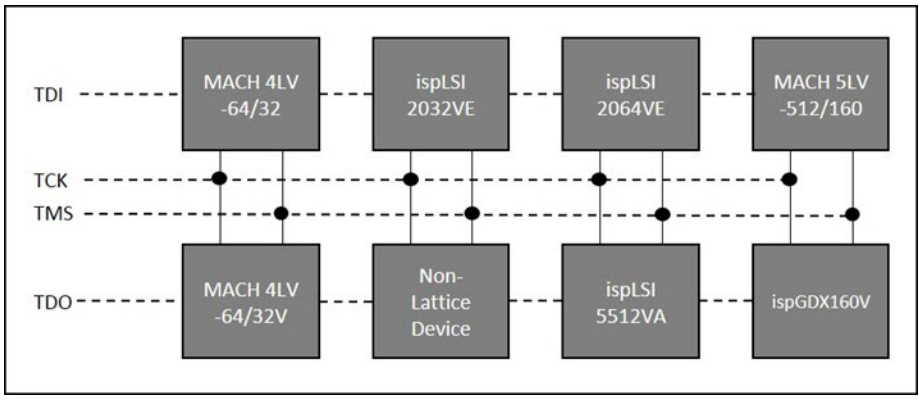
**See Also** ▶"Running Bit Generation from the Command Line" on page 590

# JTAG Scan Chains

A scan chain can include any programmable or non-programmable device compliant with IEEE-1149.1. It can also include any programmable devices that are compatible with IEEE-1149.1 but do not have a boundary scan register. This decision should be made on the basis of the test methodology employed for the board. If the test methodology employed is the traditional bed-of-nails approach used on board test systems, all the devices can be included in the same chain.

All scan chains use the simple four-wire TAP. The TCK and TMS pins are common to all devices included in the chain. TDI and TDO are daisy-chained from one device to the next. The input to the chain is TDI, and the output from the chain is TDO. A diagram demonstrating a simple scan chain is shown in Figure 25.

**Figure 25: Diagram of a JTAG Scan Chain**



# Bit Generation Considerations

Note the following information that may require additional steps to implement bit generation options properly.

- If the device is to be configured in other than a serial (master or slave) mode, there are some limitations involving startup options and configuration pins used as outputs after configuration.
- Some former **-g** options for **bitgen**, which correspond to former strategy settings for **Bitgen** in the Strategies dialog box, are now handled by setting SYSCONFIG keyword values.

**See Also** "Bit Generation Options" on page 191

# Analyzing Static Timing

Static timing analysis is a method for determining if your circuit design meets timing constraints without using simulation. The Static Timing Analysis process employs conservative modeling of gate and interconnect delays that reflect different ranges of operating conditions on various dies, providing complete verification coverage.

This section explains how to use the Radiant software static timing analysis tools and reports to help meet your design's timing constraints.

**See Also** ▶"Static Timing Analysis Tools" on page 194

- "Running Timing Analysis" on page 195
- "Using Timing Analysis View" on page 212

## Static Timing Analysis Tools

The Radiant software provides a tool called Timing Analysis View that performs static timing analysis on timing constraints.

**Timing Analysis View** Timing Analysis View analyzes timing constraints that are present in the .ldc and .pdc files. These timing constraints are defined in the Timing Constraint Editor or in a text editor before the design is mapped.

A Timing Analysis report file, which shows the results of timing constraints, is generated each time you run either the Lattice Synthesis Engine (**LSE**)**, Map Timing Analysis**, or the **Place & Route Timing Analysis** (PAR) process. The results can then be viewed in the Timing Analysis View windows. The Map Timing Analysis report (.tw1) contains estimated routing that can be used to verify the expected paths and to provide an estimate of the delays before you run Place & Route. The PAR Timing Analysis report (.twr) contains delays based on the actual placement and routing and is a more realistic estimate of the actual timing. See "Running Timing Analysis" on page 195.

The Timing Analysis view is a graphical view of the post-route Timing Analysis report, and provides path tables and a report of each timing constraint. It also allows you to cross-probe to Floorplan View to see where these paths exist on the chip. See "Using Timing Analysis View" on page 212 and "Cross-Probing from Timing Analysis View" on page 219.

**See Also** "Running Backannotation from the Command Line" on page 587

# Strategies for Timing Analysis

Reporting options for Timing Analysis View and I/O timing analysis are defined using strategies. Strategies are located in the Strategies folder of the File List view. You can associate a strategy with an active project implementation by right-clicking a strategy and choosing **Set As Active Strategy** from the pop-up menu.

By default, the Timing Analysis view uses your active strategy as its default Timing Analysis View settings when you initially open a view. Your Map and PAR Number of Paths Per Constraint and Number of Unconstrained Paths setting defaults to reporting 10 in the Timing Analysis View report.Timing Analysis Options is also defaulted to Standard Setup and Hold Analysis.

For more general information on setting strategies in the Radiant software, see "Using Strategies" on page 21.

**See Also** "Using Timing Analysis View" on page 212

- "Timing Analysis View Features" on page 213

# Running Timing Analysis

You can access the Map Timing Analysis and Place & Route Timing Analysis tools from the Task Detail View menu on the Process Toolbar. You can also access the Timing Analysis View tool using the `timing` command in the Tcl Console, a DOS console, or from an xterm in Linux. Timing Analysis View runs analysis on timing constraints that have been specified in the logical constraint file (.ldc), post synthesis design constraint file (.pdc) or the HDL source, and it generates a report of the results. You can view the Timing Analysis View report in the Reports window.

**To run Timing Analysis View from the Radiant software environment:**

- Select Task Detail View and choose **Map Timing Analysis** or **Place & Route Timing Analysis**.

  Timing Analysis View runs static timing analysis on your design and generates a map Timing Analysis View report (.tw1) or a PAR Timing Analysis View report (.twr).

**To run Timing Analysis View from the command line:**

- The following shows the simple TCL command:

```
timing -sdc <file_name>.ldc <file_name>.udb -o
<report_name>.twr
```

**To run I/O Timing Analysis View from the command line:**

- The following shows the simple TCL command:

```
timing -iotime <udb_file_name> -o <io_file_name>
```

  The output will be a *.ior file.

For more information, see "Running Timing from the Command Line" on page 584.

**See Also** "Static Timing Analysis Tools" on page 194

- "Using Timing Analysis View" on page 212

# Setting Timing Analysis Options

Timing Analysis View options are available in the active Strategy for an implementation. The options provide you with control over how timing data is reported.

**To set Timing Analysis View options in the Radiant software environment:**

- Choose **Project > Active Strategy**, and then choose either **Map Timing Analysis Settings** or **Place & Route Timing Analysis Settings** from the pulldown menu.

  Optionally, double-click the active Strategy in the Strategies folder of File List view, and then select **Map Timing Analysis** or **Place & Route Timing Analysis**.

The following Timing Analysis options are available for both Map and Place & Route Timing Analysis View:

- Analysis Options
  - Standard Setup and Hold Analysis (default) – Performs both the Setup Analysis and the Hold Analysis.
  - Hold Analysis – Performs hold analysis.
  - Standard Setup Analysis – Performs setup time checks on applicable constraints.

  For other settings, please refer to setting strategies: "Using Strategies" on page 21

  Here are some important points about connection coverage to consider:

- Some signal connections might not be constrained by the other predefined types of constraints. It is possible that all pairs of source and sink are covered by the given constraints, but there could still be connections not covered by any of the given constraints. Therefore, the connection coverage might be less than 100%.
- Sometimes an unconstrained path is found even though the connection coverage is reported as 100%. The unconstrained path refers to the path between a source and sink pair. If a legal source and sink pair is not covered by a constraint, it will be reported as an unconstrained path. Because the connections on this unconstrained path might already be covered by some other constraints, the connection coverage could still be 100%.

**See Also** "Static Timing Analysis Tools" on page 194

- "Running Timing Analysis" on page 195
- "Using Timing Analysis View" on page 212
- "Timing Analysis View Features" on page 213

# Viewing Timing Analysis Reports

Timing Analysis View reports are outputs of the Map Timing Analysis and Place & Route Timing Analysis processes. After running either of these processes from Task Detail View, you can examine the associated Timing Analysis View reports in the Timing Analysis View window. The Map Timing Analysis View report (.tw1) and the Place & Route Timing Analysis View report (.twr) files are written out to your project directory in ASCII format using the current project name for the file name.

**To view the Timing Analysis View report file in the Report window:**

1. Click on the menu item, **View > Reports** to activate the Report window (or click on the **Reports** tab in the main view).
2. Select **Map Reports > Map Timing Analysis** or **Place & Route Report > Place & Route Timing Analysis and/or I/O Timing Analysis** from the Reports pane.

   The selected report appears in the Reports window. The Design Summary pane presents a hierarchy view of the report.

   - Expand a section of the report in the Design Summary pane and click a subsection name of the .HTML link to open the section in the Reports window.
   - In the Report window, right-click and you can choose to cross probe into the **Netlist Analyzer RTL/Tech** tool.

**To view the Timing Analysis View report file in a text editor:**

- Open your text editor and navigate to the project directory. The Timing Analysis View reports are usually located in the <project_name><implementation_name> folder.

- To open the Map Timing Analysis View report, select the <project_name>.**tw1** file.
- To open the I/O Timing Analysis View report, select the <project_name>.**ior** file.
- To open the Post-Par Timing Analysis View report, select the <project_name>.**twr** file

**See Also** "Timing Analysis Report File" on page 198

- "Running Timing Analysis" on page 195
- "Setting Timing Analysis Options" on page 196
- "Timing Analysis View Features" on page 213

# Timing Analysis Report File

The Timing Analysis Report File (*_lse.twr/.tw1/.twr) is an ASCII report that enables you to determine to what extent the timing constraints for a design have been met. The *_lse.twr Timing Analysis report output file is a result of the post LSE Timing Analysis process in the Radiant software. The .tw1 Timing Analysis View report output file is a result of the Map Timing Analysis process in the Radiant software. The .twr Timing Analysis View report output file is a result of the Place & Route Timing Analysis process.

Only a verbose Timing Analysis report is generated. The following are some facts regarding verbose levels of Timing Analysis View timing reporting:

- Report concludes with Timing Summary subsections.
- The number of items reported can be limited for each constraint.
- The Timing Analysis View output format supports logical names.
- Reports also contain an combinational loop report that shows all paths that cannot be analyzed.
- The report lists delay information for all nets and paths.
- If constraints for process, temperature, or voltage exist in a constraint file, these constraints are included in the timing report.
- The details of the paths that contribute to clock skew are reported.
- The result of maximum frequency ($f_{MAX}$) in Timing Analysis View is potentially the highest frequency that the design can achieve. However, there are factors that can change the way in which $f_{MAX}$ is reported. For example, if the $f_{MAX}$ path is PLL-relevant and the PLL is configured as cycle "SHIFT", then the maximum frequency will change PLL "SHIFT" delay accordingly, which might impact clock skew and $f_{MAX}$ results. Final $f_{MAX}$ results depend upon how PLL is used in each individual case.

## Timing Analysis Report

The Radiant software's new timing report features four major sections:

- Design Checking
- Clock Summary
- Timing Analysis Summary
- Detailed Report

Each section is explored in detail below.

For each constraint that is not met, the report shows the number of items scored by Timing Analysis View, the number of errors encountered, and a detailed breakdown of the error. In addition, Timing Analysis View reports clock skew details for placed and routed designs.

A Timing Summary detailed report at the end details all the Setup and Hold timing paths.

The detailed report section, which reports circuit path source and destination information, appears at the end. If a path begins or ends at a sequential element, such as a flip-flop, the clock signal associated with the flip-flop is reported in the path delay report.

```
++++ Path 1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Path Begin       : directionR_21/Q
Path End         : counter1/countai_101__i7/D
Source Clock     : clk_1Hz
Destination Clock: clk
Logic Level      : 9
Delay Ratio      : 57.4% (route), 42.6% (logic)
Setup Constraint : 10.000 ns
Path Slack       : -1.780 ns  (Failed)

  Destination Clock Arrival Time (clk:R#8)   210.000
+ Destination Clock Source Latency             0.200
- Destination Clock Uncertainty               0.000
+ Destination Clock Path Delay                 6.100
- Setup Time                                   0.199
----------------------------------------   -------
End-of-path required time( ns )             216.101

  Source Clock Arrival Time (clk_1Hz:R#3)   200.000
+ Source Clock Source Latency                 0.000
+ Source Clock Path Delay                     8.344
+ Data Path Delay                             9.537
----------------------------------------   -------
End-of-path arrival time( ns )              217.881
```

Source and destination clock signals are listed because the path originates at a flip-flop and terminates at a flip-flop with a setup requirement relative to the signal clock. This allows you to determine what the source and destination clock signals are for sequential paths.

The body of the report enumerates each constraint as it appears in the input constraint file, the number of items scored by Timing Analysis View for that constraint, and the number and description of errors detected for the constraint. A report line for each constraint provides important information, such as the amount of delay on a net and by how much the constraint is met.

For path constraints, if there is an error, the report indicates the amount by which the constraint is exceeded. If there are no errors, the report indicates that the constraint passed and by how much. Each logic and route delay is analyzed, totaled, and reported.

With regards to Minimum Pulse Width (MPW), if a clock net goes to a pin with a minimum pulse width arc, the corresponding clock's report will contain a MPW report immediately after the constraint. Each clock constraint has a maximum of one MPW report.

## Timing Analysis Report Sections

Figure 26 shows the sections that a given Timing Analysis View Report (.tw1/ .twr) file might include. These sections will vary, depending on the FPGA architecture, timing constraints, design elements, and reporting style.

**Table of Contents** There are four major sections that is detailed in the Table of Contents: 1. Design Checking, 2. Clock Summary, 3. Timing Analysis Summary, and 4. Detailed report. Each line item in the Table of Contents is also a hyper link that will take you directly to that section of the report.

**Figure 26: Timing Analysis Report Table of Contents**

```
============================================================
                    Table of Contents
============================================================
    1  DESIGN CHECKING

        1.1  SDC Constraints

        1.2  Combinational Loop

    2  CLOCK SUMMARY

        2.1  Clock clk

        2.2  Clock clk_1Hz

    3  TIMING ANALYSIS SUMMARY

        3.1  Overall (Setup and Hold)

            3.1.1  Constraint Coverage

            3.1.2  Timing Errors

            3.1.3  Total Timing Score

        3.2  Setup Summary Report

            3.2.1  Setup Constraint Slack Summary

            3.2.2  Setup Critical Endpoint Summary

        3.3  Hold Summary Report

            3.3.1  Hold Constraint Slack Summary

            3.3.2  Hold Critical Endpoint Summary

        3.4  Unconstrained Report

            3.4.1  Unconstrained Start/End Points

            3.4.2  Start/End Points Without Timing
Constraints

            4  DETAILED REPORT

        4.1  Setup Detailed Report (all constraint based
except set_clock_latency, set_clock_uncertainty)
direction]

        4.2  Hold Detailed Report (all constraint based
except set_clock_latency, set_clock_uncertainty)
```

**Version, Build, Time Stamp, and Copyright** The date, time, the file generated, the software version that created it, and all relevant copyright information are all displayed at the top of a report.

```
Timing Report
Lattice Timing Report -  Setup  and Hold, Version Radiant
(64-bit) 1.0.0.1258.0

Mon Jun 19 16:42:58 2017

Copyright (c) 1991-1994 by NeoCAD Inc. All rights
reserved.
Copyright (c) 1995 AT&T Corp.   All rights reserved.
Copyright (c) 1995-2001 Lucent Technologies Inc.  All
rights reserved.
Copyright (c) 2001 Agere Systems   All rights reserved.
Copyright (c) 2002-2017 Lattice Semiconductor
Corporation,  All rights reserved.

Command line:    timing -sethld -v 10 -u 10 -endpoints 20
-nperend 1 -html -rpt tutorial_impl1_twr.html
tutorial_impl1.udb -gui

------------------------------
Design:          topcount
Family:          ice40up
Device:          iCE40UP5K
Package:         SG48
Performance:     6
------------------------------
```

**Design Checking** TThe Design Checking section contains two further sub-sections: SDC Constraints and Combinational Loop. SDC Constraints lists all the LDC constraints as supplied by user and also any constraints that are automatically generated from PLL device constraints. The second sub-section list combinational / asynchronous loops in the circuit for which timing paths cannot be analyzed.

```
1  DESIGN CHECKING

1.1  SDC Constraints

create_clock -name {clk} -period 30 -waveform {0.000000
15.000000} [get_ports clk1]
create_clock -name {clk_1Hz} -period 100 [get_nets counter3/
clk_1Hz]
set_multicycle_path -rise_from [get_clocks clk] -rise_to
[get_clocks clk_1Hz] 1
set_clock_latency -source 0.2 [get_clocks clk]
set_clock_uncertainty 0.3 [get_clocks clk]
set_input_delay -clock [get_clocks clk] 5 [get_ports
direction]
set_output_delay -clock [get_clocks clk] 5 [get_ports seg_1]
1.2  Combinational Loop

Combinational Loops
-------------------
++++ Loop1
my_LEDtest/i372_4_lut/B->my_LEDtest/i372_4_lut/Z

++++ Loop2
my_LEDtest/i370_4_lut/C->my_LEDtest/i370_4_lut/Z

++++ Loop3
my_LEDtest/i374_4_lut/C->my_LEDtest/i374_4_lut/Z
```

**Clock Summary** The Clock Summary section shows all the defined clocks in the .ldc files. The paths will show the capturing clock FROM signal to signal indicating launch clocks. Domain crossing clocks are also shown indicating that the launch clock and capture clocks are different.

**Timing Analysis Summary** The Timing Analysis Summary gives an overall roundup of all the Setup and Hold timing results. Constraint Coverage indicates the number of paths that the timer was able to cover based on sequentially timed paths. Timing Errors indicates the number of endpoints which violate setup or hold violations. There is also a sum of the Total Negative Slack in terms of setup and hold in the Total Timing Score summary section.

Section 3.1.3 in Figure 27 shows a sum of the Total Negative Slack in terms of setup and hold in the Total Timing Score summary section.

Section 3.2.1 Figure 27 indicates the Setup Constraint Slack Summary in which each SDC constraint is detailed and reported in terms of the required timing target, slack, logic levels and timing Score indicates the number of

timing endpoints that is timed by that constraint, and any errors for that constraint. In contrast, section 3.3.1 in Figure 27 provides the equivalent for the Hold Summary reports.

The Logical Details subsection reports circuit path Source and Destination information:

- Instance names for the following cell types: Registered elements, Ports, RAM instances.
- Indication of the triggering edge of the clock using (+/-).

The Physical Path Details shows the paths connected to their source and destination sites on the chip. At the bottom of the Delays column, the total amount of delay for the path, the percentages of the total allocated to logic and to route delays (in parentheses), and the number of logic levels (components) involved in the constraint are shown.

The first line indicates the path number of the report The header contains a line for the data path start point or the pin at which the path begins. This particular path starts at the Q output of a flip flop. The next line in the header indicates the end point of the data path. This is followed by the source and destination clock lines. The next two lines indicate the number of logic levels and the delay ratio (%route delay and %logic delay). The last two lines of the header contain the maximum delay allowed for the data path, including setup and trigger arc delays if applicable, and the slack.

Timing errors, (negative slack), indicate absolute timing constraint violations. Timing errors may indicate a need for design modifications and/or multiple placement and/or reentrant routing.

The most critical paths are reported from top to bottom in this section. These paths are all ranked in terms of weighted slack values based on one full clock cycle. If the path is not a full cycle, you might see something like “weighted slack = 2.000ns” following the pass/fail statement. This simply indicates that the actual timing value is scaled up to a full clock cycle value and that the path was scored at 1.00ns at a half cycle.

In short, Timing Summary summarizes:

- The number of timing errors found.
- A timing score showing total errors in picoseconds for all timing constraints.
- The number of paths and connections covered by the constraints and the percentage coverage over the whole design.
- A common errors matrix which reports the critical nets that are responsible for more than 10 percent of timing errors.

**Figure 27: Example of Timing Analysis Summary**

```
2  CLOCK SUMMARY

2.1 Clock clk

create_clock -name {clk} -period 30 -waveform {0.000000 15.000000} [get_ports clk1]

Single Clock Domain
--------------------------------------------------------------------------------
--------------------
              Clock clk               |                   |       Period       |
Frequency
--------------------------------------------------------------------------------
--------------------
      From clk
                                      |            Target |         30.000 ns |
33.333 MHz

                                      | Actual (all paths) |         31.770 ns |
31.476 MHz
--------------------------------------------------------------------------------
--------------------

Clock Domain Crossing
--------------------------------------------------------------------------------
--------------------
               Clock clk                |   Worst Time Between Edges   |
Comment
--------------------------------------------------------------------------------
--------------------
      From clk_1Hz
                                         |                   10.000 ns |
slack = -9.444 ns
--------------------------------------------------------------------------------
--------------------
2.2 Clock clk_1Hz

create_clock -name {clk_1Hz} -period 100 [get_nets counter3/clk_1Hz]

Single Clock Domain
--------------------------------------------------------------------------------
--------------------
            Clock clk_1Hz             |                   |       Period       |
Frequency
--------------------------------------------------------------------------------
--------------------
      From clk_1Hz
                                      |            Target |        100.000 ns |
10.000 MHz

                                      | Actual (all paths) |         11.315 ns |
88.378 MHz
--------------------------------------------------------------------------------
--------------------

Clock Domain Crossing
--------------------------------------------------------------------------------
--------------------
             Clock clk_1Hz              |   Worst Time Between Edges   |
Comment
--------------------------------------------------------------------------------
--------------------
      From clk
                                         |                   10.000 ns |
slack = 4.634 ns
--------------------------------------------------------------------------------
```

```
3  TIMING ANALYSIS SUMMARY

3.1  Overall (Setup and Hold)

3.1.1  Constraint Coverage

Constraint Coverage: 85.2632%

3.1.2  Timing Errors

Timing Errors: 7 endpoints (setup), 1 endpoints (hold)

3.1.3  Total Timing Score

Total Negative Slack: 13.631 ns (setup), 4.247 ns (hold)

3.2  Setup Summary Report

3.2.1  Setup Constraint Slack Summary
--------------------------------------------------------------------------------
                                        |             |             |        |
Actual (flop to flop)   |          |
            SDC Constraint              |   Target    |    Slack    | Levels |
Period    |  Frequency  |  Score   |  Errors
--------------------------------------------------------------------------------
                                      |            |            |        |
|             |          |
create_clock -name {clk} -period 30 -wa
veform {0.000000 15.000000} [get_ports
clk1]                                   |    -----    |    -----    |   ---- |
---- |        ---- |    69    |    --
                                      |            |            |        |
|             |          |
create_clock -name {clk_1Hz} -period 10
0 [get_nets counter3/clk_1Hz]           |    -----    |    -----    |   ---- |
---- |        ---- |    29    |    --
                                      |            |            |        |
|             |          |
set_multicycle_path -rise_from [get_clo
cks clk] -rise_to [get_clocks clk_1Hz]
1                                       |    -----    |    -----    |   ---- |
---- |        ---- |     0    |     0
                                      |            |            |        |
|             |          |
set_input_delay -clock [get_clocks clk]
 5 [get_ports direction]                |   10.000 ns |    1.660 ns |    2   |
---- |        ---- |     1    |     0
                                      |            |            |        |
|             |          |
set_output_delay -clock [get_clocks clk
] 5 [get_ports seg_1]                  |   10.000 ns |   -9.404 ns |    2   |
---- |        ---- |     1    |     1
--------------------------------------------------------------------------------
-----------------------------------------
```

**Unconstrained Report** This section is included in the Timing Analysis Summary report section 3.4. In the Unconstrained Paths section, Timing Analysis Summary reports the paths that are not constrained and shows the start point and end point of each path. Suggested timing constraints are provided to constrain the given paths.

Based on the design and the required performance, only necessary paths should be constrained so that PAR focuses only on the optimization of the important paths. However, the Unconstrained Report section of the Timing Analysis Summary report is very useful for identifying whether any missing timing constraints are really important to the design. This option does not require you to add more constraints in an attempt to constrain all paths. Instead, it serves as a reminder that there might be a necessary constraint that is missing, which could impact the desired performance of the design.

As shown in the following example, Timing Analysis View reports only the “setup” timing of unconstrained paths to avoid duplication of these same paths in a “hold” timing report.

The following shows all the Unconstrained Start/End Points. Unconstrained start points indicate that the clocked primitive departure pin is not driving any constrained (clocked) sequential element. Conversely, Unconstrained end points indicate that there is an unconstrained element driving a clocked sequential element.

The report shown in section 3.4.1 of Figure 28 also indicates Unconstrained Start/End points

**Figure 28: Example of an Unconstrained Report**

```
3.4.1  Unconstrained Start/End Points

Clocked but unconstrained timing start points
------------------------------------------------------------------
----
               Path End               |          Type
------------------------------------------------------------------
----
seg_12_33/PADDO
                                      |          No required
time
seg_13_34/PADDO
                                      |          No required
time
------------------------------------------------------------------
----
                                      |
Number of unconstrained timing start po |
ints                                  |
26
                                      |
------------------------------------------------------------------
----
```

```
------------------------------------------------------------------
----

Clocked but unconstrained timing end points
------------------------------------------------------------------
----
               Path End               |          Type
------------------------------------------------------------------
----
counter1/countai_101__i7/SR
                                      |           No arrival
time
{counter1/countai_101__i5/SR   counter1
/countai_101__i6/SR}                  |           No arrival
time
------------------------------------------------------------------
----
                                      |
Number of unconstrained timing end poin |
ts                                    |
21
                                      |
------------------------------------------------------------------
----
```

There are two remaining sections that indicate I/O ports without constraints and endpoints with False Path constraints.

**Detailed Report** This report details up to 10 timing paths for every .SDC constraint that was declared in the .ldc constraint files. The two main sections (4.1 and 4.2) are shown in Figure 29, and further quantifies the timing analysis in to Setup and Hold detailed reports.

The Path Details subsection reports circuit path Source and Destination information.

The section includes source component logical name, clock name and edge; and it includes destination component logical name, clock name and edge.

The first section is a summary of the path indicating the source and destination clocks, logic levels, delay ratios in terms of percentage of route and logic and most importantly whether the Path being analyzed has a positive or negative slack to indicate meeting or failing the constraint.

The R#X in the source / destination indicates the Rising (R) or Falling (F) edge and at which edge of the clock it is captured. i.e. R#3

The detailed path details shows the paths connected to their source and destination sites on the chip. At the bottom of the Delays column, it shows the total amount of delay for the path, the percentages of the total allocated to

logic and to route delays (in parentheses), and the number of logic levels (components) involved in the constraint.

**Figure 29: Sample of a Detailed Report**

```
Paths for the various constraints

4.1.1  Setup path details for constraint: create_clock -name
{clk} -period 30 -waveform {0.000000 15.000000} [get_ports
clk1]
------------------------------------------------------------
-------
69 endpoints scored, 7 timing errors detected.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
                    Detailed Report for timing paths
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++

++++ Path 1
++++++++++++++++++++++++++++++++++++++++++++++++++++++

Path Begin       : directionR_21/Q
Path End         : counter1/countai_101__i7/D
Source Clock     : clk_1Hz
Destination Clock: clk
Logic Level      : 9
Delay Ratio      : 57.4% (route), 42.6% (logic)
Setup Constraint : 10.000 ns
Path Slack       : -1.820 ns  (Failed)

  Destination Clock Arrival Time (clk:R#8)   210.000
+ Destination Clock Source Latency             0.200
- Destination Clock Uncertainty               0.000
+ Destination Clock Path Delay                 6.100
- Setup Time                                   0.199
----------------------------------------   -------
End-of-path required time( ns )             216.101

  Source Clock Arrival Time (clk_1Hz:R#3)   200.000
+ Source Clock Source Latency                 0.000
+ Source Clock Path Delay                     8.384
+ Data Path Delay                             9.537
----------------------------------------   -------
End-of-path arrival time( ns )              217.921
```

Timing errors, however, indicate absolute timing constraint violations. Timing errors may indicate a need for design modifications, and/or multiple placement, and/or re-entrant routing.

The most critical paths are reported from top to bottom in this section. These paths are all ranked in terms of weighted slack values based on one full clock cycle.

**I/O Timing Analysis Report** The timer will traverse all the speed grades of the device and output the worst setup and hold times for each port with the worst performance grade for each. It will also output the maximum and minimum clock to output delays. The design must be properly constrained for the timer to write out the IO timing report.

```
I/O Timing Report
==================

FPGA input ports results across Performance Grades (6, M):
-----------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
   Port Name    |     Setup      | Grade | edge |      Hold
| Grade | edge |   Clock Port
------------------------------------------------------------
------------------------------------------------------------
spi_csn         |        -1.262 ns |   6   |   R  |
1.093 ns |    6  |   R  |h_clk_cnt_i1_i0/Q(internal clock)
spi_csn         |        -2.797 ns |   6   |   F  |
3.904 ns |    6  |   R  |spi_sclk
spi_mosi        |        -3.881 ns |   6   |   R  |
4.196 ns |    6  |   R  |spi_sclk
------------------------------------------------------------
------------------------------------------------------------

FPGA output ports results across Performance Grades (6, M):
-----------------------------------
------------------------------------------------------------
------------------------------------------------------------
   Port Name    |Clock To Out (MAX)| Grade | edge |Clock To Out
(MIN)| Grade | edge |   Clock Port
------------------------------------------------------------
------------------------------------------------------------
spi_miso        |        13.104 ns |   6   |   R  |
12.817 ns |   6   |   R  |spi_sclk
------------------------------------------------------------
------------------------------------------------------------
```

# Cross-Probing to LSE Netlist Analyzer

The Radiant software, together with the LSE tool, enables you to cross-probe a path from the Timing Analysis report to Netlist Analyzer. This provides you with a schematic view of the path and helps you diagnose problem areas to improve the timing.

**To cross-probe from Timing Analysis View to the LSE Netlist Analyzer:**

1. In the Radiant software, run the **Place & Route Timing Analysis** process.

2. In the menu item select **View > Reports** to bring up the Reports table of contests page.
3. Expand the Place & Route Report and select **Place & Route Timing Analysis** (.twr file).
4. For any path start/end, data path component or timing path (.twr file), drag the mouse to highlight it.
5. Right click and select **Filter in Netlist Analyzer (RTL / Tech)**.

# Using Timing Analysis View

The Timing Analysis view provides a graphical user interface for the static timing analysis program, Timing Analysis View. As explained in "Static Timing Analysis Tools" on page 194, the Timing Analysis View allows you to view the path delay tables and Timing Analysis View report of your timing constraints after placement and routing. There are five main windows (tabs) in the Timing Analysis tool.

**General Information** Timing Analysis View's general information tab provides the basic information of the design in the top section. In the lower section, Timing Option setting information parameters can be set via the Project > Active Strategy section in the Map and P&R Timing Analysis strategy settings.

**Paths for All the Timing Constraints** This tab presents the user with all the Constraints for Setup and Hold Analysis. When a constraint is selected in the Constraint sub window, a Path Summary of all the different timing paths for the selected timing constraint is shown with information including the Start Point, End Point, Slack, Delay and Clock information. Upon clicking on any of the individual paths, a new sub window containing the Path Detail information is shown. Below that are two tabbed sub windows containing the Data Path and Clock Paths data.

**Critical Endpoint Summary** Similar to the previous tab, a list is shown with all the Path End elements and the associated slack value for the Analysis Type. Clicking on a Path End element also opens a new sub window containing Path Detail information. Below this are two tabbed sub windows containing the Data Path and Clock Paths data.

**Unconstrained Endpoint Summary** This tab gives a summary of all the variances of Unconstrained Endpoints. All start and end points that are properly clocked by a clock signal but are not constrained are listed.

For start points, this could happen either because there is no path to an end point or because the endpoint is not properly constrained. For endpoints the cause is either a missing path from a start point or an unconstrained start point. The following four subsections are also available

- Clocked but unconstrained timing start points - No slack calculated due to missing departure timing from clocked element.
- Clocked but unconstrained timing end points - No slack calculated due to missing arrival from clocked element.
- I/O ports without constraint - Any input or output port that is not constrained.
- Registers without clock definition - Clocked registers that are missing defined clocks.

**Query** A user can query the timing path by specifying source clock, destination clock, start point or end point (one condition at least). A user can also save and open these queries by clicking on the **Open** and **Save** buttons which will open a separate dialog.

See Also

- "Cross-Probing from Timing Analysis View" on page 219
- "Rearranging the Timing Analysis View Layout" on page 218

# Opening Timing Analysis View

Timing Analysis View becomes available after placement and routing. When you first open Timing Analysis View, it displays timing information based on constraints in the HDL source files, active post synthesis constraint file (.pdc) and the active logical constraint file (.ldc).

**To open Timing Analysis view:**

- In the Radiant software, choose **Tools > Timing Analysis View**.

  A progress indicator message box opens, showing that the Radiant software is calculating the delays. When the progress indicator closes, the timing constraints, settings, and analysis are displayed in the delay path tables, schematic, and report panes to the right.

  **Note**

  Each time you start Timing Analysis view, the Radiant software loads from the UDB with default timing constraints

**See Also** "Rearranging the Timing Analysis View Layout" on page 218

# Timing Analysis View Features

Timing Analysis View provides five main tab views consisting of multiple panes for viewing timing constraints, path tables, and the Timing Analysis View report. The main window enables you to examine the results of timing constraints, cross-probe delay paths to the Floorplan View, and troubleshoot paths of the design that restrict performance. See Also

- "Rearranging the Timing Analysis View Layout" on page 218
- "Cross-Probing from Timing Analysis View" on page 219

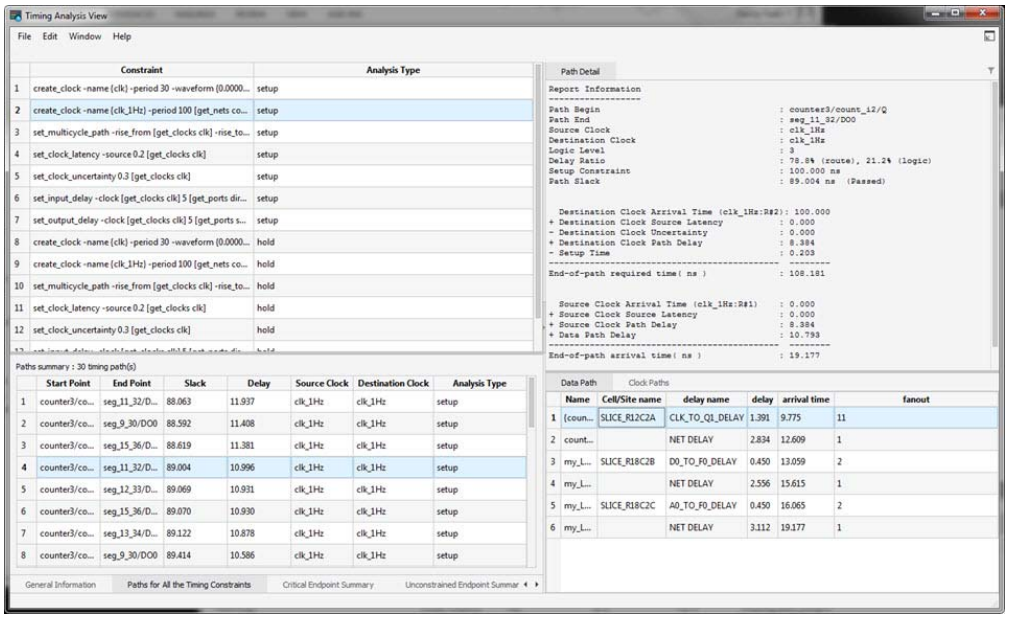
## Timing Analysis View Main Window Tabs

When you open Timing Analysis View, you are presented with five tabs, as shown in Figure 30:

- General Information
- Paths for All the Timing Constraints
- Critical Endpoint Summary
- Unconstrained Endpoint Summary
- Query

Each of the tab panes can be detached from the main window, rearranged, and resized (see “Rearranging the Timing Analysis View Layout” on page 218). When you select a constraint from the Constraint pane, you can view the path table details while viewing the Timing Analysis View report in the a second pane. This view provides multiple panes, as shown in Figure 30.

**Figure 30: Timing Analysis View Main Window**




There is a cross-probing features for viewing key timing data in your design, as well as the start points, end points, and categories of unconstrained paths. For example, you can click on a constraint you defined on a particular clock net or port and see the slack associated with it, the delay value, and the source and destination pins. Or you can click on a listed constraint for an unconstrained path and view the start point, end point, classification, and clocks. In addition, you can cross-probe a timing path to Floorplan View. This can be useful for reconfiguring or regrouping elements to optimize the timing on a critical path.

## General Information

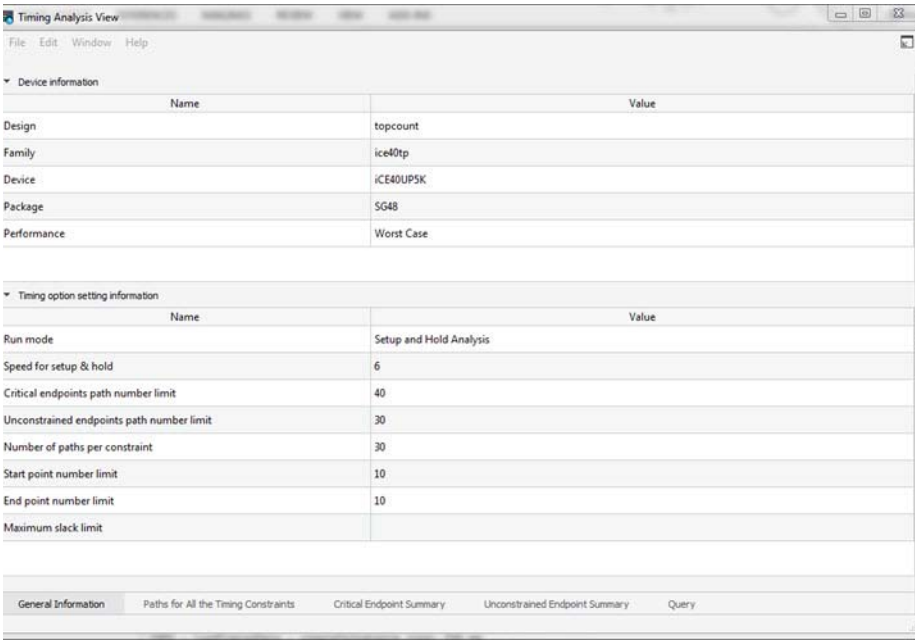
The General Information tab displays the targeted Device, Family, Package, and Performance, as shown in Figure 31. The timing options setting information can be set in the Strategies form.

The following are defined:

- Design - the top level design name.
- Family – the product series name of the Lattice FPGA part. This setting is read-only.
- Device – the product’s alpha-numeric part name that appears within the software and for ordering. The name’s components indicate device family, package, and device size. This setting is read-only.

- Package – the type of package that will house the FPGA chip and is concerned with facilitation of pin connectivity using package leads from device pads. This setting is read-only.
- Performance – current performance reading for the device.
- Run mode – either Setup or Hold or both analysis.
- Critical endpoints path number limit – the number of critical endpoints to be listed per constraint.
- Unconstrained endpoints path number limit – the number of unconstrained endpoints to be listed per constraint.
- Start / End point number limit – lists the endpoints for paths with clocking constraint but no departure or arrival element.
- Maximum slack limit – the maximum delay value for the slack.

**Figure 31: Timing Analysis View Tabs**




### Paths for All the Timing Constraints

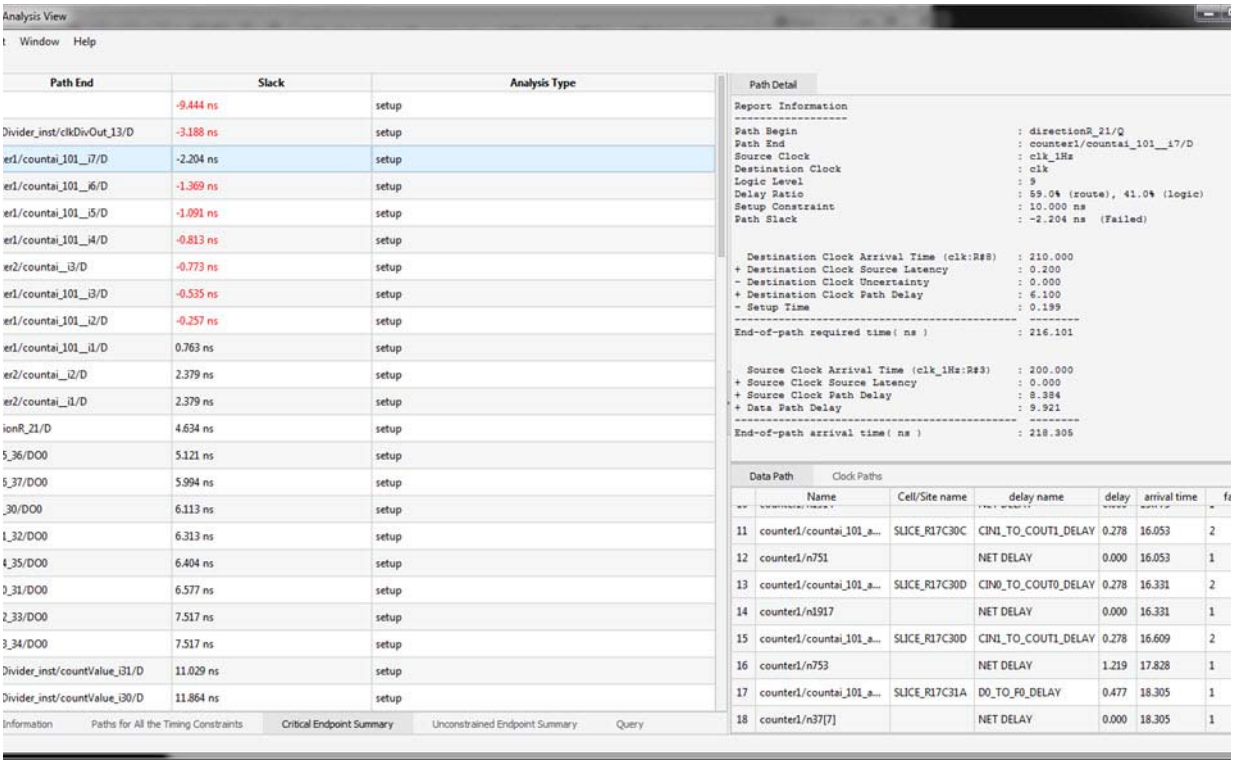
As explained above, the setup and hold summary reports contain a table indicating each clock and each exception and the worst slack for that particular clock or exception. The slack and actual frequency of the clocks are also be printed in this table. This table will indicate whether the constrained paths meet the required constraints. You will know if the design meets timing from these tables.

### Critical Endpoint Summary

This table is followed by a critical endpoints list. This list contains failing endpoints and their slacks. If there are no endpoints with a negative slack, the list will have a line indicating that there are no endpoints with a negative slack.

You will know that there is a failure from the clock/exception table but will not know how many endpoints failed or what the total negative slack of the endpoints is from that table. See Figure 32.

**Figure 32: Critical Endpoints Table**



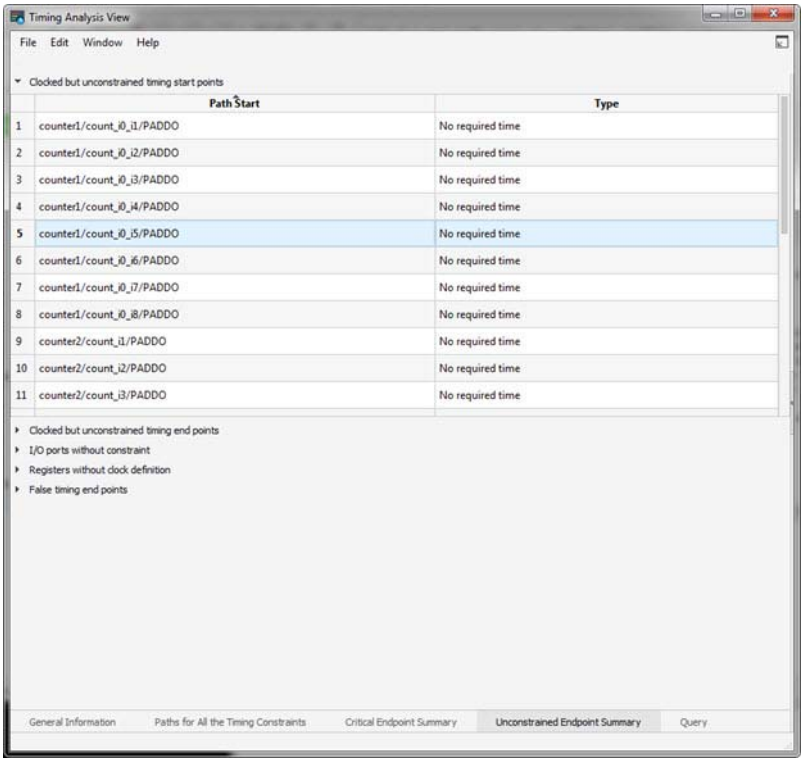

The following data is included in the path tables:

- Cell/Site name - Start point of timed element. Right clicking on this name, you can cross probe to Floorplan view.
- Weighted Slack – a measure of the amount of time that a path passes or fails the timing constraint. A positive slack indicates the margin by which the path surpasses the clock constraint. Negative slack indicates a failing path. The slack is “weighted” or scaled to one full clock cycle so that paths that have different cycles can be ranked according to their severity so that the most critical paths are ordered properly in the table.
- Arrival time– the amount of elapsed time in arriving at the destination.
- Delay – the time related to the path in nanoseconds. This value is typically a composite of multiple timing arcs including data and clock arrival times.
- Fanout– the number of inputs an output logic gate of the same type has in the design.

### Unconstrained Endpoint Summary

This shows a list of the element at path start. As mention above, this pane lists all of the endpoints that either has no constraint defined even though path is clocked. This indicates that there is no departure or arrival primitive or port. For example, an I/O port without constraint indicates an input/output pin that doesn't have a set_input_delay or a create_clock constraint. Outport ports without a set_output_delay constraint are also listed in this table. A pure false path will also be listed in the False timing end points. Figure 33 shows an example of this.

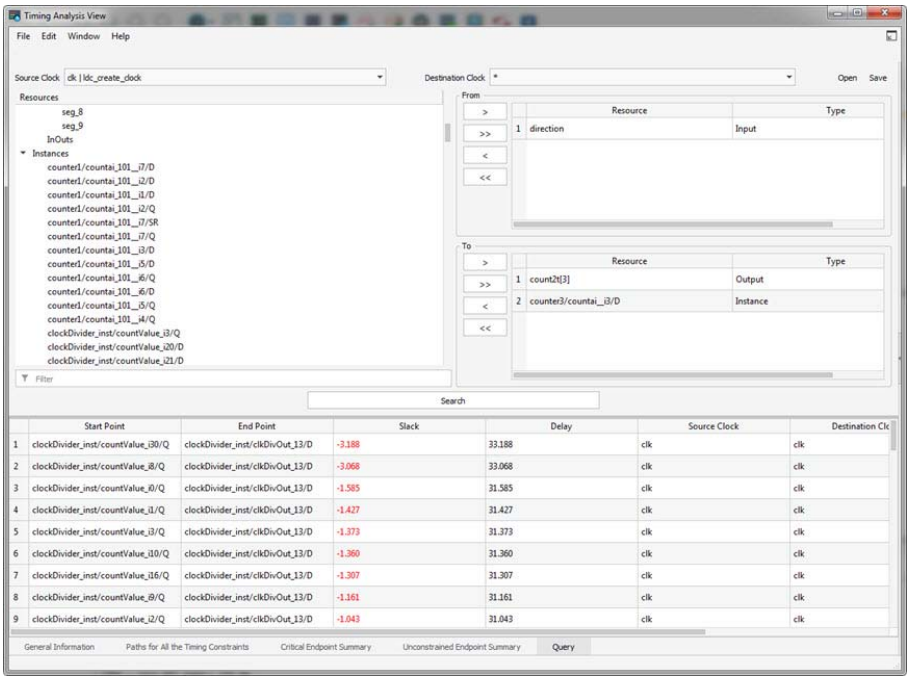
**Figure 33: .Unconstrained Timing Start Points**




### Query

This pane is used to filter out certain detailed constrained paths. Once a source or destination clock is selected from the pulldown menu, you can select the instances and IOs signals by moving the signals back and forth from the From and To panes. Once **Search** is clicked, all the found paths will be listed in the path detail window. Upon selecting a constraint, details of the full path are listed and described. A user can also Open or Save each of the queries while selecting the different IOs, instances, etc. to start the query. See Figure 34 for an example.

**Figure 34: Query Sample**




**See Also** ▶"Rearranging the Timing Analysis View Layout" on page 218

- "Cross-Probing from Timing Analysis View" on page 219
- "Setting Timing Analysis Options" on page 196

# Rearranging the Timing Analysis View Layout

Timing Analysis View's flexible interface enables you to detach the different panes, rearrange them, and resize them to make the timing details easier to view.

**Detaching or Reattaching Timing Analysis View** You can detach Timing Analysis View from the Radiant software's main window to give you more room to examine timing data and rearrange the layout of panes.

**To detach Timing Analysis View:**

- Click the Detach Tool button at the top right corner of the Timing Analysis View main window..

  This detach button is represented by a dim graphic, which distinguishes it from the detach button of the individual panes.

**To reattach Timing Analysis View:**

- Choose **Window >** **Attach Window**.

**See Also** ▶"Timing Analysis View Main Window Tabs" on page 213

▶ "Cross-Probing from Timing Analysis View" on page 219

# Cross-Probing from Timing Analysis View

In the Radiant software, many views contain corresponding cross-reference data that allow you to "jump" between views. This cross-probing feature allows you to gain new perspectives on your layout and connectivity for particular design elements that are critical for meeting your constraints. This can be very useful in guiding your design run iterations toward specific goals.

Timing Analysis View enables cross-probing within the view, and it allows you to cross-probe selected elements to Floorplan View. You can also use this feature to group components along an individual signal. See "Grouping Components Along an Individual Signal" on page 157.

You can cross-probe an unconstrained path as well as a path that is constrained by a defined constraint.

**To cross probe from Timing Analysis View:**

1. Select a constraint in any of the four tabs except General Information.
2. After the Path Summary pane opens, click on any parameter in this pane to open the Path Detail and Data / Clock Paths panes.
3. Right-click on a parameter from the Data / Clock Paths and select **Show in Floorplan / Physical View**.

**See Also** ▶"Timing Analysis View Main Window Tabs" on page 213

# Analyzing Power Consumption

Included with the Radiant software is Power Calculator, which 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 power consumption. It reports both static and dynamic estimated power consumption.

Power Calculator allows you to import frequency and activity factors from the post-PAR simulation file (.vcd file). After the design information is added, Power Calculator provides accurate power consumption analysis for the design.

Power Calculator provides two modes for reporting power consumption:

- Estimation mode: Used before completing the design;
- Calculation mode: Based on the physical netlist file (.udb) after placement and routing.

You can open Power Calculator from within the Radiant software or as a stand-alone tool from the Windows Start menu. Either method provides both estimation mode and calculation mode functionality. A stand-alone Power Calculator is also available, which allows you to estimate power consumption without having the Radiant software installed.

**Calculation Mode** Within the Radiant software environment, Power Calculator calculates power consumption based on the project's files, including device information. This information is automatically extracted when you open Power Calculator from within the Radiant software. In the stand-alone Power Calculator, power consumption is calculated based on a selected native circuit description (.udb) file or power calculator file (.pcf)

**Estimation Mode** Within the stand-alone Power Calculator, the Startup Wizard enables you to estimate power based on a selected device, and it gives you the option of including a template of resource settings. Within the

Radiant software environment, Power Calculator estimates power for an unrouted design. After the design is routed, it enables you to change the device family or other data and obtain the estimated power consumption.

**See Also** "Software Mode" on page 225

- "Starting Power Calculator from the Radiant Software" on page 221
- "Starting Power Calculator as a Stand-Alone Tool" on page 221
- "Power Analysis Design Flow" on page 223

# Starting Power Calculator from the Radiant Software

Power Calculator is available from the Radiant software as soon as a project is opened.

**To start Power Calculator from the Radiant software Tools menu or toolbar:**

1. Open a project or create a new one.
2. Choose **Tools > Power Calculator** or click the  button on the toolbar.

   Power Calculator opens in estimation mode or calculation mode, depending on the design stage, and displays the Power Summary page.

You can also start Power Calculator by creating or opening a new Power Calculator file (.pcf):

- Use the File menu or the Analysis Files folder to create a new Power Calculator file (.pcf).

  Power Calculator opens and loads the newly created .pcf file.
- Use the File menu or the Analysis Files folder to open an existing .pcf file.

  Power Calculator opens in estimation mode or calculation mode, depending on the status of the selected .pcf file.

**See Also** "Running Power Calculator from the Tcl Console" on page 222

- "Saving a Power Calculator File" on page 234
- "Adding Power Calculator Files to the Analysis Files Folder" on page 237
- "Setting a Power Calculator File as the Active Analysis File" on page 238

# Starting Power Calculator as a Stand-Alone Tool

Power Calculator is available as a stand-alone tool from the Windows Start menu. This allows you to work with a Power Calculator project, in calculation mode or in estimation mode, without opening the Radiant software. The

Startup Wizard enables you to create a new Power Calculator project, based on a selected device or a processed design, or to open an existing Power Calculator project file (.pcf).

**To start Power Calculator as a stand-alone tool:**

1. Choose **Start > All Programs > Lattice Radiant Software > Accessories > Power Calculator**.
2. After the Power Calculator Startup Wizard appears, do one of the following and click **OK**:
   - To calculate power consumption based on a mapped or placed and routed design, select **Calculate power with design (udb)**.

     Click the UDB File Browse button to navigate to the .udb file.

     After you select the .udb file, the File Name and File Directory boxes are automatically filled in by the Startup Wizard. If you want to give the Power Calculator project file a different name than the .udb file name, type the name in the File Name text box.

   - To open an existing Power Calculator project, select **Open existing PCF file**.

     Click the Browse button to navigate to the .pcf file.

     When you click OK, Power Calculator opens in estimation mode or calculation mode, depending on the status of the .pcf file.

**See Also** ▶"Running Power Calculator from the Tcl Console" on page 222

- "Saving a Power Calculator File" on page 234

# Running Power Calculator from the Tcl Console

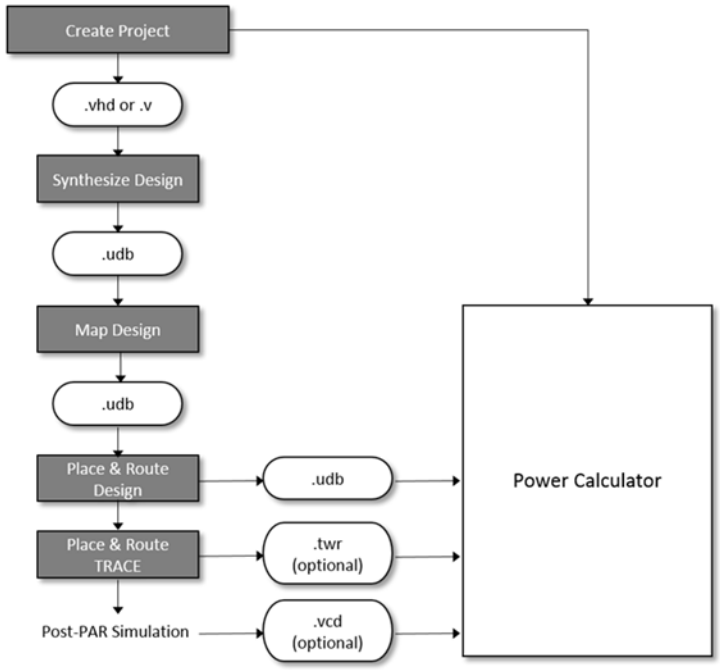
The Radiant software's Tcl console window enables you to use Tcl commands to perform many power analysis functions. For a complete list and descriptions of Power Calculator Tcl commands, see "Tcl Command Reference Guide" on page 598 in the Tcl Command Reference Guide.

**See Also** ▶"Tcl Command Reference Guide" on page 598

# Power Analysis Design Flow

Power Calculator supports all Lattice FPGA devices. The design flow involved in using Power Calculator in the Radiant software is as follows.

**Figure 35: Power Calculator Design Flow**




**See Also** ▶"Inputs" on page 223

▶ "Outputs" on page 224

# Inputs

When you first launch Power Calculator, it displays information from your design project and default resource information based on the targeted device. When you first launch Power Calculator from the Radiant software, it displays information from your design project. For an unrouted design, it shows default resource information based on the targeted device. For a routed design, it extracts information from the placed and routed .udb file.

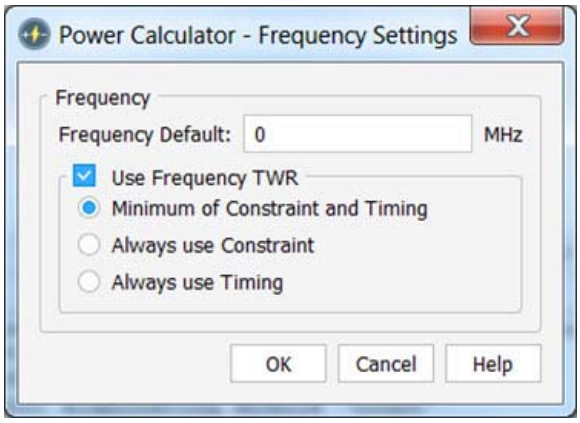
Additionally, Power Calculator accepts the following as optional input.

- Value change dump file, <*project_name*>.vcd, which is an ASCII file containing activity and frequency information. Its format is specified by the IEEE 1364 standard. It should be in the format of gate-level simulation and match the design. The .vcd file preserves waveform information that can be displayed in third-party tools such as Active-HDL.

If you provide a post-routed simulation .vcd file, Power Calculator looks up the clock signals in the .vcd file and compares them to each clock in the Power Calculator pages. If the clock names match, Power Calculator takes the frequency data from the .vcd file and populates the frequency columns, the activity factor (AF(%)) columns, or both, in the pages that contain these columns.

- Timing report file, <*project_name*>.twr, is an ASCII file containing activity and frequency information. The type of frequency and setting can be imported from the timing report file as shown in Figure 36.

**Figure 36: Frequency Settings**




**See Also** ▶"Importing a Value Change Dump (.vcd) File" on page 244

- "Changing the Global Default Activity Factor" on page 243

# Outputs

Power Calculator generates power calculation results in tabular format from information extracted from the design project. All of this information, when saved, is kept in the project's Power Calculator file (.pcf).

Power Calculator also generates a power calculation report, which can be viewed in HTML or text format. It also generates power graphs that show how power consumption is affected with varying voltage, temperature, and clock frequency.

**See Also** ▶"Saving a Power Calculator File" on page 234

- "Generating Power Graphs" on page 252
- "Viewing the Power Calculator Report" on page 254

# Static and Dynamic Power Consumption

Power Calculator reports the dynamic and static portion of the power dissipation. Power refers to the power consumed by the design. It is based on the extracted data from a placed and routed design file (.udb) or on the estimation information that you provide.

The dynamic portion is the power consumed by the used resources while they are switching. The power dissipation of the dynamic portion is directly proportional to the frequency at which the resource is running and the number of resource units used.

The static portion of power consumption is the total power consumed by the used and unused resources.

# Activity Factor Calculation

To calculate the power consumption for the routing interconnect, logic, and the read/write ports in an embedded block, Power Calculator requires a frequency and an activity factor percentage. The activity factor percentage is the percentage of time that a registered output node is active relative to a specified clock. Most of the resources associated with a clock domain run or toggle at a percentage of the frequency at which the clock runs.

The frequency appears as the “Freq. (MHz)” column on most pages. The activity factor percentage appears as the “AF (%)” column.

**See Also** ▶“Importing a Value Change Dump (.vcd) File” on page 244

- “Changing the Global Default Activity Factor” on page 243
- “Changing the Global Default Frequency Setting” on page 244

# Power Calculator Window Features

Power Calculator’s main window displays the software mode being used and the currently selected page of power consumption information. When you first open Power Calculator, the Power Summary page appears by default. All other pages of information for the device are made available from the tabs arranged at the bottom of the main window. This section describes these features and the color coding of cells.

## Software Mode

The Software Mode in the top right corner of the Power Calculator window indicates whether Power Calculator is running in estimation or calculation mode. This field is read-only.

**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. It is useful for “what if” analysis. You must supply the frequency, activity factor, and voltage.

**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 .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.

**Reverting to Estimation Mode** Power Calculator will revert to estimation mode from calculation mode in the following circumstances:

- If you start using Power Calculator in calculation mode and change the data in any cell other than AF (%), Freq., V., Dynamic Power Multiplier, Ambient Temperature, Performance Grade, Operating Condition, or Process Type, Power Calculator will automatically revert to estimation mode.
- Rerunning a process before Place & Route will change the software mode if you are working with the unsaved “untitled” temporary power calculator file. For example, if you rerun Map Design while in calculation mode using the “untitled” file, the tool will revert to estimation mode. If you are working with a saved .pcf file, the software mode will not change when you rerun a process.

**Reverting to Calculation Mode** For many types of changes, Power Calculator enables you to revert to calculation mode from estimation mode, if you have not yet saved the changes to the .pcf file. See “Reverting to Calculation Mode” on page 243 for more information.

# Power Summary

Power Summary provides an overview of power consumption conditions. It is the first page that opens when you run Power Calculator. The Power Summary enables you to change the targeted device, operating conditions, voltage, and other basic parameters. Updated estimates of power consumption are then displayed based on these changes.

**Device** The Device section enables you to select a device family, package, part name, performance grade, and operating conditions. It displays power information based on these selections.

You can specify one of the following operating conditions, depending on the device. As this is device dependent, please refer to the device data sheet for more accuracy.

- Industrial – Devices are rated at 105 degrees Celsius.
- Commercial – Devices are rated at 85 degrees Celsius.
- Automotive – Devices are rated at 125 degrees Celsius.

**Device Power Parameters Section** The Device Power Parameters section contains information pertaining to the device process conditions and power model status.

The process type specifies the corners or conditions under which the device was manufactured. It can be one of the following:

- Typical – to reflect the typical amount of current consumed by the circuit.
- Worst – to reflect the maximum amount of current consumed by the circuit.

The Power File Revision displays the status of the model:

- Advanced – The power file has all the constants, functions, formulas, and defaults. The constants are based on silicon simulation data without extracted layout parameters. The status of this file is “advanced.”
- Preliminary – The power file has all the constants, functions, formulas, and defaults. The constants are based on nominal silicon characterization data rather than simulation data. The status of this file is “preliminary.”
- Final/Production – The power file has all the constants, functions, formulas, and defaults. The constants are based on silicon characterization data that includes characterization of corner lots and conditions rather than simulation data. The status of this file is “production.”

**Environment Section** The Environment section displays information about the operating temperature of the device. The Thermal Profile button enables you to choose the thermal impedance model for the power-consumption or current-consumption calculation. The Ambient Temperature cell allows you to specify the surroundings at which the device is expected to operate, in degrees Celsius. Temperature values must be between -40 and +125 degrees Celsius.

Effective Theta-JA specifies the cumulative thermal impedances of a particular system. This figure is used in calculating the junction temperature (Tj) of a die in a particular environment according to the following formula:

Tj = *power* * *theta_effective* + *ambient_temperature*

Junction Temperature specifies the temperature of the device within the device package, in degrees Celsius. You can adjust the junction temperature by using the models in the Thermal Profile dialog box. If the calculated value in the Junction Temperature box is either larger than 125 degrees Celsius or larger than the maximum junction temperature allowed for a particular device and operating conditions, a red text appears in the Junction Temperature box.

Maximum Safe Ambient specifies the maximum safe operating temperature for a die. If this temperature is exceeded, the semiconductor physics of the silicon change, so the die can operate erratically. The life of the device is also shortened.

**Voltage/Dynamic Power Multiplier** This section contains information about the estimated power or current consumption by power supply. The Dynamic Power Multiplier specifies the derating factor for the dynamic portion of the power consumption. Power Calculator does not include derating for dynamic power. This factor enables you to change the number by the derating factor that you want to apply. The derating factor specified must be between 0.5 and 2. The default is 1.0.

**Current by Power Supply** This section displays the static, dynamic and total current consumption in amperes.

**Power by Power Supply** This section displays the static, dynamic, and total power consumption in watts.

**Power by Block** This section displays the total power consumption by the different types of blocks, in watts.

# Power Calculator Pages

Each time Power Calculator opens, it displays the Power Summary, which shows the targeted device, operating conditions, voltage, and other basic information. Additional pages are available from the tabs arranged at the bottom of the window. Except for Graph and Report, each of these pages allows you to view, edit, and add elements. The number and types of pages that are available depends on the selected device.

### Pages Available for Most Device Families

**Block RAM** The Block RAM page displays the power consumed by the embedded block RAM (EBR) in the design and the factors that affect it. Power-consumption calculation for the Block RAM page requires the frequency and activity factor per clock domain.

**Misc** The Misc page includes any non-generic IP, such as oscillators, i2c, and spi. These IPs do not have their own page due to negligible power contribution.

**PLL** The PLL page displays the estimated power consumed by the phase locked loops in the design and the factors that influence it. The power-consumption calculation is based on the input frequency, the number of PLLs in the design, and internal feedback architecture of PLL in the device.

This allows you to use the Power Option Controller to place selected blocks in standby mode.

The PLL page is available for all devices.

**Power Matrix** The Power Matrix page shows the amount of power pulled by each component in the design from multiple power sources. Two tabs are provided, allowing you to view the current usage (A) and power usage (W) of each type of component. Power Calculator enables you to view the power matrix as an HTML report. It also allows you to generate a .csv file of the matrix that you can open in a spreadsheet application.

**Logic Block** The Logic Block page displays the estimated power consumed by the logic in the design and the factors that affect it. Power-consumption calculation in the Logic section requires both the frequency and an activity factor per clock domain. Power calculation is also based on the specified number of LUTs, distributed RAMs, ripple-carry logic circuits, and registers driven by the clock.

**Clocks** The Clocks page displays the estimated power consumed by the clocks in the design and the factors that affect it. Dynamic power calculation is based on the frequency of each clock.

The Clocks page only reports clocks that go to the clock tree, such as Primary Clock, Secondary Clock and Edge Clock.

**I/O** The I/O page displays the estimated power consumed by the I/Os in the design and the factors that affect it. The power-consumption calculation for this page requires the frequency, activity factor, and number of inputs or outputs per clock domain. For bidirectional signals, it requires the number of bidirectional I/Os, input and output frequency, and input and output activity factor per clock domain. If you use an .udb file, Power Calculator extracts the information for the I/O page directly from the .udb file. If you do not use an .udb file, Power Calculator assigns the default resource usage values according to the design's family.

For low-power devices, the I/O page includes the following additional columns for controlling power options:

- Power Guard (PG) – Power Guard can be turned on to enable the blocking of signals at the input buffers. Power Guard's biggest impact is in standby mode when it can be used to switch off clock inputs that are distributed using general routing resources. After you turn Power Guard on, you can use the Power Controller to enable it.
- Allow InRD Shut-off – Selecting **Yes** in this column in the Bank Voltage section allows the disabling of the dynamic referenced and differential input buffers.

Allow LVDSO Shut-off – Selecting **Yes** in this column in the Bank Voltage section allows the LVDS output buffer driver to be turned off.

**Graph** The Graph page displays three types of graphs:

- Power vs. VCC Supply Voltage
- Power vs. Ambient Temperature

Each graph displays two plots—typical and worst case. Use the Edit > Graph Settings command to change the graphs to be displayed. The dialog box enables you to specify the range, step, X axis, and Y axis of each graph. See "Generating Power Graphs" on page 252 for more information.

To prevent any unnecessary calculations and additional processing time, graphs are only generated when you select the Graph tab. During the graph generation time, you cannot change tabs and must wait for the graph calculations to finish before performing any other action. If you switch tabs

and change any information that will alter the power, the graphs will be regenerated when you next select the Graph tab.

**Report** The Report page contains a summary of the estimated power-consumption or current-consumption data calculated by Power Calculator. Information is taken from the Power Summary and from each page of the Power Calculator user interface.

In the Power Model section, the Status can be Preliminary, Advanced, or Final.

- Preliminary – The power file has all the constants, functions, formulas, and defaults. The constants are based on nominal silicon characterization data rather than simulation data.
- Advanced – The power file has all the constants functions, formulas, and defaults. The constants are based on silicon simulation data without extracted layout parameters.
- Final/Production – The power file has all the constants, functions, formulas, and defaults. The constants are based on silicon characterization data that includes characterization of corner lots and conditions rather than simulation data.

The report is available in both text (ASCII) and HTML. It is updated each time you make a change to any of the data in the editable cells.

### Pages Dependent on Selected Device/ Device Architecture

**LED** The LED page allows you to generate the constant current sources. The signal name for Input is “EN” and “LEDPU” for Output. Default Signal Value for unconnected port: Input is “0”

Input EN : Logic “0”

The LED page is available for all devices.

**DSP** The DSP page displays the estimated power consumed by the digital signal processors in the design and the factors that affect it. The power-consumption calculation for this page requires the frequency, activity factor, and the type and number of DSPs driven by each clock.

**SRAM** The SRAM page displays the power consumed by the single port RAM used by the design and the factors that affect it. Power-consumption calculation for the SRAM page requires the frequency and activity factor per clock domain.

**I/O Termination** The I/O Termination page enables you to provide information about external terminations for the I/Os. You can specify the average equivalent thevenin resistive load in ohms (Rth) and the equivalent thevenin voltage in volts (Vth). Power-consumption calculation is based on the power consumed by the external termination that you provide. In certain cases, the termination can also be internal to the device.

## TColor Coding of Cells

The background colors of the cells on Power Calculator pages have the following significance:

- White – The cell is editable. When you edit the contents of this type of cell, the software mode does not change. If the software is in calculation mode, it will remain in calculation mode. If the software is in estimation mode, it will remain in estimation mode.
- Green – The cell is read-only or contains output from the software.
- Lite Yellow – The cell contains data extracted from a design file, such as an .udb file. If you enter data into this type of cell and the software is in calculation mode, it will change to estimation mode unless you enter data into the Performance Grade, Operating Conditions, and Process Type boxes. If the software is in estimation mode, it will remain in estimation mode.
- Red – The calculated value in the Junction Temperature box is either larger than 125 degrees Celsius or larger than the maximum junction temperature allowed for a particular device and operating conditions. The Junction Temperature box is the only cell that can display red text.

The font colors in the cells on Power Calculator pages have the following significance:

- Blue – Indicates default values.
- Grey out – Indicates values that cannot be edited on the I/O page. Since the I/O page has columns for inputs, outputs, and bidirectionals, red font prevents you from altering an I/O that is not valid. For example, if the I/O type belongs only to an I/O input, the cell in the # of Inputs column would display a value in black font, indicating that it is editable. The cells in the # of Outputs and the # of Bidi columns, however, would display values in gray font to indicate that they cannot be edited.
- Black – Indicates all other text.

# Working with Power Calculator Files

When you first open Power Calculator for a design project, it creates a temporary file that appears in the title bar as “Untitled.” This file contains default information, based on the device, or information extracted from the .udb file. When you enter data into Power Calculator pages and save the changes, the information gets stored in a Power Calculator file (.pcf). You can create a new Power Calculator file (.pcf) by saving the “Untitled” file. Or you can create a new .pcf file from the File menu before or after Power Calculator is opened.

When you use a .pcf file, rather than the “Untitled” temporary file, Power Calculator maintains the information it has already extracted from the .udb file. The saved .pcf file will not get overwritten when you rerun Map or Place & Route Design, and the software mode will not change.

This section describes how to create new Power Calculator files, how to work with multiple existing .pcf files, and how to activate a .pcf file and load it into Power Calculator. It also shows how to import a value change dump file (.vcd) for power calculation. This section describes how to create new Power Calculator project files and how to import a value change dump file (.vcd) and a timing report file (.twr) for power estimation.

# Creating a New Power Calculator File in the Radiant Software

You can create a new .pcf from the File menu or from the Analysis Files folder pop-up menu. You can do this before or after opening Power Calculator.

**To create a new .pcf file:**

1. In the Radiant software, do one of the following to open the New file dialog box:
   - choose **File > New > File**
   - Press **Ctrl+N**.
   - Right-click the Analysis Files folder in the File List pane and choose **Add > New File** from the pop-up menu.
2. In the New file dialog box, select **Power Calculator Files** from the Source Files list.
3. Type a name for the new .pcf file in the name box.
4. In the Location box, enter the path and name of the directory where the Power Calculator file (.pcf) is to be stored. You can use the Browse button to navigate to the desired directory. By default, the file is stored in the current project folder.

   The “Add to Implementation” option adds the new .pcf file to the Analysis files folder and is selected by default. If you do not wish the .pcf file to be added to the Analysis files folder for the current project, clear the “Add to Implementation” option. You cannot clear this option if you are using the Analysis Files folder to add a new file.
5. If your project contains more than one design implementation, select the desired implementation from the drop-down menu. By default, the active implementation is already selected.
6. Click **New** to create the .pcf file in the selected directory.

   If Power Calculator is already open and contains unsaved changes to the current .pcf file, the Confirm dialog box will appear. Click **Yes** if you want to save the changes or **No** to discard them.

   Power Calculator opens, if it is not open already. This might take a few seconds. The new .pcf file is loaded into Power Calculator and its name is displayed in the title bar. The .pcf file is added to the Analysis Files folder where its name is highlighted in bold type, indicating that it is set as the active .pcf file for the current implementation. Because the .pcf file is set as active, it will be loaded automatically when you close and reopen

Power Calculator. When you rerun a process in the Radiant software, the .pcf file will not get overwritten.

**Note**

Each time you create a new Power Calculator file through the File menu or the Analysis Files pop-up menu, the new .pcf file is automatically set as the active one for the current implementation unless you have cleared the “Add to Implementation” option. To make the .pcf file inactive, right-click the file name in the Analysis Files folder and choose **Set as Inactive**.

**See Also** ▶“Inputs” on page 223

- “Saving a Power Calculator File” on page 234
- “Outputs” on page 224
- “Opening an Existing Power Calculator File in the Radiant Software” on page 235

# Creating a New Power Calculator Project File

After you have opened Power Calculator, you can create a new Power Calculator Project File (.pcf) from the File menu.

**To create a new .pcf file:**

1. In Power Calculator, choose **File > New File**.
2. In the New Project dialog box, type a file name in the File Name text box.
3. Click the **Browse** button and navigate to the desired directory for the new Power Calculator project file.
4. After you have selected the desired directory, click **OK** to return to the dialog box, and then click **OK**.

   Power Calculator loads the new project file.

**See Also** ▶Inputs

- Saving a Power Calculator Project File
- Outputs

# Saving a Power Calculator Project File

After entering data , an asterisk appears in the title bar to indicate that there are unsaved changes. The asterisk will remain until you save the .pcf file.

**To save changes to the Power Calculator file:**

- Press **Ctrl+S** or click  on the toolbar. Alternatively, choose **File > Save <file_name>.pcf.**

**To save a Power Calculator file to a different directory or file name:**

1. Choose **File > Save <file_name>.pcf As** to open the Save dialog box.
2. In the Save Power Calculator dialog box, navigate to the desired directory.
3. Type a name in the File Name box.
4. Click **Save**.

# Creating a New Power Calculator File in the Stand-Alone Power Calculator

If you have started Power Calculator as a stand-alone tool, you can create a new Power Calculator project file (.pcf) from the File menu.

**To create a new .pcf file from the stand-alone Power Calculator:**

1. From the stand-alone Power Calculator, choose **File > New File**.
2. In the Power Calculator – New Project dialog box, type a name for the file and browse to the desired directory.
3. To use an existing .udb file for the project, browse to the location of the .udb file. Otherwise, leave the .udb box empty.
4. Click **OK**.

   If you selected an .udb file in Step 3, Power Calculator loads the new project in calculation mode. If you did not select an .udb file, Power Calculator loads the new project in estimation mode.

**See Also** ▶"Inputs" on page 223

- "Saving a Power Calculator File" on page 234
- "Opening an Existing Power Calculator File in the Stand-Alone Power Calculator" on page 236

# Saving a Power Calculator File

When you enter data into Power Calculator, an asterisk appears in both the title bar and the Power Calculator tab to indicate that there are unsaved changes. If you have not yet created a Power Calculator file (.pcf) for your project, or if no .pcf file has been set as the active file, "Untitled" will appear in the title bar. The "Untitled" file is a temporary file that contains default settings based on the device. To add power analysis changes to your project, you must save the "Untitled" as a .pcf file, open an existing .pcf file, or create a new one. Afterwards, you can simply click the Save button to write any changes to the .pcf file. You can also save an existing .pcf file to a different directory or file name.

**To save the Untitled file:**

1. Choose **File > Save Untitled As**, and in the dialog box, navigate to the desired directory.
2. Type a name in the file name box and click **Save**.

   The information entered in your project is saved in the .pcf file, and the .pcf file is automatically added to the Analysis Files folder in the File List pane.

**To save changes to an existing Power Calculator file:**

- Choose **File > Save** or press **Ctrl+S** or click  .

**To save a Power Calculator file to a different directory or file name:**

1. Choose **File > Save <file_name>.pcf As** to open the Save dialog box.
2. In the Save In box, navigate to the directory in which to save the .pcf file.
3. Type a different name in the File Name box.
4. In the Files of Type box, select **Power Calculator File (.pcf)**.
5. Click **Save**.

   The saved .pcf file is added to the project's Analysis Files folder in the File List pane.

**See Also**

- "Adding Power Calculator Files to the Analysis Files Folder" on page 237
- "Setting a Power Calculator File as the Active Analysis File" on page 238
- "Outputs" on page 224


# Opening an Existing Power Calculator File in the Radiant Software

You can open an existing Power Calculator (.pcf) file in the Radiant software before or after opening Power Calculator. When you open a .pcf file in the Radiant software, the design information must match the information in the current project. You can open an existing .pcf file from the File menu, or from the Analysis Files folder.

**To open an existing .pcf file from the File menu:**

1. Choose **File > Open > File** or press **Ctrl+O**.
2. In the Open File dialog box, select **Power Calculator Files (.pcf)** from the Files of Type drop-down menu. If Power Calculator is already open, this will already be selected as the default file type.
3. Navigate to the directory that contains the desired .pcf file, select the file, and click **Open**.

Power Calculator opens automatically, if it is not already open. The Power Summary page is displayed and the information from the selected .pcf file is loaded. The name of the .pcf file appears in the title bar.

**To open a recently opened .pcf file:**

- Choose **File > Recent Files > *filename*** from the list of the four most recently opened files.

**Note**

When you open a .pcf file from the File menu, it is not automatically added to the Analysis Files folder. You must add it to the folder manually.

**To open an existing .pcf file from the Analysis Files directory:**

1. In the Radiant software, select the File List tab in the pane on the left.
2. Expand the Analysis Files folder.

   You can open the active .pcf file, if it is not already open, or one that is not active. An active .pcf file appears in bold type.
3. Double-click the name of the .pcf file that you want to open. Optionally, right-click the file name and choose **Open**.

   The information from the .pcf file is loaded into Power Calculator, and the name of the file appears in the title bar.

**See Also**

- "Adding Power Calculator Files to the Analysis Files Folder" on page 237
- "Setting a Power Calculator File as the Active Analysis File" on page 238


# Opening an Existing Power Calculator File in the Stand-Alone Power Calculator

If you have opened Power Calculator as a stand-alone tool, you can open an existing Power Calculator file (.pcf) from the File menu.

**To open an existing .pcf file from the stand-alone Power Calculator:**

1. Choose **File > Open File**.
2. Navigate to the desired .pcf file and click **Open**.

   If you have unsaved changes in the currently loaded .pcf file, a Confirm message box will appear. Click **Yes** if you want to save the changes. Otherwise, click **No**.

   If an .udb file is associated with the selected .pcf file, Power Calculator loads the project in calculation mode. If no .udb file is associated with the .pcf file, Power Calculator loads the new project in estimation mode.

**See Also** ▶Saving a Power Calculator Project File

- Creating a New Power Calculator Project File
- "Saving a Power Calculator File" on page 234
- "Creating a New Power Calculator File in the Stand-Alone Power Calculator" on page 234

# Adding Power Calculator Files to the Analysis Files Folder

The Radiant software enables you to add existing Power Calculator files (.pcf) to the Analysis Files folder. This gives you easy access to the .pcf files in your project and enables you to designate a .pcf file as the active one for a design implementation. When a .pcf file has been set as the active analysis file, it will be loaded into Power Calculator automatically when you close Power Calculator and reopen it.

You can add a .pcf file to the Analysis Files folder before or after starting Power Calculator.

**To add an existing .pcf file to the Analysis Files directory:**

1. In the Radiant software, select the File List tab in the pane on the left.
2. Right-click the Analysis Files folder and choose **Add > Existing File** from the pop-up menu.
3. In the dialog box, select **Analysis Files (*.pcf)** in the Files of Type box.
4. Navigate to the directory that contains the desired .pcf file, select it, and click **Add**.

   The file is added to the Analysis Files folder. You can double-click the .pcf file to open it in Power Calculator.

**Note**

When you add a .pcf file to the Analysis Files folder, it is not automatically set as the Active .pcf file for the implementation. You must set it as the active file manually.

**See Also** ▶"Removing Power Calculator Files from the Analysis Files Folder" on page 238

- "Creating a New Power Calculator File in the Radiant Software" on page 232
- "Setting a Power Calculator File as the Active Analysis File" on page 238

# Removing Power Calculator Files from the Analysis Files Folder

There may be times when you want to remove files from the Analysis Files folder to make room for others. When you remove a Power Calculator File (.pcf) from the Analysis Files folder, it is not deleted from your project. You can always add it back later.

**To remove a .pcf file from the Analysis Files folder:**

- Right-click the file name and choose **Remove**.

  The file is removed from the Analysis Files list, but it remains in your project directory.

# Setting a Power Calculator File as the Active Analysis File

In order for Power Calculator to load a specific Power Calculator File (.pcf) each time it opens, you must designate the .pcf file as the active one for the implementation. This is done automatically when you create a new .pcf file. If no .pcf file has been set as the active one, Power Calculator will extract information from the placed and routed .udb file when you open Power Calculator. If no .udb file is available, it will display default resource information based on the targeted device.

**To set a .pcf file as the active analysis file:**

1. Make sure that you have added the desired .pcf file to the Analysis Files folder.
2. Right-click the desired .pcf file in the Analysis Files folder and choose **Set as Active PCF**.

   The Radiant software displays the .pcf file name in bold type, indicating that it is the active power analysis file for the current design implementation. The Radiant software does not automatically load the newly activated file into Power Calculator if a different .pcf file is already open.
3. To open the active .pcf file, if a previous file is already open, double-click the activated .pcf file name in the Analysis Files folder; or right-click it and choose **Open**.

   When you close and reopen Power Calculator for the current design project, the active .pcf file will be loaded.

## Changing an Active Power Calculator File to Inactive

It is not required that a Power Calculator File (.pcf) be set as the active one for a design implementation. You can always open a .pcf file manually from the File menu or from the Analysis Files folder.

**To change an active Power Calculator File to inactive:**

- Right-click the name of the active .pcf file in the Analysis File folder and choose **Set as Inactive** from the pop-up menu.

  When you close and reopen Power Calculator, it displays default information or information from the routed .udb file. “Untitled” appears in the title bar.

# Entering Data

When you have a design open in the Radiant software, Power Calculator extracts information such as device, package, part, performance grade, and operating conditions. You can modify the device settings and the editable cells on any page. If Power Calculator is in calculation mode when you make any change other than Activity Factor, Frequency, Voltage, Dynamic Power Multiplier, Ambient Temperature, Performance Grade, Operating Condition, or Process Type, it will revert to estimation mode. You can revert to calculation mode after making many types of changes, if they have not yet been saved to the .pcf file.

Power Calculator allows you to enter data directly in the editable cells. It also enables you to make global changes to frequency and activity factors by changing the default setting in the Frequency Settings and Activity Factors Settings dialog boxes. You can use a simulation file to populate both Frequency and AF (activity factor) cells.

**See Also**

- “Software Mode” on page 225
- “Reverting to Calculation Mode” on page 243
- “Power Calculator Pages” on page 228


## Editing Cells

Power Calculator includes built-in design rule checks. It automatically checks values that you enter into editable cells to ensure that they do not violate design rules. If you attempt to enter an inappropriate value in the Type, # I/P, # O/P, or # Bidi cell in the I/O page, Power Calculator will block the invalid value and display the previous value in the cell.

Power Calculator also provides tool tips that display the valid range of values for an editable cell. To ensure that the value you are entering is a valid one, hold your mouse over the cell to view the tool tip.

Most cells on Power Calculator pages are editable text cells that enable you to type a modified value. Others cells, such as the Device and Power parameters sections of the Power Summary page, contain visible drop-down menus for making a selection. Still others, such as those in the Type column on the I/O page, contain hidden drop-down menus that become visible when you double-click a cell.

**To edit a cell:**

- Depending on the type of cell you are editing, do one of the following:
  - Double-click the editable cell, type a new value, and then press **Enter** or click anywhere outside the cell.
  - Select a value from the visible drop-down list.
  - Double-click the cell and select a value from the drop-down list that appears.

  Power Calculator calculates the results automatically and displays them. It also updates the Report page.

**See Also** "Power Calculator Pages" on page 228

- "TColor Coding of Cells" on page 231
- "Cutting and Pasting Cell Contents" on page 242
- "Copying and Pasting Cell Contents" on page 242
- "Changing Values Automatically" on page 242

# Editing Pages

You can change the settings and values on any Power Calculator page and, if desired, save the results to a separate Power Calculator File (.pcf) in the Analysis Files folder.

**To edit Power Calculator pages:**

1. On the Power Summary page, modify any settings in the Device section as desired.

   When you select a different device, Power Calculator compares the design's requirements against the available resources in the selected device. If the selected device is not suitable for the design—for example, if the number of LUTs in the design exceeds those available in the device—Power Calculator will generate an error message and not allow the change. To proceed with the change, you would need to reduce the design size to fit the smaller device.

2. In the Device Power Parameters section of the Power Summary page, set the Process Type option, which specifies the process corners or conditions under which the device was manufactured. It can be one of the following:
   - Typical – specifies typical conditions to reflect the typical amount of current consumed by the circuit.

- Worst – specifies fast conditions to reflect the maximum amount of current consumed by the circuit.

3. In the Environment section of the Power Summary page, do the following:
   - Click **Thermal Profile** to select a thermal impedance model or enter your own Effective Theta-JA value.
   - Change the ambient temperature, as desired.

   See "Controlling Operating Temperature" on page 247
4. In the Voltage/Dynamic Power Multiplier section, enter new values, as desired, for voltage and DPM.

   The voltage is the estimated power consumption by power supply. DPM is the derating factor for the dynamic portion of the power consumption.
5. Select other tabs and enter values into the editable cells of the other Power Calculator pages. The number and types of pages varies according to the device family.
6. Save your changes.

**See Also** "Power Calculator Pages" on page 228

- Saving a Power Calculator Project File
- Saving a Power Calculator File
- "Adding Power Calculator Files to the Analysis Files Folder" on page 237
- "Setting a Power Calculator File as the Active Analysis File" on page 238
- "Editing Cells" on page 239

# Adding and Deleting Clock Rows

You can easily add and delete clock rows on the Power Calculator pages.

**To add a clock row to a page:**

1. Right-click inside the desired table and choose **Add Row** from the pop-up menu.
2. Enter the appropriate data in the cells that have a white or light yellow background. Some columns with a light yellow background, such as Type, offer drop-down menus from which you can select settings.

**To delete a clock row from a page:**

- Right-click in the row that you want to delete and choose **Remove Row**.

**See Also** "Editing Cells" on page 239

# Cutting and Pasting Cell Contents

You can cut the contents of a cell in a clock row and paste them in a cell in the same row or another row.

**To cut and paste cell contents:**

1. Double-click the desired cell to select its contents, and then right-click.
2. From the pop-up menu, select **Cut**. Alternatively, you can press **Ctrl+x**.

   You cannot cut the contents of any cells or columns that include a drop-down menu or text that is read-only.
3. Double-click the cell into which you want to paste the contents that you have cut, and then right-click.
4. From the pop-up menu, select **Paste**. Alternatively, you can press **Ctrl+v**.

**See Also** ▶"Editing Cells" on page 239

▶ "Changing Values Automatically" on page 242

# Copying and Pasting Cell Contents

You can copy the contents of a cell in a clock row to a cell in the same row or another row.

**To copy and paste the contents of a cell:**

1. Double-click the desired cell to select its contents, and then right-click.
2. From the pop-up menu, select **Copy**. Alternatively, you can press **Ctrl+c**.
3. Double-click the cell into which you want to paste the contents that you have copied, and then right-click.
4. From the pop-up menu, select **Paste**. Alternatively, you can press **Ctrl+v**.

**See Also** ▶"Editing Cells" on page 239

▶ "Changing Values Automatically" on page 242

# Changing Values Automatically

When you change values on any of the pages, Power Calculator recalculates the results automatically. For example, when you change the frequency of a clock in one cell and press Enter, Power Calculator automatically changes the frequency for that clock in all Frequency cells.

You can also use the Activity Factor Settings and Frequency Settings dialog boxes to make changes to all frequency and activity factor cells.

**See Also** ▶"Changing the Global Default Activity Factor" on page 243

▶ "Changing the Global Default Frequency Setting" on page 244

- "Importing a Value Change Dump (.vcd) File" on page 244

# Reverting to Calculation Mode

After you have made changes that causes the software to run in estimation mode, Power Calculator allows you to revert to calculation mode, under the following circumstances:

- The changes you made have not been saved to the .pcf file.

**To revert to calculation mode from estimation mode:**

1. Choose **Edit > Revert to Calculation Mode**.
2. In the Confirm dialog box, click **Yes** to confirm that you want to discard all the changes that you made in estimation mode.

   Power Calculator removes all the changes you made and reverts to the settings in the .pcf file.

**See Also** "Software Mode" on page 225

# Changing the Global Default Activity Factor

Power Calculator automatically assigns a global default activity factor of 10 percent in the cells of the pages that display an activity factor, such as AF (%), or that use an activity factor in calculations, such as Input AF (%). These default values appear in blue font. You can use the Edit menu to globally change this default activity factor.

**To globally change the default activity factor:**

1. Choose **Edit > Activity Factor Settings**.
2. In the Power Calculator - Activity Factor Settings dialog box, enter the new activity factor in the **Activity Factor Default** text box.
3. Click **OK**.

All the default activity factors appearing in blue font are changed to the new activity factor. Power Calculator automatically saves the new default.

If you manually change an activity factor in only one cell, the font becomes black to indicate that it is not a default value.

You can use a .vcd file to populate the cells that display or use activity factors. The resulting values are not considered defaults and therefore appear in black font.

**See Also** "Activity Factor Calculation" on page 225

- "Importing a Value Change Dump (.vcd) File" on page 244

# Importing a Value Change Dump (.vcd) File

Power Calculator enables you to import a value change dump (.vcd) file of simulation results into your project. Normally, you would import a .vcd file only when you want the Frequency and AF (activity factor) cells on Power Calculator pages to be populated with frequency and activity factor data from the .vcd file.

To ensure that Power Calculator populates the Frequency and AF cells with the VCD information, make sure that you follow these requirements:

- The .vcd file should be in the format of gate-level simulation, and it should match the design.
- A post-PAR timing simulation netlist must be used for name matching. You cannot use the RTL design.
- A stimulus must be used in the simulator that actually toggles the signals you are interested in; otherwise, you will see no difference in Power Calculator after the VCD is read.

**To import a .vcd file into your project:**

1. Choose **Edit > Open Simulation File** to open the Power Calculator – Open Simulation File dialog box.
2. In the VCD File box, type or select the path and name of the .vcd file that you want to open.
3. In the Module Name in VCD box, specify the name of the module in the .vcd file from which to take the frequency and activity factor data.
4. Select the Case Sensitive option if the name of the .vcd file to be imported is case-sensitive.
5. Click **OK**.

   Power Calculator's Frequency and AF cells are now populated with the data from the .vcd file.

**See Also** "Changing the Global Default Frequency Setting" on page 244

# Changing the Global Default Frequency Setting

You can globally change the default frequency values for the clocks listed in the Power Calculator pages. These default values appear in blue font in the Freq. (MHz) columns.

You can change this global frequency value in one of two ways:

- Specify a value in the Frequency Settings dialog box.

- In the Frequency Settings dialog box, select **Use Frequency TWR** options to import frequencies from the timing report.

**Note**

The frequency of some clocks, such as those that have high dependency on user constraints and usage, cannot be imported by using **Use Frequency TWR** options. In cases where clock frequency cannot be imported, you must specify a value.

**To globally change the default frequency setting by specifying a value:**

1. Choose **Edit** > **Frequency Settings**.
2. In the Frequency Settings dialog box, enter the new frequency in the **Frequency Default** cell, in megahertz.

   The default is 0 megahertz.
3. Click **OK**.

**To globally change the default frequency setting by using values from the .twr file:**

1. Choose **Edit** > **Frequency Settings**.
2. In the Power Calculator - Frequency Settings dialog box, select one of the following options in the Frequency TWR box. For all of these options, make sure timing report contains the names of the clocks in the pages for which you want default frequency values.
   - Minimum of Constraint And Timing – Specifies that the default frequency in the Frequency (MHz) column of the Power Calculator pages be taken from the lesser of the constrained frequency or the actual frequency in timing report. The frequency is in megahertz.
   - Always Use Constraint– Specifies that the default frequency in the Frequency (MHz) column of the Power Calculator pages be taken from the constrained frequency in timing report. The frequency is in megahertz.
   - Always Use Timing– Specifies that the default frequency in the Frequency (MHz) column of the Power Calculator pages be taken from the actual frequency in timing report. The frequency is in megahertz.
3. Click **OK**.

Power Calculator now populates the Frequency cells of its pages with the frequency data from the timing report.

**See Also** ▶"Inputs" on page 223

# Estimating Resource Usage

Power Calculator allows you to specify an estimate of resources that the design will use, for the purpose of power analysis. The estimate can be based on design type or on component utilization. Based on your selections, Power Calculator immediately displays the number of resources that will be utilized for each type.

**To estimate resource usage based on design type:**

1. Choose **Edit > Resource Settings**.
2. In the dialog box, select **Specify Resource by Design Type**.
3. Select the design type from the drop-down menu.

   Power Calculator calculates the resource usage based on the design and displays the utilization in the bottom portion of the dialog box.
4. Click **OK**.

**To estimate resource usage based on component utilization:**

1. Choose **Edit > Resource Settings**.
2. In the dialog box, select **Specify Resource by Component Utilization**.
3. Do one or both of the following:
   - Select a Small, Medium, or Large option based on the design size.

     Power Calculator displays a percentage of Logic, I/O, and EBR based on your selection.
   - Select a percentage from the Logic(%), I/O(%), and EBR(%) drop-down menus.

   Power Calculator calculates the resource usage based on your selections and displays the utilization in the bottom portion of the dialog box.
4. Click **OK**.

**See Also** "Estimating Routing Resource Usage" on page 246

# Estimating Routing Resource Usage

You can estimate the amount of routing resources that your design will use for the purpose of power analysis.

**To estimate routing resource usage:**

1. Choose **Edit > Estimation Mode Settings**.

   The Power Calculator - Estimation Mode Settings dialog box appears.
2. From the Routing Resource Utilization drop-down menu, select the amount of routing resources that you expect your design to use. You can select from the following:

- Low – Uses a small amount of routing resources.
- Medium – Uses an average amount of routing resources. This setting is the default.
- High – Uses a large amount of routing resources.

3. Click **OK**.

**See Also** ▶"Estimating Resource Usage" on page 246

# Controlling Operating Temperature

Minimizing the device's operating temperature is critical to reducing power consumption.

A device has two parts: the die, which is the silicon inside the device, and the package, which is the outer shell. Each die-package combination has a thermal resistance value (often referred to as theta), which is a measure of how well the combination can dissipate heat. Lower values indicate better heat dissipation for the device.

Thermal impedance is the cumulative individual thermal resistances of a defined network. Power Calculator offers different models that provide ways to calculate the thermal impedance for a given device when it is mounted on the board. These models cover scenarios related to board sizes, air flows, and heat sinks that affect the thermal impedance. You can use these models to calculate the thermal impedance for the scenario that you choose.

You can choose the thermal impedance models by clicking the **Thermal Profile** button in the Environment section at the top right of the Power Summary page.

These thermal impedance models use the following terminology:

- Junction temperature – the temperature of the die in the device package, in degrees Celsius. You can adjust the junction temperature by choosing a model that applies a heat sink and changes the air flow value. Junction temperature is also affected by the package that you select in the Package Type box. In addition, the changes that you enter in many of the editable (white and turquoise) cells on the Power Summary page affect junction temperature.
- Heat sink – any material or object that dissipates unwanted heat from a device by absorbing it and conducting it away to a surface from which it dissipates into its surroundings. The reduction of junction-to-ambient thermal impedance depends on different factors, such as the speed and direction of the air flow over the heat sink and the materials used to attach the heat sink to the package. For heat-sink properties and proper attachment methods, contact your heat-sink manufacturer for specifications.
- Air flow – the movement of air around the device in a package to cool it. It is measured in linear feet per minute (LFM). The higher the air flow value you select, the greater the cooling effect on the device.

- Ambient temperature – the expected operating temperature, in degrees Celsius, of the medium surrounding a device in a package.
- Theta JA – the thermal impedance between the silicon die and the ambient air within a JEDEC-defined environment. The boards used to measure these values have four layers, and their size is defined by JEDEC specifications.
- Effective Theta JA – similar to Theta JA, but defined as the sum of all the package and board thermal resistances outside of a JEDEC-defined environment. It indicates how well the heat dissipates from the die to the ambient (air) for a particular thermal network as a whole outside of a JEDEC-defined environment.
- Theta JB – indicates how well the heat dissipates from the junction on the silicon die to the board.
- Theta JC – indicates how well the heat dissipates from the junction of the die to the package case in which it is enclosed, as defined by the JEDEC specifications.
- Theta BA – indicates how well the heat dissipates from the board to the ambient (air).
- Theta CS – indicates how well the heat dissipates from the package case to the heat sink. It is a measure of the thermal resistance of the interface material that makes contact between the package case and the heat sink attached to the package. It can be thermal grease, double-sided sticky tape, glue, or phase-shift material.
- Theta SA – indicates how well the heat dissipates from the heat sink to the ambient (air).

**See Also** ▶"Power Calculator Pages" on page 228

# Selecting a Thermal Impedance Model

You can experiment with various board sizes, heat sinks, and air flow settings to select a thermal impedance model for your design. The choice of board affects the Theta JB and Theta BA values, and the heat sink and air-flow selections affect the Theta SA and Theta JA values.

**Note**

The current values provided with the thermal models are averages of different values, and they are provided as a courtesy. To produce better predictions, it is advised that you submit your own thermal values.

**To select a thermal impedance model:**

1. In the Environment section of the Power Summary page, click **Thermal Profile**.

   The Effective Theta-JA specifies the cumulative thermal impedance of a particular system. This figure is used in calculating the junction

temperature (Tj) of a die in a particular environment according to the following formula:

Tj = *power * theta_effective + ambient_temperature*.

2. In the Power Calculator - Thermal Profile dialog box, select the source of the effective thermal impedance (effective Theta JA) value:
   - If you want Power Calculator to calculate the effective Theta JA value from the selections that you make in the Board Selection, Heat Sink Selection, and Airflow Selection drop-down menus, choose **Use Thermal Models**.
   - If you want to provide your own effective Theta JA value, choose **User-Defined Effective Theta-JA**.
3. If you chose Use Thermal Models:
   a. Select the size of the board that to use from the Board Selection drop-down menu:
      - JEDEC Board (2S2P) – Specifies that a board defined by JEDEC specifications be used. These specifications are based on real boards and measurements conducted in the lab used by Lattice Semiconductor. For packages < 27.0 mm in length, the buried planes are 74.2 mm x 74.2 mm (3” x 3”). For packages larger than or equal to 27.0 mm, the buried planes are 99.6 mm x 99.6 mm (4” by 4”). Power Calculator uses Theta JA or Theta JC, depending on the type of heat sink selection you make in the Heat Sink Selection menu. Theta JA is used only if you choose No Heat Sink. The ThetaJA value is the published value measured in the lab.
      - Small Board – Specifies that a board that is slightly larger than the JEDEC board be used. The board is assumed to be 6” to 8” square. This setting adds the Theta JB value to the board thermal impedance.
      - Medium Board – Specifies that a medium board be used, one that is assumed to be 8” to 12” square. This setting adds the Theta JB value to the board thermal impedance.

- Large Board – Specifies that a large board be used, one that is assumed to be larger than 14” square. This setting adds the Theta JB value to the board thermal impedance.

**Note**

Many factors can affect the actual thermal properties and change the values, including:

- The number of board layers.
- Airflow over the board.
- Number of devices powered up around the device package and their distance from the package.
- Thickness of the copper and the width of the traces on each layer.
- Number of thermal vias.
- Shape and thickness of each power and ground plane below the transfer of heat from the die to the ambient environment.

As a result, users are encouraged to make their own thermal measurements or simulations and use them in Power Calculator to see what kind of junction temperature might result.

b. Select the type of heat sink from the Heat Sink Selection drop-down menu:
   - No Heat Sink – Specifies that no heat sink be used. Theta JA is used, and you must choose the air flow from the Airflow Selection menu. The No Heat Sink setting is the default.
   - Low-Profile Heat Sink – Specifies that a short heat sink be used.
   - Medium-Profile Heat Sink – Specifies that a medium heat sink be used. This setting adds the Theta JC value to the Heat Sink thermal impedance.
   - High-Profile Heat Sink – Specifies that a tall heat sink be used.
   - Custom-Profile Heat Sink – Enables you to specify your own heat-sink value in the Theta-SA for Custom Heat Sink box. The Airflow Selection option is not available when you select Custom-Profile Heat Sink.

c. Select the air flow in linear feet per minute (LFM) from the Airflow Selection drop-down menu:
   - 0 LFM
   - 200 LFM
   - 500 LFM

   If you use a heat sink, the 0 LFM setting is not available.

   The Airflow Selection option is not available when you select Custom-Profile Heat Sink from the Heat Sink Selection menu.

4. If you chose User-Defined Theta-JA Effective, enter your effective thermal impedance value in the Effective Theta-JA box. Values entered must be greater than 0.

This box is not available if you selected Use Thermal Models.

5. Click **OK**.

The new effective Theta JA value now appears in the Effective Theta-JA box in the Environment section of the Power Summary page.

**See Also** ▶"Controlling Operating Temperature" on page 247

▶ "Changing the Ambient Temperature" on page 251

## Changing the Ambient Temperature

The ambient temperature is the expected operating temperature, in degrees Celsius, of the medium surrounding a device in a package. You can select an ambient temperature in the range of -40 to 125 degrees Celsius.

**To change the ambient temperature:**

- In the Environment section of the Power Summary page, enter a value, in degrees Celsius, in the Ambient Temperature box and press **Enter** or click anywhere outside the cell.

  The value entered must be between -40 and +125 degrees Celsius. The default value is 25 degrees Celsius for all devices.

When you change the value in the Ambient Temperature box, Power Calculator updates the following cells:

- Junction Temperature.
- Values in the Static (A) and Total (A) columns of the Current by Power Supply and Power by Power Supply sections.
- The totals in the Power by Block Column.

If you enter a value that is beyond the commercial, industrial, or automotive device limits, you will receive an error message that displays the range of valid values.

**See Also** ▶"Controlling Operating Temperature" on page 247

▶ "Selecting a Thermal Impedance Model" on page 248

# Viewing and Printing Results

In addition to the automatically generated reports from power settings, Power Calculator provides graphs of power consumption and enables you to print information from the pages and generate a comma-separate value file (.csv) from the command line.

# Generating Power Graphs

Power Calculator can create graphs showing how power consumption is affected when you vary the voltage, temperature, and clock frequency in the design. You can generate three default power graphs on the Graph page:

- Power vs. Supply Voltage
- Power vs. Ambient Temperature
- Power vs. Frequency

Each graph displays a typical and worst case plot. You can select the X axis and Y axis for each graph.

Graphs are only generated when you select the Graph tab. During graph generation, you cannot select a different tab until the calculations are finished. If you change the information in any other page that alters the power, the graphs will be regenerated when you again select the Graph tab. If you do not change any power information, the graphs will not be regenerated.

**To generate the Power vs. Supply Voltage graphs:**

1. Choose **Edit > Graph Settings**.
2. In the Power by Section part of the Graphs Settings dialog box, set the following options:
    a. In the Y Axis box, select Total Power or the specific type of block power to display, such as I/O or Block RAM, to place on the Y axis of the graph.
    b. In the X Axis box, select the type of supply voltage to place on the X axis of the graph.
    c. In the Lower Limit box, enter the lower boundary of the voltage range on the X axis. The lower limit can be 5 percent lower than the nominal supply value.
    d. In the Upper Limit box, enter the upper boundary of the voltage range on the X axis. The upper limit can be 5 percent higher than the nominal supply value.
    e. In the Resolution box, enter the step in which the supply voltage values on the X axis should appear. The voltage supply step is limited to a resolution of .01.
3. Click **OK** to close the dialog box and apply the settings.
4. Click the Graph tab to see the resulting charts.

**To generate the Power vs. Ambient Temperature graphs:**

1. Choose **Edit > Graph Settings**.
2. In the Power by Temperature part of the Graphs Settings dialog box, set the following options:
    a. In the Y Axis box, select Total Power or the type of block power, such as I/O or Block RAM, to place on the Y axis of the graph.

b. In the X Axis box, select **Ambient Temperature** for the X axis of the graph. Temperature is in degrees Celsius.

c. In the Lower Limit box, enter the lower boundary of the temperature range on the X axis. The range limits are determined by the device's operating condition:

- Industrial: –40 through 105
- Commercial: 0 through 85
- Automotive: –40 through 125

d. In the Upper Limit box, enter the upper boundary of the temperature range on the X axis. The range limits are determined by the device's operating condition:

- Industrial: –40 through 105
- Commercial: 0 through 85
- Automotive: –40 through 125

e. In the Resolution box, enter the step in which the temperature values on the X axis should appear. The temperature step is limited to resolution of 10 degrees Celsius.

3. Click **OK** to close the dialog box and apply the settings.
4. Once the graphic data has been prepared, click the Graph tab to see the resulting charts.

**To generate the Power vs. Frequency graphs:**

1. Choose **Edit > Graph Settings**.
2. In the Power by Frequency part of the Graphs Settings dialog box, set the following options:

a. In the Y Axis box, select Total Power or the specific type of block power, such as I/O or Block RAM, to place on the Y axis of the graph.

b. In the X Axis box, select the clock to place on the X axis of the graph. It can be any of the clocks listed on the Clocks page.

c. In the Lower Limit box, enter the lower boundary of the frequency range on the X axis. The lower boundary is limited to 0 MHz.

d. In the Upper Limit box, enter the upper boundary of the frequency range on the X axis. The upper boundary is limited to 10000 MHz.

e. In the Resolution box, enter the step in which the frequency values on the X axis should appear. The default frequency increment is 20 MHz.

3. Click **OK** to close the dialog box and apply the settings.
4. Once the graphic data has been prepared, click the Graph tab to see the resulting charts.

# Viewing the Power Calculator Report

The report page contains a summary of the estimated power-consumption or current-consumption data calculated by Power Calculator. It is available in text (ASCII) format and in HTML format, and it is updated each time you make a change to any of the data in the editable cells.

**To view the Power Calculator report:**

- To view the results in text format, click the Report tab.
- To view the results in HTML format, click **View HTML Report**.

**See Also** "Entering Data" on page 239

"Printing Information" on page 254

# Printing Information

After calculating power, you can print the results displayed on any of the pages, including the Report and Graph pages. Optionally, you can preview pages before printing.

**To preview and print:**

1. In the Power Calculator window, click the tab of the desired page.
2. Choose **File > Print Preview** to activate the Print Preview dialog box.

   Use the zoom tools and the Fit Page and Fit Width buttons on the toolbar to position the page in the window.

   When there are multiple pages, use the Next, Last, Previous, and First arrows to navigate through them.
3. Select the Portrait or Landscape button on the toolbar.
4. Click **Print** to queue the results to a printer.
5. In the Print dialog box, click **Print**.

**To print information from a page:**

1. In the Power Calculator window, click the tab of the desired page.
2. Choose **File > Print**.
3. In the dialog box, click **Print**.

**See Also** "Entering Data" on page 239

"Power Calculator Pages" on page 228

# Programming the FPGA

After you have created and verified your design, you can use the final output data file to download or upload a bitstream to or from an FPGA device using the Radiant software Programmer.

The Radiant software Programmer supports serial and microprocessor programming of Lattice devices in PC and Linux environments. A device can be scanned automatically using the Radiant software Programmer graphical user interface.

The Radiant software Programmer is available from the Radiant software environment, and is also available in a standalone version.

Features include:

- Scan and display contents (.xcf file)
- Download data files to devices
- Create/modify/display .xcf file
- Generate embedded file on the .xcf file

The Radiant software Programmer uses a Single Document Interface (SDI) where a single .xcf project is displayed per the Radiant software Programmer instance. Opening additional .xcf files in the Radiant software-integrated mode will close the current .xcf and open the specified .xcf. To open additional .xcf files in standalone mode, a separate instance of the Radiant software Programmer will need to be launched.

The main view displays the devices in the current Radiant software Programmer project resulting from the  Scan action, or from manual creation in a table.

Double-clicking on an uneditable cell or right-clicking and selecting **Device Properties** opens the “Device Properties Dialog Box” on page 285 that displays more information about the selected device. Additionally, some entries may be edited directly on the table by clicking them.

Columns can be displayed or hidden by choosing **View > Columns**. Some columns become un-editable (grayed out) if the selected operation or other option does not support it.

Each row has a column for the device status. The status indicates whether the operation performed was successful or not. This field is also used for read back operations to display what is read back if the data to display is short. For larger data sets that are read back, a dialog box is displayed.

**See Also** ▶"Programming the FPGA" on page 258

- "File Formats" on page 259
- "SPI Flash Support" on page 259
- "Using the Radiant Software Programmer" on page 267
- "Programmer Options" on page 284
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# File Formats

Lattice supports the following file formats.

**Data File** A data file can be a hex, or bitstream file. Each of these files is based upon an IEEE programming standard:

**Bitstream** Data files used for Configuring volatile memory (SRAM) of our FPGA's.

**Hex** Hexadecimal PROM data files used for Programming into external non-volatile memory, such as parallel or Serial Peripheral Interface (SPI) Flash devices.

**See Also** ▶"Programming the FPGA" on page 258

- "Using the Radiant Software Programmer" on page 267
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# SPI Flash Support

The Radiant software Programmer, combined with a Lattice cable fly-wire, supports the programming of SPI flash devices.

**Lattice Devices That Support SPI Flash Configuration** The iCE40 device family of FPGAs can be configured directly from an SPI flash memory devices.

**Third Party SPI Flash Devices** The following tables list supported third-party SPI flash devices. Click the third-party manufacturer name to link to the table of supported third-party SPI flash devices.

**Note**

The Radiant software Programmer support for additional third-party SPI flash devices may be added from time to time. To see if support for third-party SPI flash devices has been added, in addition to those listed in the tables below, check the SPI Flash options in the "Device Properties Dialog Box" on page 285.

AMIC

Atmel

Intel

Macronix

NexFlash

Numonyx

Micron

Spansion

SSTI

STMicro

WindBond.

**Table 12: AMIC**

| SPI Device |
|---|
| SPI-A25L10P |
| SPI-A25L20P |
| SPI-A25L40P |
| SPI-A25L80P |
| SPI-A25L16P |

Return to Top

**Table 13: Atmel**

| SPI Device |
|---|
| SPI-AT25DF021 |
| SPI-AT25DF041A |
| SPI-AT25F512A |
| SPI-AT25F1024A |
| SPI-AT25F2048 |
| SPI-AT25F4096 |
| SPI-AT25FS010N |
| SPI-AT25FS040N |
| SPI-AT25DF081 |
| SPI-AT25DF161 |
| SPI-AT25DF321 |
| SPI-AT25DF641 |
| SPI-AT26DF081A |
| SPI-AT26DF161 |
| SPI-AT26DF161A |
| SPI-AT26DF321 |
| SPI-AT45DB041D |
| SPI-AT45DB081D |
| SPI-AT45DB161D |
| SPI-AT45DB321D |
| SPI-AT45DB642D |

Return to Top

**Table 14: Intel**

| SPI Device |
|---|
| SPI-25F016S33 |
| SPI-25F160S33 |
| SPI-25F320S33 |
| SPI-25F640S33 |

Return to Top

**Table 15: Macronix**

| SPI Device |
|---|
| MX25L1005 |
| MX25L2005 |
| MX25L4005(A) |
| MX25L8005 |
| MX25L1605 |
| MX25L3205 |
| MX25L6405 |
| MX25L12805 |
| MX25U8035 |

Return to Top

**Table 16: NexFlash**

| SPI Device |
|---|
| NX25P10 |
| NX25P20 |
| NX25P40 |
| NX25P80 |
| NX25P16 |
| NX25P32 |

Return to Top

**Table 17: Numonyx**

| SPI Device |
|---|
| SPI-M25P05-A |
| SPI-M25P10-A |
| SPI-M25P20 |
| SPI-M25P40 |
| SPI-M25P80 |
| SPI-M25P16 |
| SPI-M25P32 |
| SPI-M25P64 |
| SPI-M25P128 |
| SPI-M25PE10 |
| SPI-M25PE20 |
| SPI-M25PE40 |
| SPI-M25PE80 |
| SPI-M25PE16 |
| SPI-M25PX80 |
| SPI-M25PX16 |
| SPI-M25PX32 |
| SPI-M25PX64 |
| SPI-M45PE10 |
| SPI-M45PE20 |
| SPI-M45PE40 |
| SPI-M45PE80 |
| SPI-M45PE16 |
| SPI-N25Q032 |
| SPI-N25Q064 |
| SPI-N25Q128 |
| SPI-N25Q128A |

Return to Top

**Table 18: Micron**

| SPI Device |
|---|
| SPI-M25P05-A |
| SPI-M25P10-A |
| SPI-M25P20 |
| SPI-M25P40 |
| SPI-M25P80 |
| SPI-M25P16 |
| SPI-M25P32 |
| SPI-M25P64 |
| SPI-M25P128 |
| SPI-M25PE10 |
| SPI-M25PE20 |
| SPI-M25PE40 |
| SPI-M25PE80 |
| SPI-M25PE16 |
| SPI-M25PX80 |
| SPI-M25PX16 |
| SPI-M25PX32 |
| SPI-M25PX64 |
| SPI-M45PE10 |
| SPI-M45PE20 |
| SPI-M45PE40 |
| SPI-M45PE80 |
| SPI-M45PE16 |
| SPI-N25Q032 |
| SPI-N25Q032A |
| SPI-N25Q064 |
| SPI-N25Q128 |
| SPI-N25Q128A |
| SPI-N25Q256 |
| SPI-N25Q512 |

Return to Top

**Table 19: Spansion**

| SPI Device |
|---|
| SPI-S25FL001D |
| SPI-S25FL002D |
| SPI-S25FL004D |
| SPI-S25FL040A |
| SPI-S25FL004A |
| SPI-S25FL008A |
| SPI-S25FL016A |
| SPI-S25FL032A |
| SPI-S25FL064A |
| SPI-S25FL032P |
| SPI-S25FL064P |
| SPI-S25FL164K |
| SPI-S25FL128P00 |
| SPI-S25FL128S |
| SPI-S25FL204K |
| SPI-S25FL208K |
| SPI-S25FL216K |

Return to Top

**Table 20: SSTI**

| SPI Device |
|---|
| SPI-SST25LF020A |
| SPI-SST25LF040A |
| SPI-SST25LF080A |
| SPI-SST25VF512A |
| SPI-SST25VF010A |
| SPI-SST25VF020 |
| SPI-SST25VF040 |
| SPI-SST25VF080 |
| SPI-SST25VF040B |
| SPI-SST25VF080B |
| SPI-SST25VF016B |
| SPI-SST25VF032B |
| SPI-SST25VF064C |

Return to Top

**Table 21: STMicro**

| SPI Device |
|---|
| SPI-M25P05-A |
| SPI-M25P10-A |
| SPI-M25P20 |
| SPI-M25P40 |
| SPI-M25P80 |
| SPI-M25P16 |
| SPI-M25P32 |
| SPI-M25P64 |
| SPI-M25PE10 |
| SPI-M25PE20 |
| SPI-M25PE40 |
| SPI-M25PE80 |
| SPI-M25PE16 |
| SPI-M45PE10 |
| SPI-M45PE20 |
| SPI-M45PE40 |
| SPI-M45PE80 |
| SPI-M45PE16 |

Return to Top

**Table 22: WindBond**

| SPI Device |
|---|
| W25P10 |
| W25P20 |
| W25P40 |
| W25P80 |
| W25P16 |
| W25P32 |
| W25X10 |
| W25X20 |
| W25X40 |
| W25X80 |
| W25X16 |
| W25X32 |
| W25X64 |

Return to Top

**See Also** "Programming the FPGA" on page 258

- "Using the Radiant Software Programmer" on page 267
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Using the Radiant Software Programmer

This section provides procedures for using the Radiant software Programmer. Topics include:

# Plugging the Cable into the PC

You can plug the USB cable into the PC whether the PC is turned on or off. However, make sure that the Radiant software Programmer is closed before plugging the cable into your PC or unplugging it.

**To plug the cable into the PC:**

1. Make sure that the Radiant software Programmer is closed, and then plug the cable into your PC.
2. If your PC is turned on, wait about one minute for the Windows operating system to recognize the USB cable. The amount of time will vary, depending on your PC's speed.

**Note**

Be sure to turn off the target board's power before connecting or disconnecting the USB cable to the target board.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Detecting a Cable

You can use the Radiant software Programmer Detect Cable feature to determine which cable and port you are using.

**To detect a cable:**

- In the Cable Setup dialog box, click **Detect Cable**. The Radiant software Programmer detects all available cables connected to your PC, and lists the cable type, port settings, and descriptions in the Output console.

**To detect FTDI cable:**

10. To plug in FTDI cable, see "Plugging the Cable into the PC" on page 268.
11. In the Cable Setup dialog box, click **Detect Cable**. The Radiant software Programmer detects the HW-USBN-2B (FTDI) cable connected to your PC, FTUSB-0 as the port settings, and descriptions in the Output console.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Cable Setup Dialog Box" on page 290
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Selecting from Multiple Cables

The Radiant software Programmer can recognize as many as five USB cables plugged into the same PC.

The first USB cable plugged into the PC is assigned port Ez-USB-0. The second is assigned port Ez-USB-1, and so on. The first FTDI USB2 cable is assigned port FTUSB-0, and the second FTUSB-1, and so on.

If the PC is turned off, and then later turned back on, the assigned ports might change, since the PC detects the cables according to the USB port address instead of the order in which they were plugged into the PC. Since the USB cable port assignments might change during power-up, you might have to reselect the USB cable in the Cable Setup dialog box.

**To select a USB cable from multiple cables:**

- In the Cable Setup dialog box, select the USB cable you want to use from the Port pulldown list.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Cable Setup Dialog Box" on page 290
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Creating a New Radiant Software Programmer Project

By creating a new Radiant software Programmer project, you create a chain file (.xcf) from a scan with default settings, create an .xcf file from a scan with custom settings, or a new blank project.

How you start the Radiant software Programmer also depends on your operating system, and whether you are using it integrated with the Radiant software or using the stand-alone version.

**Note**

You can change the way the Radiant software Programmer writes the paths to data files in the .xcf file by editing the programmer .ini file.

By default, the path in the .xcf file is displayed in "cross platform compatible" format, meaning forward-slashes ("/") when using the Radiant software Programmer on Windows and Linux. By editing the programmer .ini file, you can change the way the path is saved to "Native Delimiter" format, meaning back-slashes ("\") when using the Radiant software Programmer on Windows. Radiant software Programmer on Linux will continue to use forward-slashes ("/"), thus, this option is not required for Linux.

To enable Native Delimiter format, add the following line to the end of the programmer.ini file:

PathDelimiter=1

If you wish to change back to the default Cross Platform Compatible format, you can change the value in the programmer.ini file to:

PathDelimiter=0

or delete the line entirely.

The programmer.ini file is located in the following directory:

<*Drive*>:\Users\<User Name>\AppData\Roaming\LatticeSemi\DiamondNG

**To create a new Radiant software Programmer project:**

1. Make sure that the board is turned on and that the Lattice parallel or USB cable is properly connected to the computer.
2. Issue the start command.
   - In the Radiant software main window, choose **Tools > Programmer**; or in Windows choose **All Programs > Lattice Radiant Software> Accessories > Radiant Programmer**; or click  in the Radiant software toolbar.
   - In Linux, from the *<install_path>/programmer/bin/lin64* directory, enter the following on a command line:

     ```
     ./programmer
     ```
3. In the Getting Started dialog box, enter a project name in the **Project Name** box.
4. Browse to the location where you wish your project to reside in the **Project Location** box.
5. Choose one of the following:
   - **Create a new project from a scan.** This option creates a new project from a scan.
     - If you have a board connected to your computer, you can click the **Detect Cable** button to detect the cable and port. Otherwise, you

can manually select cable and port from the **Cable** and **Port** pulldown lists.

- **Create a New Blank Project** - This option creates a new blank Radiant software Programmer project.

6. Click **OK**.

**See Also** "Using the Radiant Software Programmer" on page 267

- "Getting Started Dialog Box" on page 285
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Opening an Existing Radiant Software Programmer Project

To open an existing Radiant software Programmer project, you must have an existing chain file (.xcf). The daisy chain configuration must be the same as that for which the chain configuration file was created. Each unique chain configuration requires a unique chain configuration file.

How you start the Radiant software Programmer also depends on whether you are using it integrated with the Radiant software or using the stand-alone version, and on your operating system.

**To start the Radiant software Programmer with an existing project:**

1. Issue the start command. To start:
   - In the Radiant software main window, choose **Tools > Programmer**; or in Windows choose **All Programs > Lattice Radiant Software > Accessories > Radiant Programmer**; or click  in the Radiant software toolbar.
   - In Linux, from the *<install_path>/programmer/bin/lin64* directory, enter the following on a command line:

     ```
     ./programmer
     ```

2. In the Getting Started dialog box, choose **Open an Existing Programmer Project.**
3. Click **OK**.

**See Also** "Using the Radiant Software Programmer" on page 267

- "Getting Started Dialog Box" on page 285
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Checking the XCF Project

The Check XCF button allows you to perform a design rule check on your XCF project setup before performing any actions. This feature is run automatically when the .xcf file is saved.

**To check the XCF project:**

- In the Radiant software Programmer, choose **Run > Check XCF Project** or click  in the toolbar. The Radiant software Programmer checks the XCF project and indicates in the output pane whether or not the project is valid.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Scanning the Device

There are several ways to scan a device. When you create a new project, you can choose to scan you board, perform a custom scan, or create a blank project without performing a scan.

**Note**

Not all Lattice device families support Scan capability. Refer to the *Lattice Radiant Software 1.0 Release Notes* for more information on the device family that does not support Scan.

**To scan the device chain on the board that is connected to your PC:**

1. Make sure that the board is turned on and that the cable is properly connected to the Radiant software Programmer.
2. In the Radiant software Programmer, choose **Run > Scan Device** or click  in the toolbar. The Radiant software Programmer opens a new configuration file, scans the printed circuit board connected to your computer, and lists the devices in the new file.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Selecting from a Matching Device ID List

When a scanned device shares the same device ID with other devices, the software indicates this in the main window with the message “Cannot identify detected device on *row number*>. Please manually select correct device.”

**To select a different device from the matching ID list:**

1. Click the device in the Device column
2. Select the device you want to use from the dropdown menu.

**See Also** ▶“Using the Radiant Software Programmer” on page 267

- “Programming the FPGA” on page 258
- “Programming and Configuring iCE40 Devices with Programmer” on page 293

# Adding a Lattice Device to a Chain

You can add any Lattice device to an existing chain.

**To add a Lattice device to a chain:**

1. In the Radiant software Programmer, choose **Edit > Add Device,** click the  button on the toolbar, or right-click and choose **Add Device**.
2. Select a Lattice device family from the **Device Family** column dropdown menu.
3. Select a device from the **Device** column dropdown menu.

   The software inserts the new device into the active chain.

**See Also** ▶“Using the Radiant Software Programmer” on page 267

- “Programming the FPGA” on page 258
- “Programming and Configuring iCE40 Devices with Programmer” on page 293

# Adding a Generic JTAG Device to a Chain

You can add any one of the following types of devices to a chain:

- JTAG-NOP Device
- iCE40 UltraPlus

**To add a Generic JTAG device:**

1. In the Radiant software Programmer, choose **Edit > Add Device,** click the  button on the toolbar or right-click and choose **Add Device**.
2. Select Generic JTAG Device from the **Device Family** column dropdown menu.
3. Select a JTAG-NOP from the **Device** column dropdown menu.
4. Highlight the row, and choose **Edit > Device Properties**, click the  button on the toolbar, or right-click and choose **Device Properties**. In the Device Properties dialog box, click the Browse button and navigate to the bitstream for the device. Select the file and click **Open**.

   The software inserts the new device into the active chain.

**To add an iCE UltraPlus device:**

1. In the Radiant software Programmer, choose **Edit > Add Device,** or click the  button on the toolbar or right-click and choose **Add Device**.
2. Select iCE40 UltraPlus from the **Device Family** column dropdown menu.
3. Select iCE40UP3K or iCE40UP5K from the **Device** column dropdown menu.
4. Highlight the row, and choose **Edit > Device Properties**, click the  button on the toolbar, or right-click and choose **Device Properties**. In the Device Properties dialog box, click the Browse button and navigate to the bitstream for the device. Select the file and click **Open**.

   The software inserts the new device into the active chain.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Device Properties Dialog Box" on page 285
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Removing a Device from a Chain

**To remove a device from the active chain:**

1. In the Radiant software Programmer, select the device that you want to delete.

2. Choose **Edit > Remove Device**, click the  button on the toolbar, or right-click and chose **Remove Device**.

**Note**

There is no Undo for this command.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Editing Device Properties

**To edit device properties in a chain:**

1. In the Radiant software Programmer, select the device that you want to edit.
2. Choose **Edit > Device Properties**, click the  button on the toolbar, or right-click and choose **Device Properties**.
3. In the Device Properties dialog box, edit the device properties.
4. Click **OK** to close the Device Properties dialog box.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Device Properties Dialog Box" on page 285
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Moving a Device Up or Down

**To move a device up or down in the Radiant software Programmer:**

- With the left mouse button, select and hold the number in the far left in the row of the device that you want to move. Then drag and drop to the row where you want to move the device.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Setting Target Memory, Port Interface, Access Mode, and Operation

**To set a device operation:**

1. Select a device.
2. Double-click the appropriate column for the device.
3. In the Device Properties dialog box, select the specific operation for the device in the Operation drop down menu.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Device Properties Dialog Box" on page 285
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Setting Programming Characteristics

Use the Settings dialog box to specify how the Radiant software Programmer will process the configuration. You can also specify other options.

**To specify Programming characteristics:**

1. In the Radiant software Programmer, choose **Edit > Settings**.

   The Settings dialog box opens.
2. Select the options you want, and then click **OK**.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Settings Dialog Box" on page 289
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Downloading Design Files

After you have specified all of the processing options, use the Program command to download the design files to the devices.

- In the Radiant software Programmer, choose **Run > Program Device,** or click  on the toolbar.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258

- “Programming and Configuring iCE40 Devices with Programmer” on page 293

# Generating Embedded Design Files

Lattice Embedded VME enables in-field upgrades of Lattice programmable devices by suitable embedded processors. Lattice provides the option to generate several different file formats for different embedded target options. For example, for iCE40UP, the following embedded solution is supported:

- Slave SPI Embedded: Enables field upgrades via the slave SPI port.

**To generate embedded design files:**

1. In the Embedded Options tab, select the desired embedded options. For more information on embedded options, refer to Table 23 on page 277 and Table 24 on page 278.
2. Choose **Run > Generate Embedded Code**, or click  on the toolbar. The files are generated in the specified directory.

The following table lists device family, input file types, and output file extensions for Slave SPI embedded deployment.

**Table 23: Slave SPI Embedded -- Device Family, Input File 1, Output File 1, and Output File 2**

| Device Family | Input File | Output File 1 | Output File 2 |
|---|---|---|---|
| iCE40UP | XCF File (.xcf)<br>Bitstream (.bin/.hex) | Algorithm VME File (*_algo.sea) - This is the algorithm file that specifies which operations will be executed must be used with a VME data file. This file is used in device programming or in field upgrading. | Data File (*_data.sed) |

The following table lists Compress Embedded Files, Converted VME files to HEX (c.) Prom-Based Embedded VME -- for Slave SPI embedded deployment, and Insert Source Code options.

**Table 24: Slave SPI Embedded Options -- Compress Embedded Files, Convert VME files to HEX (c.) Prom-Based Embedded VME, and Insert Source Code**

| Device Family | Compress Embedded Files | | Convert VME files to HEX (c.) Prom-Based Embedded VME | | Include Source Code | |
|---|---|---|---|---|---|---|
| iCE40UP | **On** | **Off** | **On** | **Off** | **On** | **Off** |
| | Compress data. This option reduces the file size of the VME file through compression, using less memory space. Unchecking this option generates an uncompressed VME, which is useful for debugging. Compressed and uncompressed VME have the same level of performance. | Do not compress. | This operation will create a VME file that will be used in file based embedded programming where the programming of the device is based on the input file. When this option is turned on then it will create a .c file which will be used in EPROM base programming to create a single complied image that will be loaded in to the CPU. An then the CPU will use this image to then program the device. | Do not generate .c file. | Copy contents of sspiembedded\sourcecode directory to working project directory.<br>There are two directories of the Slave SPI Embedded source code:<br>▶ sspiem<br>▶ sspiem_eprom<br>**Note**: If the “Convert VME files to HEX (c.) Prom-Based Embedded VME” is checked, the contents of the sspiem_ecprom directory will be copied.<br>If the “Convert VME files to HEX (c.) Prom-Based Embedded VME” is not checked, the contents of the sspiem directory will be copied. | Do not copy. |

**See Also:** ▶Lattice Radiant Software User Guide

# Adding a Custom SPI Flash Device

This feature allows you users to add your own SPI Flash device to the Radiant software Programmer database.

**To create a Custom SPI Flash device:**

1. Choose **Tools > Custom Flash Device**.
2. In the Edit Custom Device dialog box, select **SPI Serial Flash Custom**, and click **Add**.
3. In the Custom SPI Flash dialog box, enter:
   - **Device Description** -- this can be any alpha-numeric string that describes the device (for example: M25P32-VMF6C).

- **Device Name** -- this can be any alpha-numeric string that describes the device name (for example: SPI-M25P32).
- **Package** -- this can be any alpha-numeric string that describes the package (for example: 16-pin SOIC).
- **Device ID** -- this can be any alpha-numeric string that describes the device ID (for example: 0x15).

4. In the Device Vendor dropdown menu, choose device vendor.
5. In the Device Density dropdown, choose device density.
6. In the Byte Per Sector dropdown, choose the desired value.
7. If the SPI Flash device supports sector protection, check the **Protection Options On** box.
8. Click **OK**.
9. Once the custom SPI flash has been added, it can be selected in the Device Properties dialog box.

**See Also** "Using the Radiant Software Programmer" on page 267

- "Device Properties Dialog Box" on page 285
- "Custom SPI Flash Dialog Box" on page 288
- "Edit Custom Device Dialog Box" on page 288
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Editing a Custom SPI Flash Device

If you have already created a custom SPI Flash device, you can edit its properties in the Custom SPI Flash dialog box.

**To edit the properties of a Custom SPI Flash device:**

1. Choose **Tools > Custom Flash Device**.
2. In the Edit Custom Device dialog box, select **SPI Serial Flash Custom**, select the SPI Flash device you want to edit, and click **Edit**.
3. In the Custom SPI Flash dialog box, enter:
   - **Device Description** -- this can be any alpha-numeric string that describes the device (for example: M25P32-VMF6C).
   - **Device Name** --lists the device name (for example: SPI-M25P32). This can't be edited.
   - **Package**-- this can be any alpha-numeric string that describes the package (for example: 16-pin SOIC).
   - **Device ID**-- this can be any alpha-numeric string that describes the device ID (for example: 0x15).

4. In the Device Vendor dropdown menu, choose device vendor.
5. In the Device Density dropdown, choose device density.
6. In the Byte Per Sector dropdown, choose the desired value.
7. If the SPI Flash device supports sector protection, check the **Protection Options ON** box.
8. Click **OK**.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Custom SPI Flash Dialog Box" on page 288
- "Edit Custom Device Dialog Box" on page 288
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Removing a Custom SPI Flash Device

Once you have created a custom SPI Flash device, you can remove it from the Custom SPI Flash dialog box.

**To edit the properties of a Custom SPI Flash device:**

1. Choose **Tools > Custom Flash Device**.
2. In the Edit Custom Device dialog box, select **SPI Serial Flash Custom**.
3. Select the SPI Flash device you want to remove.
4. Click **Remove**.
5. Click **OK**.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Custom SPI Flash Dialog Box" on page 288
- "Edit Custom Device Dialog Box" on page 288
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Programming Using Custom SPI Flash Device

After a custom SPI Flash device has been created, you can program it using the Radiant software Programmer.

**To program using a Custom SPI Flash device:**

1. Left-click on the row corresponding to the device you want to edit, and choose **Edit > Device Properties** to display the Device Properties dialog box.
2. In the Target Memory Mode dropdown list, choose **External SPI Flash memory (SPI FLASH)**.
3. In the Family dropdown list, choose **SPI Serial Flash Custom**.
4. In the Device dropdown list, choose the name of the custom device you created.
5. Click **OK.**

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Device Properties Dialog Box" on page 285
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Unlocking Secure SPI Flash Support

If a custom SPI flash device or a SPI flash device was secured and you wish to reprogram the SPI flash, you must first unlock the device before reprogramming it.

**To unlock a secure SPI flash:**

1. Create a new the Radiant software Programmer project as described in "Creating a New Radiant Software Programmer Project" on page 269 or open an existing Radiant software Programmer project as described in "Opening an Existing Radiant Software Programmer Project" on page 271.
2. Right-click on the row corresponding to the device you would like to edit, and choose **Edit > Device Properties**.
3. In the Operation dropdown list, chose **SPI Flash Unlock Device**.
4. Ensure that the **Secure SPI Flash Golden Pattern Sectors** box is unchecked.
5. Click **OK.**
6. In the Radiant software Programmer, choose **Run > Program,** or click  on the toolbar.

The SPI flash is now unlocked and can be reprogrammed.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Configuring SPI Flash Options

You can use the Device Properties dialog box to configure SPI Flash options for Ultra Plus devices.

**To configure SPI Flash Options:**

1. In Programmer, select the device that you want to edit.
2. Choose **Edit > Device Properties**, or click the  button on the toolbar, or right-click and choose **Device Properties** to display the Device Properties dialog box.
3. In the Target Memory pulldown menu, choose **External SPI Flash Memory (SPI FLASH)**.
4. In the Port Interface pulldown menu, choose **SPI**.
5. In the Access Mode pulldown menu, choose **Direct Programming**.
6. the Options pulldown menu, chose the desired SPI Flash option.
7. In the Programming File box, browse to and select the programming file (.bit, .rbt, .bin, .hex, .mcs, .exo, or .xtek).
8. Select the desired **Family**, **Vendor**, **Device**, and **Package** options.
9. Click **Load from File** to select the data size entered. You can change this size by typing a different file size. This feature is useful if you only want to use a portion of the file for the operation. If you change the file size, it must be no larger than the full file size or the device size.
10. Select start address and end address from the **Start Address (Hex)** and **End Address (Hex)** dropdown menus. For the AMD parallel flash, the starting address is automatically selected and cannot be changed.
11. To erase the flash device if a verify error occurs, click, the **Erase SPI Part on Programming Error** box.
12. Click **OK.**

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258
- "Device Properties Dialog Box" on page 285
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Changing the Port Assignment

When you first launch the Programmer software with the cables properly connected, it connects to the first available port that it detects. You can change the connecting port, as well as other cable options, using the Cable and Port Setting dialog box.

**To change the port setup:**

1. In Programmer, choose **View > Cable Setup**.
2. In the Cable Setup dialog box, choose the options that you want. Changes are immediately applied.

   If the cable is not connected or cannot be detected (if the board power in not on, for example), Programmer software displays an error message.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258
- "Cable Setup Dialog Box" on page 290
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Testing the Cable Signal

The Cable Signal Tests dialog box helps you debug or improve the cable signal. You can use an oscilloscope, while running the continuous loop feature, to test your printed circuit board.

To test the cable signal:

1. In Programmer, choose **View > Cable Setup**.
2. In the Cable Setup dialog box, click the **Debug Mode** button.

   The Cable Signal Tests dialog box opens.
3. In the dialog box, click **Power Check** to test the connection.

   The VCC Status LED will blink green if the cable and power are detected. It will produce an error message and blink red if the cable or power is not detected.
4. In the dialog box, select an option for the pins you want to test. The options are defined as follows:
   - **Toggle** – Alternates between logic 1 and logic 0.
   - **Hold High** – Holds at logic 1.
   - **Hold Low** – Holds at logic 0.
   - **Read (TDO only)** – Reads back the data from TDO and records the read back data in the Programmer log file.

   **Note**

   The options available for cable signal testing varies, depending on the type of cable you are using and the options you have selected in the Cable Setup dialog box. Options not available are grayed out.
5. To test the settings you have selected, click **Test**.

This causes the pins that have a Toggle setting to toggle once from logic 1 to logic 0 to logic 1.

6. To perform continuous testing, click **Loop Test**.

   This causes the pins that have a Toggle setting to toggle continuously until you stop the process.

7. To stop the loop process, press **ESC**.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Cable Signal Tests Dialog Box" on page 292
- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

## Viewing the Log File

An ASCII text log file summarizes the results of the Programmer operations.

**To view the log file:**

- Click the  button on the toolbar. An ASCII text file opens showing the contents of the log.

**See Also** ▶"Using the Radiant Software Programmer" on page 267

- "Programming the FPGA" on page 258
- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Programmer Options

This section lists the options available in Programmer.

Topics include:

# Getting Started Dialog Box

The following options are available in the Getting Started dialog box:

**Project Name** Specifies the name of the project.

**Project Location:** Specifies the location of the project.

**Create a New Project from a Scan** Scans the attached chain with the last used cable settings.

**Cable** Specifies the download cable type:

- **HW-USBN-2A -** (Lattice HW-USBN-2A USB port programming cable)
- **HW-USBN-2B (FTDI)** - (Lattice HW-USBN-2B (FTDI) USB programming cable)
- **HW-DLN-3C (Parallel)** - (Lattice HW-DLN-3C parallel programming cable)

**Port** Specifies the serial port to which the download cable is connected. Select the port from the list or click Detect Cable.

**Detect Cable** Automatically detects where the download cable is connected.

**Blank Programmer Project** Creates a new blank project.

**Open an Existing Programmer Project** 1.Allows the user to browse to an existing .xcf configuration file and open an existing Programmer project.

**See Also** "Programmer Options" on page 284

- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Device Properties Dialog Box

The Device Properties dialog box consists of two tabs: General Tab and Device Information Tab.

### General Tab

The following options are available in the Device Properties dialog box General tab, depending on selected Access Mode and Operation.

**Target Memory** Target memory is the storage memory type on the FGPA. For example, the following table lists Target Memory options for iCE40UP.

**Table 25: Target Memory**

| Target Memory | Description |
|---|---|
| Compressed Random Access Memory (CRAM) | The memory type used for CRAM configuration. |
| Non Volatile Configuration Memory (NVCM) | This memory type is used to program the non-volatile configuration memory (NVCM) while the device is not operational. |
| External SPI Flash Memory (SPI FLASH) | Serial peripheral interface (SPI) memory flash memory. |

**Port Interface** The following table lists Port Interface for iCE40UP.

**Table 26: Port Interface**

| Target Memory | Description |
|---|---|
| Slave SPI | The four-wire serial peripheral interface (SPI) communications protocol. |
| SPI | Serial peripheral interface (SPI) communications protocol. |

**Access Mode** Selects the mode for programming the device. For example, iCE40UP supports the following access mode.

**Table 27: Access Mode**

| Access Mode | Description |
|---|---|
| Direct Programming | The mode is used for direct programming while the device is not in operation. |

**Operation** Lists the available operation modes for the device. Select one from the pulldown list. For example, iCE40UP supports the following operations.

**Table 28: Operation**

| Operation | Description |
|---|---|
| Blank Check | Checks if the device is erased (blank). |
| CRAM Read and Save | Shifts in a bitstream by directly loading the iCE40 CRAM over the SPI bus. The contents in memory are then read and saved to an output file. |
| Erase, Program, Verify | Erases, programs, and verifies the new configuration memory. For devices with a Feature Row, this operation will erase, program, and verify the Feature Row in direct programming mode. It will not erase and verify the Feature Row in background programming mode (XFLASH). |
| Fast Configuration | Shifts in a bitstream directly into device and the device takes care of the rest for configuration. |

**Table 28: Operation (Continued)**

| Operation | Description |
|---|---|
| Fast Program, Read and Save | Shifts in a bitstream directly into device and the device takes care of the rest for configuration. The contents in memory are then read and saved to an output file. |
| Fast Program, Read and Save without DONE bit | Shifts in a bitstream directly into device and the device takes care of the rest for configuration, but without the DONE bit. The contents in memory are then read and saved to an output file. |
| Program Only | Programs a new configuration into memory. Issues programming instructions and then the data is taken from the data file. |
| Program, Verify | Programs the memory of the device and then verifies the contents of that memory. |
| Program, Verify, Secure | Programs the memory of a device, verifies the contents of memory, secures the configuration block. Secure inhibits readback of the device. |
| Read DONE bit | Reads to see if the DONE bit has or has not been set. |
| Read Device Properties | Reads the properties of the device. |
| Secure Device | Secures the configuration block. Secure inhibits readback of the device. |
| Verify ID | Verifies the ID of the device. |
| Verify Only | Verifies the new configuration memory. |

### Device Information Tab

The Device Information tab displays information on the device selected. The content can vary depending upon the device selected, and can include:

- Family Name
- Device Name
- VCC Voltage Supply
- Programming Pins
- JTAG IDCODE
- JTAG IDCODE MASK
- IDCODE Length
- SOFT IDCODE
- Maximum Erase Pulse Width (ms)
- Maximum Program Pulse Width (ms)

**See Also** ▶"Programmer Options" on page 284

- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Edit Custom Device Dialog Box

The following options are available in the Edit Custom Device dialog box:

**Device Family** Dropdown list of custom device families.

**Device** The name that was assigned to the device.

**Package** The name that was assigned to the package.

**Device Description** The description that was assigned to the device.

**See Also** ▶"Programmer Options" on page 284

- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Custom SPI Flash Dialog Box

The following options are available in the Custom SPI Flash dialog box:

**Device Family** Lists the device family being used.

**Device Description** The name that was assigned to the device.

**Device Name** This can be any alpha-numeric string that describes the device name (for example: SPI-M25P32).

**Package** This can be any alpha-numeric string that describes the package (for example: 16-pin SOIC).

**Device Vendor** The vendor of the custom SPI Flash device.

**Device Density** The size of the Serial SPI Flash device (512KBits - 256MB).

**Device ID** This can be any alpha-numeric string that describes the device ID (for example: 0x15).

**Byte Per Sector** Allows you to choose desired bytes per sector value from dropdown.

**Protection Options ON** If the SPI Flash device supports sector protection, check this box to enable protection options. This will allow you to select **Secure SPI Flash Golden Pattern Sectors** in the "Device Properties Dialog Box" on page 285.

**See Also** ▶"Programmer Options" on page 284

- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Settings Dialog Box

The following options are available in the Settings dialog box:

## General Tab

**At Programmer Start-Up** Allows you to select between showing the Programmer Getting Started dialog box at start-up, or opening the last Programmer project at start-up.

**USERCODE Display** Allows the user to choose in which format the USERCODE will be displayed. Choices are Hex, ASCII, and Decimal.

**Device Family Selection List Order** Allows the user to display the Device Family pulldown list in the Programmer main window in either chronological or alphabetical order. The default is chronological order.

**Log File Path** Shows the location of the Programmer log file, and allows you to browse to and set a new location for the Programmer.log file.

**Clear Log File Each Time Application Starts** Clears log file each time the application starts: If the box is unchecked, the log file will continue increase in size until the file is manually erased.

## Programming Tab

**Sequential Mode** Programs all the devices on the board one at a time. If an Operation Override is selected, all Operation descriptions in the chain file are temporarily changed to the override setting.

**Use Default JTAG States (TLR/TLR)** Uses Test-Logic-Reset (TLR) as the starting and ending JTAG state of the JTAG State Machine for download.

**Avoid Test Logic Reset (TLR) State** Avoids the Test Logic Reset State of the JTAG State Machine during download.

**Use Custom JTAG States** Specify Test-Logic-Reset (TLR) or Run-Test/Idle (RTI) as the starting and ending JTAG state of the JTAG State Machine for download.

**Initial TAP State** Specify TLR or RTI as the starting JTAG state of the JTAG State Machine for download.

**Final TAP State** Specify TLR or RTI as the ending JTAG state of the JTAG State Machine for download.

**Check Cable Setup Before Programming** Confirms that the cable signals are correctly connected to the board and devices in the JTAG chain on the board match the devices selected in the .xcf file.

**Continue Download on Error** Ignores any errors while downloading and continues running.

**See Also** ▶"Programmer Options" on page 284

- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Cable Setup Dialog Box

The following options are available in the Cable Setup dialog box:

**Detect Cable** Automatically detects where the download cable is connected.

**Cable** Specifies the download cable type:

- **HW-USBN-2A -** (Lattice HW-USBN-2A USB port programming cable)
- **HW-USBN-2B (FTDI)** - (Lattice HW-USBN-2B (FTDI) USB programming cable)
- **HW-DLN-3C (Parallel)** - (Lattice HW-DLN-3C parallel programming cable)

**Port** Specifies the serial port to which the download cable is connected. Select the port from the list or click Detect Cable.

**Custom Port** Specifies a custom parallel port to which the download cable is connected. Use hexadecimal format to type the port name.

**Use Default Clock Divider** Uses the fastest TCK clock speed.

**Use Custom Clock Divider** Enables the Use Custom Clock Divider feature.

**TCK Divider Setting (0 - 30x)** Allows you to slow down the TCK clock. This is done by extending the low period of the clock. Refer the following tables for specific frequency settings for USB-2B (2232H FTDI USB host chip), USB-2B (2232D FTDI USB host chip), and USB-2A and Parallel port cables.

**Table 29: USB-2B (2232H FTDI USB host chip)**

| Divider | Clock Frequency[1] | Divider | Clock Frequency[1] |
|---|---|---|---|
| 0 | 30 MHz | 5 | 5 MHz |
| 1 | 15 MHz (Default) | 6 | 4.2 MHz |
| 2 | 10 MHz | 7 | 3.7 MHz |
| 3 | 7.5 MHz | 8 | 3.1 MHz |
| 4 | 6 MHz | 9 | 3 MHz |
| | | 10 | 2.7 MHz |

[1]Calculation formula for USB-2B (2232H FTDI USB host chip):

Frequency = 60 MHz / (1 + ClockDivider) *2

**Table 30: USB-2B (2232D FTDI USB host chip)**

| Divider | Clock Frequency[2] | Divider | Clock Frequency[2] |
|---|---|---|---|
| 0 | 6 MHz | 5 | 1 MHz |
| 1 | 3 MHz (Default) | 6 | 0.8 MHz |
| 2 | 2 MHz | 7 | 0.7 MHz |
| 3 | 1.5 MHz | 8 | 0.65 MHz |
| 4 | 1.2 MHz | 9 | 0.6 MHz |
| | | 10 | 0.5 MHz |

[2]Calculation formula for USB-2B (2232D FTDI USB host chip):

Frequency = 12 MHz / (1 + ClockDivider) *2

**Table 31: USB-2A and Parallel port**

| Divider | Low Pulse Width delay[3] | Divider | Low Pulse Width delay[3] |
|---|---|---|---|
| 0 | NA | 5 | 5x |
| 1 | 1x | 6 | 6x |
| 2 | 2x | 7 | 7x |
| 3 | 3x | 8 | 8x |
| 4 | 4x | 9 | 9x |
| | | 10 | 10x |

[3]The USB-2A frequency fixed at 1.5 MHz and Parallel Port frequency fixed at 500 KHz

**Use Default I/O Settings** Only use the four JTAG signals.

**Use Custom I/O Settings** Specify additional non-JTAG signals connected to the board.

**INITN Pin Connected** Select this option if you connect the INIT pin to the download cable. This option is only available with the Lattice USB port programming cable.

**Done Pin Connected** Select this option if you connect the DONE pin to the download cable. This option is not available for the Lattice HW-DLN-3C parallel programming cable.

**TRST Pin Connected** Select this option if you connect the TRST pin to the download cable. Specify active high or active low.

**PROGRAMN Pin Connected** Select this option if you connect the PROGRAMN pin to the download cable.

**ispEN Pin Connected** Select this option if you connect the ispEN pin to the download cable. Specify active high or active low.

**See Also** ▶"Programmer Options" on page 284

- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Cable Signal Tests Dialog Box

The following options are available in the Cable Signal Tests dialog box:

**Toggle** Alternates between logic 1 and logic 0.

**Hold High** Holds at logic 1.

**Hold Low** Holds at logic 0.

**Test** Applies the selected settings to the selected pin connections. This causes the pins that have a toggle setting to toggle once from logic 1 to logic 0 to logic 1.

**Loop Test** Performs continuous testing, causing the pins that have a toggle setting to toggle continuously until you click ESC.

**View Log** Opens the log file, an ASCII text file that summarizes the results of the Cable Signal Test operations.

**Power Check** Tests the cable and power connections, causing the VCC Status LED to blink green or red to indicate that cable and power are detected or not detected.

**VCC Status** Shows the status of the cable and power connection. The LED blinks green when the cable and power are detected. It blinks red and produces an error message when cable or power is not detected.

**Toggle Delay** Sets the delay after each operation. Default is 0.

**Number of Bytes** Sets the number of bytes read by each operation. Default is 0.

**See Also** ▶"Programmer Options" on page 284

- "Programming and Configuring iCE40 Devices with Programmer" on page 293

# Programming and Configuring iCE40 Devices with Programmer

The Radiant software Programmer supports programming and configuration of iCE40 devices.

There are three basic flows for programming or configuring iCE40 devices:

- iCE40 CRAM Configuration Flow using .bin and .hex files output from the Radiant software. The default is .bin.

  CRAM configuration is accomplished by directly loading the iCE40 CRAM over the SPI bus.
- iCE40 NVCM Programming Flow using Nonvolatile Configuration Memory (.nvcm) files output from the Radiant software.

  NCVM programming involves transmitting programming data over the SPI bus to the NVCM array internal to the iCE40 device. The NVCM is one-time programmable (OTP).
- iCE40 SPI Flash Programming Flow using a separate SPI Flash device to configure an iCE40 device.

  In this flow, the iCE40 device acts as the SPI bus master and will therefore control the data flow from the configuration device.

**See Also** ▶Configuring an iCE40 Device Using the CRAM Flow

- Programming an iCE40 Device Using the NVCM Flow
- Programming an iCE40 Device Using SPI Flash Device Flow
- Using the Radiant Software Programmer
- Lattice Radiant Software User Guide
- TN1248 - iCE40 Programming and Configuration

## Configuring an iCE40 Device Using the CRAM Flow

iCE40 CRAM Configuration Flow uses .hex files, .bin files, or Intel-Hex files output from the Radiant software. CRAM configuration is accomplished by directly loading the iCE40 CRAM over the SPI bus.

When using the Radiant software Programmer to configure an iCE40 device, you must first create a new Programmer project, or open an existing Programmer project. By creating a new Programmer project, you create a chain file (.xcf).

**To create a new Programmer project:**

1. Make sure that the board is turned on and that the Lattice USB programming cable is properly connected to the computer.

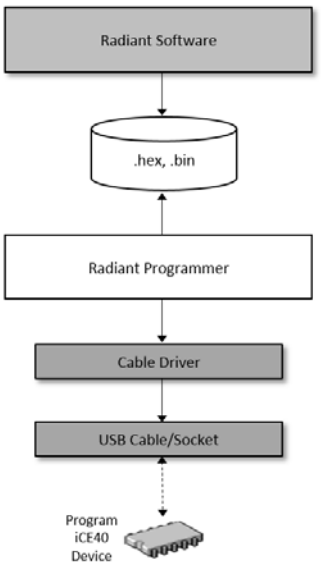
2. Issue the start command.
   - In Windows, go to the Windows Start menu and choose **All Programs > Lattice Radiant Software> Accessories > Radiant Programmer.**
   - In Linux, from the *<install_path>/programmer/bin/lin64* directory, enter the following on a command line:

     ```
     ./programmer
     ```
3. In the Getting Started dialog box, enter a project name in the **Project Name** box.
4. Browse to the location where you wish your project to reside in the **Project Location** box.
5. Choose one of the following:
   - **Create a new project from a scan.** This option creates a new project from a scan.
     - If you have a board connected to your computer, you can click the **Detect Cable** button to detect the cable and port. Otherwise, you can manually select cable and port from the **Cable** and **Port** pulldown lists.
   - **Create a New Blank Project** - This option creates a new blank the Radiant software Programmer project.
   - Click **OK**.
6. In Programmer:
   - Select the iCE40 family in the Device Family dropdown menu.
   - Select the desired iCE40 device in the Device dropdown menu.
7. Choose **Edit > Device Properties**, or click the  button on the toolbar, or right-click and choose **Device Properties**.
8. In the Device Properties dialog box:
   - In the Target Memory pulldown menu, choose **Compressed Random Access Memory (CRAM)**.
   - In the Port Interface pulldown menu, choose **Slave SPI**.
   - In the Access Mode pulldown menu, choose **Direct Programming**.
   - In the Operation pulldown menu, choose the desired operation, as described in the Device Properties dialog box.
9. In the Programming File box, browse to the Radiant software-generated programming file (.hex or .bin).
10. Click **OK** to close the Device Properties dialog box.

After you have specified all of the processing options, use the Program command to download the design file to the iCE40 device.

- In Programmer, choose **Run > Program,** or click  on the toolbar.

The diagram below shows the iCE40 CRAM configuration flow




**See Also**
- "Programming and Configuring iCE40 Devices with Programmer" on page 293
- "Getting Started Dialog Box" on page 285
- "Device Properties Dialog Box" on page 285
- Using the Radiant Software Programmer
- TN1248 - iCE40 Programming and Configuration
- Lattice Radiant Software User Guide

# Programming an iCE40 Device Using the NVCM Flow

iCE40 NVCM Programming Flow uses Nonvolatile Configuration Memory (.nvcm) files output from the Radiant software.

When using the Radiant software Programmer to configure an iCE40 device, you must first create a new Programmer project, or open an existing Programmer project. By creating a new Programmer project, you create a chain file (.xcf).

To create a new Programmer project:

1. Make sure that the board is turned on and that the Lattice USB programming cable is properly connected to the computer.
2. Issue the start command.
   - In Windows, go to the Windows Start menu and choose **All Programs > Lattice Radiant Software > Accessories > Radiant Programmer.**

- In Linux, from the *<install_path>/programmer/bin/lin64* directory, enter the following on a command line:

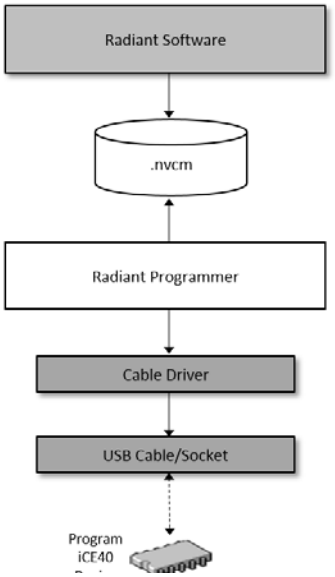
  ```
  ./programmer
  ```

3. In the Getting Started dialog box, enter a project name in the **Project Name** box.
4. Browse to the location where you wish your project to reside in the **Project Location** box.
5. Choose one of the following:
   - **Create a new project from a scan.** This option creates a new project from a scan.
     - If you have a board connected to your computer, you can click the **Detect Cable** button to detect the cable and port. Otherwise, you can manually select cable and port from the **Cable** and **Port** pulldown lists.
   - **Create a New Blank Project** - This option creates a new blank Radiant software Programmer project.
   - Click **OK**.
6. In Programmer:
   - Select the iCE40 family in the Device Family dropdown menu.
   - Select the desired iCE40 device in the Device dropdown menu.
7. Choose **Edit > Device Properties**, or click the  button on the toolbar, or right-click and choose **Device Properties**.
8. In the Device Properties dialog box:
   - In the Target Memory pulldown menu, choose **Non Volatile Configuration Memory (NVCM)**.
   - In the Port Interface pulldown menu, choose **Slave SPI**.
   - In the Access Mode pulldown menu, choose **Direct Programming**.
   - In the Operation pulldown menu, choose the desired operation, as described in the Device Properties dialog box.
9. In the Programming File box, browse to the Radiant software-generated NVCM programming file (.nvcm).
10. Click **OK** to close the Device Properties dialog box.

After you have specified all of the processing options, use the Program command to download the design file to the iCE40 device.

In Programmer, choose **Run > Program,** or click  on the toolbar.

The diagram below shows the iCE40 NVCM programming flow




**See Also** ▶"Programming and Configuring iCE40 Devices with Programmer" on page 293

- "Getting Started Dialog Box" on page 285
- "Device Properties Dialog Box" on page 285
- Using the Radiant Software Programmer
- Lattice Radiant Software User Guide
- TN1248 - iCE40 Programming and Configuration

# Programming an iCE40 Device Using SPI Flash Device Flow

iCE40 SPI Flash Programming Flow involves using a separate SPI Flash device to configure an iCE40 device.

When using the Radiant software Programmer to configure an iCE40 device, you must first create a new Programmer project, or open an existing Programmer project. By creating a new Programmer project, you create a chain file (.xcf).

**To create a new Programmer project:**

1. Make sure that the board is turned on and that the Lattice USB programming cable is properly connected to the computer.
2. Issue the start command.
   - In Windows, go to the Windows Start menu and choose All **Programs > Lattice Radiant Software > Accessories > Radiant Programmer.**

- In Linux, from the *<install_path>/programmer/bin/lin64* directory, enter the following on a command line:

  ```
  ./programmer
  ```

3. In the Getting Started dialog box, enter a project name in the **Project Name** box.
4. Browse to the location where you wish your project to reside in the **Project Location** box.
5. Choose one of the following:
   - **Create a new project from a scan.** This option creates a new project from a scan.
     - If you have a board connected to your computer, you can click the **Detect Cable** button to detect the cable and port. Otherwise, you can manually select cable and port from the **Cable** and **Port** pulldown lists.
   - **Create a New Blank Project** - This option creates a new blank Radiant software Programmer project.

   **Note**

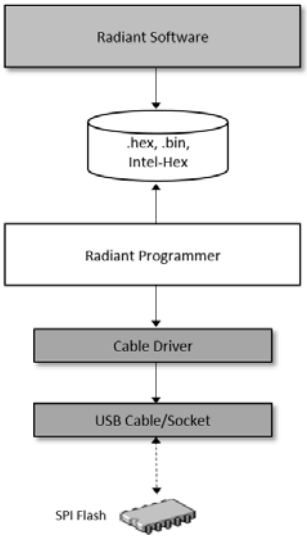
   Because iCE40 devices are Slave SPI, and not JTAG, Programmer does not support the "**Create a New Project from a Scan**" option for iCE40 devices.

6. Click **OK**.
7. In Programmer:
   - Select the iCE40 family in the Device Family dropdown menu.
   - Select the desired iCE40 device in the Device dropdown menu.
8. Choose **Edit > Device Properties**, or click the  button on the toolbar, or right-click and choose **Device Properties**.
9. In the Device Properties dialog box:
   - In the Target Memory pulldown menu, choose **External SPI Flash Memory (SPI FLASH)**.
   - In the Port Interface pulldown menu, choose **SPI**.
   - In the Access Mode pulldown menu, choose **Direct Programming**.
   - In the Operation pulldown menu, choose the desired operation, as described in Device Properties dialog box.
10. In the Programming File box, browse to the Radiant software-generated programming file (.hex or .bin).
11. In the Advanced Flash Options box, select **Family**, **Vendor**, **Device**, and **Package**.
12. Click **Load from File** to fill in the Data File Size box. You can change this size by typing a different file size. This feature is useful if you only want to use a portion of the file for the operation. If you change the file size, it must be no larger than the full file size or the device size.

13. Select start address and base address from **Start Address (Hex)** and **Base Address (Hex)** dropdown menus.
14. If you want to erases the flash device if a verify error occurs, click, the **Erase SPI Part on Programming Error** box.
15. Click **OK**.

After you have specified all of the processing options, use the Program command to download the design file to the SPI Flash device.

In Programmer, choose **Run > Program,** or click  on the toolbar.The diagram below shows the iCE40 SPI Flash programming flow.




**See Also** ▶"Programming and Configuring iCE40 Devices with Programmer" on page 293

- "Adding a Custom SPI Flash Device" on page 278
- "Editing a Custom SPI Flash Device" on page 279
- "Removing a Custom SPI Flash Device" on page 280
- "Programming Using Custom SPI Flash Device" on page 280
- "Getting Started Dialog Box" on page 285
- "Device Properties Dialog Box" on page 285
- Using the Radiant Software Programmer
- Lattice Radiant Software User Guide
- TN1248 - iCE40 Programming and Configuration

# Deploying the Design with the Deployment Tool

The Radiant software Deployment Tool allows you to generate files for deployment for single devices, a chain of devices, and can also convert data files to other formats and use the data files it produces to generate other data file formats.

Deployment Tool is a stand-alone tool available from the Radiant software Accessories. The Deployment Tool graphical user interface (GUI) is separate from the Radiant software design environment.

A four-step wizard allows you to select deployment type, input file type, and output file type.

**See Also** ▶"Deployment Function Types" on page 300

- "iCE40 Warm Boot/Cold Boot Bitstream Output Options" on page 301
- "Creating a New Deployment" on page 302
- "Opening an Existing Deployment" on page 303
- "Opening an Existing Deployment While Running the Deployment Tool" on page 304
- "Creating a New Deployment While Running the Deployment Tool" on page 304
- "Using Quick Launch Button to Create a New Deployment" on page 305
- "Viewing the Deployment Tool Log File" on page 306
- "Changing the Deployment Tool Log File Settings" on page 306

## Deployment Function Types

This section provides tables that list device family, input file types, output file extensions, and options for each available deployment function type.

Click on the list items below to jump to topics that contain lists of device family, input file types, output file type/extensions, and options for each deployment function type.

- "File Conversion Deployment Function Type" on page 300
- "Tester Deployment Function Type" on page 301
- "External Memory Deployment Type" on page 301

### File Conversion Deployment Function Type

Not supported for iCE40UP devices.

## Tester Deployment Function Type

Not supported for iCE40UP devices.

## External Memory Deployment Type

See Table 32 for input file types, output file extensions and options for File Conversion deployment function type for the iCE40 device family.

### Advanced SPI Flash Options -- iCE40 Warm Boot/Cold Boot

The following table lists options -- iCE40 Warm Boot/Cold Boot -- for advanced SPI Flash deployment.

**Table 32: Device Family Types**

| Device Family | Enable Cold Boot | | Number of Patterns | | | | | |
|---|---|---|---|---|---|---|---|---|
| iCE40 | **Off (Default)** | **On** | **Number of Patterns** | **Pattern 1** | **Starting Address 1** | **Pattern 2 - 3** | **Pattern 4** | **Starting Address 4** |
| | **POR Boot Pattern** | | 2/3/4 | Selected Pattern 1. | 0x010000 Sectors. | User Data 2 - 3. | Selected Pattern 4. | 0x010000 Sectors. |
| | 1/2/3/4 | Cold Boot | | | | | | |

# iCE40 Warm Boot/Cold Boot Bitstream Output Options

The following output format options are available for generating iCE40 warm boot/cold boot bitstreams. Except where otherwise noted, the default setting is listed first for each option.

### Intel Hex/Motorola Hex/Extended Tektronix Hex

**SPI Flash Size: 1 Mb (512 Kb - 256 Mb)**: Possible PROM sizes are 512 Kb, 1 Mb, 2 Mb, 4 Mb, 8 Mb, 16Mb, 32 Mb, 64 Mb, 128 Mb, and 256 Mb.

**Byte Wide Bit Mirror: OFF/ON**: Flips each byte. Default: bytes are not flipped. Optional.

**Number of Patterns: 2 (2 - 4)**: Specifies the number of iCE40 configuration files.

**Enable Cold Boot: OFF/ON**: Optional. Default is warm boot. If selected, cold boot is chosen.

**Power On Reset Boot Image 1 (1 - 4)**: Indicates the input configuration file path and name. Must be specified for each configuration file.

**Pattern (1 - 4) Starting Address**: SPI flash address location for the configuration file. Must be specified for each configuration file.

**See Also** ▶External Memory Deployment Type

- Creating a New Deployment

# Creating a New Deployment

The Deployment Tool has a wizard interface that guides you through the process of creating a deployment for the various function types and output file types.

**To create a new Deployment:**

1. Issue the start command in one of the following ways:
   - In the Radiant software Programmer main window, choose **Tools > Deployment Tool**.
   - In Windows, go to the Windows Start menu and choose **All Programs Lattice Radiant Software > Accessories > Radiant Programmer > Deployment Tool.**
   - In Linux, from the <*Programmer install path*>/bin/lin*64* directory, enter the following on a command line:

     ```
     ./deployment
     ```
2. In the Getting Started dialog box, choose **Create a New Deployment.**
3. In the Function Type dropdown menu, choose from one of the following options:
   - File Conversion
   - Tester
   - External Memory
4. In the Output File Type box, choose an output file type.
5. Click **OK**. The Deployment Tool loads the device database. A four step process must now be completed.
6. In “Step 1”, do one of the following, depending on the type of file you are using:
   - If you are creating a deployment from an .xcf file, use the ... **(Browse)** button to browse to the project’s .xcf file.
   - If you are creating a deployment from a data file, select Input File(s) window, click **... (Browse)** and browse to the projects .jed file. The Device Family and Device boxes are automatically populated by the Deployment Tool.

7. Click **Next**.
8. In “Step 2” (Options dialog box), select the desired options for your output file type.
9. Click **Next**.
10. In the “Step 3” (Select Output File(s) dialog box), select the desired output file type from the dropdown menu, and using the using the ... button, browse to the desired location of the output file(s).
11. Click **Next**.
12. In the “Step 4” (Generate Deployment dialog box), review the Deployment Tool Summary and Command Line Information.
13. Choose **File > Generate**, or click the button, to perform the Deployment. Results of the generation are available for review.
14. Choose **File > Save**, or click the button, to save the deployment (.ddt) file to your desired location.

**See Also** “iCE40 Warm Boot/Cold Boot Bitstream Output Options” on page 301

# Opening an Existing Deployment

If a deployment was saved as a deployment (.ddt) file, the deployment can be opened in Deployment Tool and regenerated using the same deployment options, or by applying different deployment options.

**To open an existing deployment:**

1. Issue the start command in one of the following three ways:
   - In the Programmer main window, choose **Tools > Deployment Tool**.
   - In Windows, go to the Windows Start menu and choose **All Programs Lattice Radiant Software > Accessories > Radiant Programmer > Deployment Tool.**
   - In Linux, from the <*Programmer install path*>/bin/lin*64* directory, enter the following on a command line:

     ```
     ./deployment
     ```
2. In the Getting Started dialog box, choose **Open an Existing Deployment.**
3. Click **OK**. The Deployment Tool loads the device database. A four step process must now be completed.
4. In “Step 1”, do one of the following, depending on the type of file you are using:
   - If you are creating a deployment from an .xcf file, use the ... **(Browse)** button to browse to the project’s .xcf file.
   - If you are creating a deployment from a data file, select Input File(s) window, click **... (Browse)** and browse to the projects .jed file. The

Device Family and Device boxes are automatically populated by the Deployment Tool.

5. Click **Next**.
6. In “Step 2” (Options dialog box), select the desired options for your output file type.
7. Click **Next**.
8. In the “Step 3” (Select Output File(s) dialog box), select the desired output file type from the dropdown menu, and using the using the ... button, browse to the desired location of the output file(s).
9. Click **Next**.
10. In the “Step 4” (Generate Deployment dialog box), review the Deployment Tool Summary and Command Line Information.
11. Choose **File > Generate**, or click the button, to perform the Deployment. Results of the generation are available for review.
12. Choose **File > Save**, or click the button, to save the deployment (.ddt) file to your desired location.

**See Also** ▶“Creating a New Deployment” on page 302

▶ “iCE40 Warm Boot/Cold Boot Bitstream Output Options” on page 301

# Opening an Existing Deployment While Running the Deployment Tool

You can open a different existing deployment while running the Deployment Tool, and save the deployment you are currently working on.

**To open an existing deployment while running the Deployment Tool:**

1. Choose **File > Open**.
2. In the Open dialog box, browse to the deployment (.ddt) file that you wish to open and select it.
3. Click **Open**.

**See Also** ▶“Creating a New Deployment” on page 302

▶ “iCE40 Warm Boot/Cold Boot Bitstream Output Options” on page 301

# Creating a New Deployment While Running the Deployment Tool

You can create a new deployment while running the Deployment Tool.

**To create a new deployment while running the Deployment Tool:**

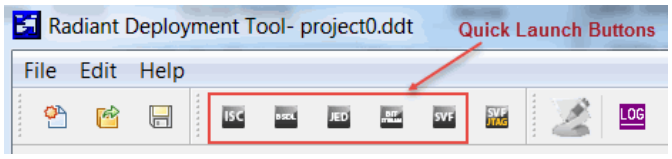
1. Choose **File > New Project**, or click the button.
2. Enter a name for your new programmer project
3. Use the Browse button to select a location to store your project.
4. Select one of the following project sources:
   - Scan: scan a device by selecting the appropriate cable and port to for the desired board.
   - Blank Programmer Project
5. Click **OK**. The new project is loaded.

**See Also** "Creating a New Deployment" on page 302

- "Using Quick Launch Button to Create a New Deployment" on page 305
- "iCE40 Warm Boot/Cold Boot Bitstream Output Options" on page 301

# Using Quick Launch Button to Create a New Deployment

Quick launch buttons allow you to create a new Deployment while running the Deployment Tool without having to use the Getting Started dialog box. The toolbar supports up to six quick launch buttons.




You can customize which quick launch buttons appear in the toolbar. Choices of quick launch buttons include:

- IEEE 1532 ISC Data File
- Application Specific BSDL File
- JEDEC File
- Bitstream
- JEDEC to Hex
- SVF - Single Device
- SVF - JTAG Chain
- STAPL - Single Device
- STAPL - JTAG Chain
- ATE

- Hex Conversion
- Dual Boot
- Advanced SPI Flash
- sysCONFIG Daisy Chain

**To customize which quick launch buttons appear in the Deployment Tool toolbar:**

1. Place the cursor over a quick launch button, and right-click.
2. In the drop-down menu, check the box of the quick start button that you want to display in the toolbar. A maximum of six buttons may be selected.

**See Also** ▶"Creating a New Deployment" on page 302

- "Creating a New Deployment While Running the Deployment Tool" on page 304
- "iCE40 Warm Boot/Cold Boot Bitstream Output Options" on page 301

# Viewing the Deployment Tool Log File

The log file shows recent actions in a text editor.

**To view the log file:**

Click  button on the toolbar.

**See Also** ▶"Deploying the Design with the Deployment Tool" on page 300

# Changing the Deployment Tool Log File Settings

The log file shows recent actions in a text editor. You can change settings to clear log file path each time the application starts, and you can also change where the log file is saved.

To change where the log file is saved:

1. Choose **Edit > Settings**.
2. In the Log File Path dialog box, browse to the folder where you wish to save the deployment_tool.log file.

**To clear the log file each time the application starts:**

1. Choose **Edit > Settings**.
2. Check the **Clear Log File Each Time Application Starts** box.

If the **Clear Log File Each Time Application Starts** box is unchecked, the log file will continue increase in size until the file is manually erased.

**See Also** ▶"Deploying the Design with the Deployment Tool" on page 300

# Debugging SVF, STAPL, and VME Files

Download Debugger is a stand-alone software tool for debugging the following file types:

- Serial Vector Format (SVF)
- Standard Test And Programming Language (STAPL)
- Lattice Embedded (VME)

Download Debugger allows you to program a device, and edit, debug, and trace the process of SVF, STAPL, and VME files.

Download Debugger also allows you to create, edit, or view a VME file in hexadecimal format.

This section provides procedures for using Download Debugger. Topics include:

# Understanding SVF Files

The Serial Vector Format file (.svf) is the medium for exchanging descriptions of high-level 1149.1 bus operations. The SVF file is defined as an ASCII file that consists of a set of SVF statements. The 1149.1 bus operations consist of scan operations and movements between deferent stable states. Refer to the Serial Vector Format Specification Rev E for detailed definitions of SVF statement formats. Current SVF specifications are available from Asset-Intertech website at www.asset-intertech.com.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Download Debugger Software Support of SVF Operations

The Download Debugger supports a selective set of SVF operations. The following table lists SVF operations supported by the Download Debugger.

| SVF Operation | Description | Support Status |
|---|---|---|
| ENDDR | Specifies default end state for DR scan operations. | Full support |
| ENDIR | Specifies default end state for IR scan operations. | Full support |
| HDR | (Header Data Register) Specifies a header pattern, which is placed at the beginning of subsequent DR, scan operations. | Supports TDI keyword only. SMASK, TDO and MASK are not supported. |
| HIR | (Header Instruction Register) Specifies a header pattern, which is placed at the beginning of subsequent IR, scan operations. | Supports TDI keyword only. SMASK, TDO and MASK are not supported. |
| RUNTEST | Forces the 1149.1 bus to the RUN_TEST/IDLE state for a specified number of clocks. | Full support |
| SDR | (Scan data register) Performs an 1149.1 data register scan. | Supports TDI, TDO, and MASK keywords. SMASK is ignored. |
| SIR | (Scan Instruction Register) Performs an 1149.1 instruction register scan. | Supports TDI, TDO, and MASK keywords. SMASK is ignored. |
| STATE | Forces the 1149.1 bus to a specified stable state. | Supports only single target state. The Download Debugger does not allow you to specify custom traverse path. |
| TDR | (Trailer Data Register) Specifies a trailer pattern, which is appended to the end of subsequent DR scan operations. | Supports TDI keyword only. SMASK, TDO, and MASK are not supported. |
| TIR | (Trailer Data Register) Specifies a trailer pattern which is appended to the end of subsequent IR scan operations | Supports TDI keyword only. SMASK, TDO, and MASK are not supported. |

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Running Download Debugger

Download Debugger runs separately from the Radiant software environment.

**To start the Download Debugger:**

1. Issue the start command.
   - The Radiant software Programmer main window: choose **Tools > Download Debugger**.
   - Windows: go to the Windows Start menu and choose **All Programs > Lattice Radiant Software > Accessories > Radiant Programmer > Download Debugger.**
   - In Linux, from the <*Programmer install_path*>*/bin/lin64* directory, enter the following on a command line:

     ```
     ./debugger
     ```

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Opening an Existing SVF File

The Download Debugger Edit Window allows you to perform various editing functions on your SVF file. You can check the SVF file for errors line by line when you program your device.

**To open an existing SVF file:**

1. In Download Debugger, choose **File > Open** to open the dialog box.
2. Locate and select the desired SVF file, and then click **Open**.

   The selected SVF file opens in the Download Debugger Edit Window.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Creating a New SVF File

**To create a new empty SVF file:**

- In Download Debugger, choose **File > New** to open the SVF Debugger Edit Window and create a new empty SVF file.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Setting Device Programming Options in Download Debugger

**To set SVF options:**

1. In Download Debugger, choose **Configuration > Options**.

   The "Options Dialog Box" on page 319 opens.

2. Select the options that you want and click **OK**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Setting Port Assignments and Options in Download Debugger

Use the "Cable and I/O Port Settings Dialog Box" on page 317 to specify the preferred location for download, select the port address or cable, and choose the download cable type.

**To set port assignments and options:**

1. In Download Debugger, choose **Configuration > Cable and I/O Port Setup**.

   The "Cable and I/O Port Settings Dialog Box" on page 317 opens.

2. Select the options that you want and click **OK**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Setting and Removing Breakpoints in an SVF File

By using the Download Debugger, you can set a breakpoint in your SVF file before processing it. The process stops at the line that you specify. After clicking OK in the dialog box, you can resume the process.

**To set a breakpoint in an SVF File:**

1. Select a line in your SVF file.
2. Choose **Command > Breakpoint > Toggle Breakpoint**.

**To select the next breakpoint:**

1. Select a line in your SVF file.
2. Choose **Command > Breakpoint > Next Breakpoint**.

**To select the previous breakpoint:**

1. Select a line in your SVF file.
2. Choose **Command > Breakpoint > Previous Breakpoint**.

**To clear all breakpoints:**

1. In your SVF file, select the breakpoint you want to remove.
2. Choose **Command > Breakpoint > Clear All Breakpoints**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Processing an SVF File

After setting device programming options and port assignments, you can process your SVF file with the Download Debugger. You can program a device with a download cable, step through each line of the process, reset it and begin again.

**To process an SVF file:**

- Choose **Command > Go**.

**To step through a process, line by line:**

- Choose **Command > Step**.

  The process will step through the SVF file line by line.

**To reset a process and start a process from the beginning:**

- Choose **Command > Reset**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Viewing the Download Debugger Log File

The Download Debugger generates a log file of the process and reports any errors encountered. When you select **Continue on Error** in the "Options Dialog Box" on page 319, the log file also reports state machine transitions and delay time for debug.

**To view a log file:**

- In Download Debugger, choose **View > View Log File**.

  The log file is displayed in a text editor.

**See Also** "Clearing the Contents of a Log File in Download Debugger" on page 313

- "Debugging SVF, STAPL, and VME Files" on page 308

# Saving a Log File in Download Debugger

**To save a log file:**

1. In Download Debugger, choose **Configuration > Options**.
2. The "Options Dialog Box" on page 319 opens.
3. Use the Browse button to specify the path of the directory you wish to save your log file to.
4. Click **OK**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Clearing the Contents of a Log File in Download Debugger

**To clear the contents of the log file:**

- In Download Debugger, choose **Edit > Clear Log File**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Editing an SVF File

Use the Edit menu commands in the Download Debugger to edit an SVF file.

**To edit an SVF file:**

1. In Download Debugger, select a line in your SVF file.
2. Make the edits using commands in the Edit menu, and click **File > Save**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Opening an Existing STAPL File

The Download Debugger allows you to perform various editing functions on your STAPL file. You can check the STAPL file for errors line by line when you program your device.

**To open an existing STAPL file:**

1. In Download Debugger, choose **File > Open** to open the dialog box.
2. Locate and select the desired STAPL (.stp) file, and then click **Open**.

   The selected STAPL file opens in the Download Debugger.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Creating a New STAPL File

**To create a new empty STAPL file:**

- In Download Debugger, choose **File > New**.

  A blank edit window opens. You can start entering contents for your new STAPL file, and save the file when you finish.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Viewing STAPL Processing

To view the output information during STAPL processing, you should open the Output Information dialog box before processing the STAPL file.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Setting and Removing Breakpoints in a STAPL File

By using the Download Debugger, you can set a breakpoint in your STAPL file before processing it. The process stops at the line that you specify. After clicking OK in the dialog box, you can resume the process.

**To set a breakpoint in a STAPL file:**

1. Select a line in your STAPL file.
2. Choose **Command > Breakpoint > Toggle Breakpoint**.

**To select the next breakpoint:**

1. Select a line in your STAPL file.
2. Choose **Command > Breakpoint > Next Breakpoint**.

**To select the previous breakpoint:**

1. Select a line in your STAPL file.
2. Choose **Command > Breakpoint > Previous Breakpoint**.

**To clear all breakpoints:**

1. In your STAPL file, select the breakpoint you want to remove.
2. Choose **Command > Breakpoint > Clear All Breakpoints**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Processing a STAPL File in Download Debugger

After setting device programming options and port assignments, you can process your STAPL file with the Download Debugger and examine the progress in the Output Information dialog box.You can program a device with a download cable, step through each line of the process, reset it and begin again.

**To process a STAPL file:**

- Choose **Command > Go**.

**To step through a process, line by line:**

- Choose **Command > Step**.

  The process will step through the STAPL file line by line.

**To reset a process and start a process from the beginning:**

- Choose **Command > Reset**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Viewing the Download Debugger Log File

The Download Debugger generates a log file that tracks the process and reports any errors encountered. The log file contains all the information displayed in the Status window and also the status of the cable. When you select **Continue on Error** in the "Options Dialog Box" on page 319, the log file also reports state machine transitions and delay time for debug.

**To view a log file:**

- In Download Debugger, choose **View > View Log File**.

  The log file is displayed in a text editor.

**See Also** "Saving a Log File in Download Debugger" on page 313

- "Clearing the Contents of a Log File in Download Debugger" on page 316
- "Debugging SVF, STAPL, and VME Files" on page 308

# Saving a Log File in Download Debugger

**To save a log file:**

1. In Download Debugger, choose **Configuration > Options**.

   The "Options Dialog Box" on page 319 opens.
2. Use the Browse button to specify the path of the directory you want to save your log file to.
3. Click **OK**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Clearing the Contents of a Log File in Download Debugger

**To clear the contents of the log file:**

- In Download Debugger, choose **Edit > Clear Log File**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

## Editing a STAPL File

Use the Edit menu commands in the Download Debugger to edit a STAPL file.

**To edit a STAPL file:**

1. In Download Debugger, select a line in your STAPL file.
2. Make the edits using commands in the Edit menu, and click **File > Save**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

## Setting Windows Options in Download Debugger

Use the Windows options to change the way how multiple windows are displayed in Download Debugger.

**To cascade open windows:**

- Choose **Window > Cascade**.

**To tile open windows:**

- Choose **Window > Tile**.

**See Also** "Debugging SVF, STAPL, and VME Files" on page 308

# Download Debugger Options

This section lists the options available in Download Debugger.

Topics include:



## Cable and I/O Port Settings Dialog Box

The following options are available in the Cable and I/O Port Settings dialog box.

**Cable Description** This option is displayed when multiple cables are detected. When enabled, this option allows you to select the desired cable.

**Cable** Specifies the download cable type: Lattice (parallel port), USB (LSC USB cable), or USB2 (FTDI USB2 cable)

**Port** Specifies the port to which the download cable is connected.

**Custom Port** Specifies a custom parallel port to which the download cable is connected. Use hexadecimal format to type the port name.

**Using Slave SPI Interface Connection** This option allows you to target the slave SPI port of the selected device. This allows you to debug slave SPI device programming

**Using JTAGI2C Interface Connection (HW-USBN-2B Cable Only)** Only use an HW-USBM-2B cable for a JTAGI2C interface connection.

**Use Default Clock Divider** Uses the fastest TCK clock speed.

**Use Custom Clock Divider** Enables the Use Custom Clock Divider feature.

**TCK Divider Setting (0 - 30x)** Allows you to slow down the TCK clock. This is done by extending the low period of the clock. Refer the following tables for specific frequency settings for USB-2B (2232H FTDI USB host chip), USB-2B (2232D FTDI USB host chip), and USB-2A and Parallel port cables.

**Table 33: USB-2B (2232H FTDI USB host chip)**

| Divider | Clock Frequency[1] | Divider | Clock Frequency[1] |
|---|---|---|---|
| 0 | 30 MHz | 5 | 5 MHz |
| 1 | 15 MHz (Default) | 6 | 4.2 MHz |
| 2 | 10 MHz | 7 | 3.7 MHz |
| 3 | 7.5 MHz | 8 | 3.1 MHz |
| 4 | 6 MHz | 9 | 3 MHz |
| | | 10 | 2.7 MHz |

[1]Calculation formula for USB-2B (2232H FTDI USB host chip):

Frequency = 60 MHz / (1 + ClockDivider) *2

**Table 34: USB-2B (2232D FTDI USB host chip)**

| Divider | Clock Frequency[2] | Divider | Clock Frequency[2] |
|---|---|---|---|
| 0 | 6 MHz | 5 | 1 MHz |
| 1 | 3 MHz (Default) | 6 | 0.8 MHz |
| 2 | 2 MHz | 7 | 0.7 MHz |
| 3 | 1.5 MHz | 8 | 0.65 MHz |
| 4 | 1.2 MHz | 9 | 0.6 MHz |

**Table 34: USB-2B (2232D FTDI USB host chip) (Continued)**

| | | |
|---|---|---|
| | 10 | 0.5 MHz |

[2]Calculation formula for USB-2B (2232D FTDI USB host chip):

Frequency = 12 MHz / (1 + ClockDivider) *2

**Table 35: USB-2A and Parallel port**

| Divider | Low Pulse Width delay[3] | Divider | Low Pulse Width delay[3] |
|---|---|---|---|
| 0 | NA | 5 | 5x |
| 1 | 1x | 6 | 6x |
| 2 | 2x | 7 | 7x |
| 3 | 3x | 8 | 8x |
| 4 | 4x | 9 | 9x |
| | | 10 | 10x |

[3]The USB-2A frequency fixed at 1.5 MHz and Parallel Port frequency fixed at 500 KHz

**Use Default I/O Settings** Only use the four JTAG signals.

**Use Custom I/O Settings** .Specify additional non-JTAG signals connected to the board.

**INITN Pin Connected** Select this option if you connect the INIT pin to the download cable. This option is only available with the ispDOWNLOAD USB cable.

**DONE Pin Connected** Select this option if you connect the DONE pin to the download cable. This option is only available with the ispDOWNLOAD USB cable.

**TRST Pin Connected** Select this option if you connect the TRST pin to the download cable. Specify active high or active low.

**PROGRAM Pin Connected** Select this option if you connect the PROGRAM pin to the download cable.

**ispEN Pin Connected** Select this option if you connect the ispEN pin to the download cable. Specify active high or active low.

**See Also** "Download Debugger Options" on page 317

# Options Dialog Box

The following options are available in the within the three tabs (General, SVF, STAPL) found in the Options dialog box.

**Log File Name** Allows you to specify name and location of log file.

**Clear Log File Each Time Start Application** A Check to clear the log file each time Download Debugger is started.

**Source Editor** Allows you to specify font, size, tab size, of Source Editor.

**Show Line Number** Toggles display of line numbers in left margin of Source Editor.

**Show Indicator Margin** Toggles display of indicator margin in left side of Source Editor.

**Disable SVF Syntax Checker** Disables the tool that checks the syntax of the SVF file.

**Ignore IR and DR Header Trailer in SVF** Ignores the HIR, TIR, HDR, and TDR information in SVF file. If you set up HIR, TIR, HDR, or TDR in this dialog box, this option must be selected.

**Continue on Error** Continues processing the file even if there is an error. You can look at the log file for the errors encountered. When this option is selected, the log file also reports state machine transitions and delay time for debug.

**Mixed Chain** Tells the Download Debugger that the hardware configuration is a mixed chain, therefore disabling the ISP chain when processing the SVF file.

**TCK Frequency** Specifies the TCK clock frequency for the chosen SVF file. This value is used to determine the TCK clock period and the length of the delay times.

**SVF Vendor** Allows you select from the following SVF vendors for the selected device: JTAG STANDARD, LATTICE, ALTERA, and XILINX. This option is available for JTAG devices only.

**Starting TAP State** Select the starting JTAG state of the JTAG State Machine for download. Only two states are available: TLR (Test-Logic-Reset) and RTI (Run-Test/Idle).

**Instruction Register Header** Sets the IR header length in number of bits (0-100).

**Instruction Register Trailer** Sets the IR trailer length in number of bits (0-100).

**Data Register Header** Sets the HDR in number of bits (0-100).

**Data Register Trailer** Sets the TDR in number of bits (0-100).

**Ignore IR and DR Header Trailer in STAPL** Ignores the HIR, TIR, HDR and TDR information in STAPL file. If you set up HIR, TIR, HDR, or TDR in this dialog box, this option must be selected.

**See Also** “Download Debugger Options” on page 317

# Using Programming File Utility

The Radiant software Programming File Utility is a stand-alone tool that allows you to view, compare, and edit data files. When comparing two data files, the software generates an output (.out) file with the differences highlighted in red.

**See Also** ▶"Running Programming File Utility" on page 321

- "Viewing Data Files" on page 322
- "Comparing Two Data Files" on page 322
- "Editing Feature Row Values" on page 323
- "Editing Control Register0 Values" on page 323
- "Editing the USERCODE in the Data File" on page 324
- "Securing the Device and Setting the Persistent Bit" on page 325

## Running Programming File Utility

**To Run the Programming File Utility:**

1. Issue the start command.
   - Form Programmer main window: choose **Tools > Programmer File Utility.**
   - In Windows: go to the Windows Start menu and choose **All Programs Lattice Radiant Software > Accessories > Radiant Programmer.**
   - In Windows, go to the Windows Start menu and choose **Programs > Lattice Radiant Programmer > Programming File Utility.**
   - In Linux: from the <*Programmer install path*>*/bin/lin64* directory, enter the following on a command line:

     ```
     ./fileutility
     ```

   The Programming File Utility window opens.

**See Also** ▶"Using Programming File Utility" on page 321

# Viewing Data Files

**To view a data file:**

- In the Programming File Utility, choose **File > Open** and select the data file you want to view.

  **Note**

  The data file can be in any of the following formats: .jed, .isc, .rbt, .rbka, .mska, .bit, .rbk, .msk, .bsm, .bsd, .bin, .hex, .mcs, .exo, xtek, nvcm, .out

Use the **Edit > Find** and **Edit > Find Next** commands to search for text in the file. Choose **File > Save As** to save the file with another file name.

Change file appearance such as font and font size by choosing **Configuration > Options** and changing settings in the Options dialog box.

**See Also** ▶"Using Programming File Utility" on page 321

# Comparing Two Data Files

You can use the Programming File Utility application to compare two data files. After creating the output file, the software displays the output file and highlights the differences between the two files.

**To compare two data files:**

1. Choose **Command > Data Files Comparison** or click  on the toolbar.

   The Data Files Comparison Dialog Box opens.
2. Click the Browse buttons under First Data and Second Data to select the two data files you want to compare.
3. Optionally, if you are comparing two readback files (.rbk), browse for the mask file (.msc) to include masked bits in the comparison.
4. If desired, type a different name for the output file or browse to a file name that you want to overwrite. By default, the software uses the first data file name and appends the extension .out.
5. Click **OK**.

   The software compares the files and displays a message indicating whether the two files are identical or contain differences.
6. Click **OK** to close the message box and display the output file in the Programming File Utility window.
7. If the files contain differences, the Next  and Previous  arrows will become enabled on the toolbar. Use these buttons to examine each difference in the output file.

**Viewing the Output File**

The output file displays lines of data from the first file. Each of these lines is followed by a comparison line which shows dots where the files are the same. When different, the line is highlighted in red and the data from the second file is displayed.

An output file that uses a mask file to compare two readback files displays each line of data from the first readback file in black, followed by a line of data in green from the mask file, and then a comparison line. The comparison line shows dots where the two readback files are the same. When different, the line is highlighted in red and the data from the second readback file is displayed.

**See Also** ▶"Using Programming File Utility" on page 321

# Editing Feature Row Values

You can use the Feature Row Editor to enable or disable silicon features in Lattice FPGA devices by editing the Feature Row fuse settings in the data file.

**To edit control register values:**

1. Choose **Tools > Feature Row Editor**.

   The "Feature Row Editor Dialog Box" on page 326 opens.

2. Using the **(Browse)** button, select the data (.jed) file you want to edit.

3. Click **Read** to display current data file settings.

   The software displays the default values in the top row. The second row shows the values that can be modified in black, and those that cannot be modified in red.

4. Click the cell of a value displayed in black to change it.

   - To change the value in Feature Row, click the value cell and toggle the value from 1 to 0 or from 0 to 1.

5. To overwrite the existing data file, choose **Save**.

6. To create a new data file with a different name, choose **Save As** and save as a different file name.

**See Also** ▶"Using Programming File Utility" on page 321

# Editing Control Register0 Values

You can use Control Register0 Editor to read the control registers of a bitstream file, modify the settings, and save them.

**To edit control register values:**

1. Choose **Tools > Control Register0 Editor**.

   The Control Register 0 editor opens (see "Control Register0 Editor Dialog Box" on page 326 for more information).

2. Using the **(Browse)** button, browse to the .isc, .rbt, or .bit file that you want to edit, select the file, and then click **Open**.

   The editor opens with the selected file.

3. Click **Read** to display the values of the control register.

4. The software displays the default values in the top row.

   The software displays the default values in the top row. The second row shows the values that can be modified in black, and those that cannot be modified in red.

5. Click the cell of a value displayed in black to change it.

   - To change a value, click the value cell and toggle the value from 1 to 0 or from 0 to 1.

6. To overwrite the existing data file, choose **Save.**

7. To create a new data file with a different name, choose **Save As** and save as a different file name.

8. A message box pops up, displaying the new control register value. Click **OK**.

9. A confirmation message box informs you that Write to file was successful. Click **OK**.

10. Click **Close** to exit the Control Register.

**See Also** "Using Programming File Utility" on page 321

# Editing the USERCODE in the Data File

The USERCODE Editor allows you to add information, such as serial code number, design code, or lot number, to the .jed, .isc, .rbt, or .bit file. This information is written to the USERCODE after the fuse map data in a JEDEC file. The signature must be in either decimal (0-9), ASCII, or hexadecimal (0-F) characters. In an ISC data file, this information is written in the user code data record in hexadecimal.

**To use the USERCODE Editor:**

1. Choose **Tools > USERCODE Editor**.

   The USERCODE Editor opens (see "USERCODE Editor Dialog Box" on page 327 for more information).

2. Using the **(Browse)** button, browse to the .jed, .isc, .rbt, or .bit file that you want to edit, select the file, and then click **Open**.

   The editor opens with the selected file.

3. Click **Read**.

   If there is no data in the field, you will see a string of 8 Fs (Hexadecimal).

4. Edit the string as required. The USERCODE edit box displays the current bit position as you type. The signature must be in decimal (0-9), ASCII, or hexadecimal (0-F) characters.
5. Click **Save** or **Save As** to save the USERCODE data when finished.
6. Click **Yes** when prompted to overwrite the data file.
7. A confirmation message box informs you that the USERCODE write to file was successful. Click **OK**.

If you open the .jed file in a text editor, you will see the new USERCODE data added to the bottom of the file, as shown in the example below:

```
N User Electronic Signature
Data*U00000000000000000000000010011110011001100*
```

If you open the .isc data file in a text editor, you will see the new USERCODE data added to the end of the bottom of the file, as shown in the example below:

```
data usercode (initialize --LATTICE_NOTEDATA_WIDTH = 32--
LATTICE_NOTEADDRESS_WIDTH = 1(97D84A22) ),
```

**See Also** ▶"Using Programming File Utility" on page 321

▶ "Running Programming File Utility" on page 321

# Securing the Device and Setting the Persistent Bit

The Security and Persistent Fields Editor allows you to read both the security settings and the persistent bit of the bitstream file and to change the setting to ON or OFF.

**To read the settings of the security and persistent fields:**

1. Choose **Tools > Security and Persistent Fields Editor.**
2. Click the **Browse** button and select the desired bitstream data file.
3. Click **Read** to view the settings for Security and Persistent Bit and display them in the Status cells.
4. To change a setting from OFF to ON or vice versa, click the status cell.
5. Do either of the following to save the changes:
   - To save the changes to the current data file, click **Save**. In the pop-up dialog box, click **Yes** to confirm that you want to overwrite the data file.

- To save the changed data file to a different location or with a different file name, click **Save As**. In the Save As Data File dialog box, specify file name and location, and click **Save**.

6. A confirmation message box informs you that Write to file was successful. Click **OK**.
7. Click **Close** to exit the Security and Persistent Fields Editor.

**See Also** "Using Programming File Utility" on page 321

- "Running Programming File Utility" on page 321

# Programming File Utility Options

This section lists the options available in Programming File Utility. Topics include:



## Feature Row Editor Dialog Box

The following options are available in the Feature Row Editor dialog box:

**Device** Lists the project's device. If a device is not displayed, use the browse button to locate the correct data file. Once a device is displayed, click **Read** to get the correct Max Bits/Digits information.

**Read** Reads the data file values and displays them in the display field.

**Save** Saves the data file.

**Save As** saves the data file with another file name.

**Close** Closes this dialog box.

**See Also** "Programming File Utility Options" on page 326

## Control Register0 Editor Dialog Box

The following options are available in the Control Register0 Editor dialog box:

**Device** Lists the project's device. If a device is not displayed, use the browse button to locate the correct data file. Once a device is displayed, click **Read** to get the correct Max Bits/Digits information.

**Read** Reads the data file values and displays them in the display field.

**Save** Saves the data file.

**Save As** saves the data file with another file name.

**Close** Closes this dialog box.

**See Also** ▶"Programming File Utility Options" on page 326

# USERCODE Editor Dialog Box

The following options are available in the USERCODE Editor dialog box:

**Device** Lists the project's device. If a device is not displayed, use the browse button to locate the correct data file. Once a device is displayed, click **Read** to get the correct Max Bits/Digits information.

**USERCODE Format** Allows you to display the USERCODE field in hexadecimal, decimal, or ASCII format.

**Usercode** Allows you to specify a USERCODE value to override the current value in the data file.

**Read** Reads the USERCODE field in the data file and displays it in the display field.

**Save** Writes the value in the USERCODE display field and saves the data file.

**Save As** Writes the value in the USERCODE display field allows you to save the data file with another file name.

**Close** Closes this dialog box.

**See Also** ▶"Programming File Utility Options" on page 326

# Testing and Debugging On-Chip

The final stage of developing a design is testing it on the actual Field Programmable Gate Arrays (FPGA) on either a test board or in your system. Lattice Semiconductor offers the following tool for debugging both the hardware aspect of the design and—if you are using the LatticeMico32 microprocessor—the software aspect:

- Reveal Analyzer to check for and to analyze specific events on signals

Before you start debugging, Reveal Analyzer requires that a special interface module be added to the design. After it's been added, rerun the design implementation process (Synthesize Design, Map Design, Place & Route Design) and generate bitstream data  file (depending on the device family) to program the FPGA. The tools can share the same ispDOWNLOAD cable and JTAG port that Programmer uses to download the design.

## About Reveal Logic Analysis

One of the most common activities in debugging is logic analysis. To do this, use Reveal Inserter and Reveal Analyzer. You can use both with all supported FPGA devices.

Reveal continuously monitors signals within the FPGA for specific conditions, which can range from simple to quite complex. When the trigger condition occurs, Reveal can save signal values preceding, during, and following the event for analysis, including a waveform presentation. The data can be saved to a value change dump file (.vcd), which can be used with tools such as ModelSim, or to an ASCII tabular format that can be used with tools such as Excel.

Before running Reveal Analyzer, use Reveal Inserter to add Reveal modules to your design. In these modules, specify the signals to monitor, define the trigger conditions, and other options. Currently, you can have only one

module due to the soft JTAG core.When the modules are set up, regenerate the bitstream data  file to program the FPGA.

Before starting a test run, set up Reveal Analyzer. This includes setting a number of options, including modifying trigger conditions and customizing the waveform display. You can save these settings for later use. During and after a test run, view the incoming data in Reveal's LA Waveform view. You can also save the data to a .vcd or .txt file to analyze with other tools.

**NOTE**

The current version of the Radiant software support for UltraPlus devices doest not support multi-core debug for Reveal.

**See Also**

- Lattice Radiant Software User Guide

# Using the Reveal Example Project

For hands-on experience using the Reveal tools, try the following example project while studying the online help.

The example is a simple 3-bit counter coded in Verilog and already has a Reveal module inserted. It also already has trace data that can be viewed in Reveal Analyzer. A test board is not required to open the Reveal tools or to view the existing trace data. To actually run Reveal Analyzer to collect new data, however, you'll need a test board.

**To start the Reveal example project:**

1. Choose **File > Open > Design Example**.

   The Open Example dialog box opens.

2. Click **counter_reveal**.
3. Browse to select the location.
4. Click **OK**.

   The count design project opens. At this point, you can open Reveal Inserter.

   **Note**

   If you want to preserve the original example for future experiments, choose **File > Archive Project** now.

5. In the Process bar on top, double-click **Export Files** to implement the design.

If you want to use the example project with a test board, change the device type to match the board by double-clicking the device in the File List view and running the design implementation process.

# Creating Reveal Modules

Create the modules for Reveal logic analysis with Reveal Inserter. The process consists of identifying trace signals, which are the signals that you want to analyze, defining a trigger signal, which is the event you want to analyze, and setting a few options. Then you insert the modules into your design and implement it.

With Reveal Inserter, it is easy to set up simple triggering conditions, as well as extremely complex ones. Triggering in Reveal is based on the trigger unit and the trigger expression. A trigger unit is used to compare signals to a value, and a trigger expression is used to combine trigger units to form a trigger signal.

Currently, you can only have one module due to the soft JTAG core. Each module has its own settings, trace signals, and trigger signal. In many cases, a single module is all that is required to debug a design. However, in designs with multiple clock regions, it may be necessary to sample different clock regions at the same time. For those types of designs, it is recommended that you use multiple modules, one for each clock region.

Take some time to plan the logic analysis, as Reveal modules can take a considerable amount of FPGA resources and creating the modules can take a significant amount of time. Find out how many EBRs and slices are still available, and consider adding all the trace signals and trigger events that you might want to analyze. The trigger signals can be modified in Reveal Analyzer while you're running tests if the necessary signals and capacity have been specified in the modules.

## About Reveal Inserter

Reveal Inserter has several views to help you manage the Reveal modules, find signals, and set up trace and trigger signals.

**Dataset** Dataset provides a list of all the modules in the Reveal project. To work on the trace and trigger signals of a module, select the module in the Dataset view. You can also add, remove, or rename modules by right-clicking a module name and choosing from the pulldown menu. See "Managing the Modules in a Project" on page 331.

**Design Tree** Design Tree provides a list of all the buses and signals in the design. You can select signals to trace and to trigger on by dragging them

from the Design Tree view. To find signals, use the Signal Search function. See “Searching for Signals” on page 332.

Design Tree also keeps track of how signals are being used. See “Viewing Signals in Design Tree” on page 332.

**Trigger Output** Trigger Output provides a list of trigger signals available from other modules. These signals can also be traced and triggered on by the current logic analyzer module. Select these signals by dragging them from the Trigger Output view. To make a trigger available in the Trigger Output view, see “Trigger Out” on page 352.

**Trace Signal Setup** Trace Signal Setup is where you assemble and organize the list of signals to be traced by the current logic analyzer module. Select signals by dragging them from the Design Tree and Trigger Output views. You can rearrange the signals in the list and organize them into groups. This view also provides several options for the trace operation. See “Setting Up Trace Signals” on page 336.

**Trigger Signal Setup** Trigger Signal Setup is where you assemble the trigger for the current logic analyzer module. The trigger is the event that tells the module to save the data from the trace signals. Triggers are built up from “trigger units,” collections of signals compared to specific values, and “trigger expressions,” logical and sequential combinations of trigger units. See:

- “About Trigger Signals” on page 340
- “Setting Up Trigger Units” on page 341
- “Setting Up Trigger Expressions” on page 345
- “Setting Trigger Options” on page 351

## Managing the Modules in a Project

Each Reveal Inserter project can include up to 15 modules and are listed in the Dataset view. You can add, rename, and remove modules in the Dataset view.

**To add a module:**

1. Choose **Debug > Add New Core**.
2. From the submenu, choose a module type.

   Reveal Inserter adds a new module in the Dataset pane.

**To rename a module:**

1. Double-click the module name in the Dataset pane.
2. Type the new name of the module over the old name. The module name must:
   - Begin with a letter.
   - Consist of letters, numbers, and underscores (_).

- Be different from all other modules in the Reveal project.
- Be different from all other modules and instances in the design.

3. Press **Enter**.

**To remove a module:**

1. Select the module in the Dataset pane.
2. Press **Delete**.

## Viewing Signals in Design Tree

The Design Tree pane shows the hierarchy of the whole design with all of the signals and buses. From the Design Tree pane, you can select signals and drag them to the Trace Signal Setup and Trigger Signal Setup views. To help you find signals, Design Tree has a search function, explained in "Searching for Signals" on page 332.

The Design Tree pane also gives some information about the signals and how they are being used in the Reveal module. If you select a signal in the hierarchy, the Output tab displays information about it. If a signal is used in the Reveal module, it appears in the Design Tree pane in bold, followed by a symbol showing how the signal is being used:

- @Tc: trace signal
- @Tg: trigger unit signal
- @C: control signal (sample clock or sample enable)
- @Mx: mixed bus. Some of the signals are being used one way and some are not or are being used in another way.

**To view all signals in the design hierarchy:**

- Right-click on the design name in the Design Tree pane and choose **Expand All** from the pop-up menu.

**To view the buses, ports, top-level signals, and top level of the hierarchy:**

- Right-click on the design name in the Design Tree pane and choose **Collapse All** from the pop-up menu.

## Searching for Signals

You can find signals in the Design Tree pane with a text search that includes wild card characters.

**To search for signals:**

1. In the Signal Search box in the Design Tree pane, type the full name of the signal or part of the name with wild cards (see the following table). The

search is case-insensitive. The bit range of a bus, such as the “[7:0]” in cout[7:0] is not part of the name and should not be included in the search. Instead, just search for “cout”.

| Wildcard | Finds | Example |
|---|---|---|
| ? | Any single character | “?out” finds aout, bout, and cout. |
| * | A sequence of any number of characters | “c*” finds cin and cout. |
| [<*string*>] | Any character in the string | “[ab]out” finds aout and bout. |
| [^<*string*>] | Any character except those in the string | “[^ab]out” finds cout but not aout or bout. |
| [<*c1*>-<*c2*>] | Any character in the range from <*c1*> through <*c2*> | “[a-c]out” finds aout, bout, and cout.<br>“cout:[4-6]” finds cout:4, cout:5, and cout:6. |
| [^<*c1*>-<*c2*>] | Any character except those in the range from <*c1*> through <*c2*> | “cout:[^3-7]” finds cout:0, cout:1, and cout:2. |

2. Click **Search**.

   If only one signal is found, it is highlighted it in the Design Tree pane.

   If more than one signal is found, the Search Result dialog box opens with a list of the signals found.

3. Select the desired signals and click **OK**.

   Click on a signal to select it. To select more than one signal, control-click on each one. To select all signals and buses in a range, click on the first signal to select and Shift-click on the last one.

   The selected signals are highlighted in the Design Tree pane.

**See Also** ▶“Viewing Signals in Design Tree” on page 332

## Checking the Design Rules

Use the Design Rule Check tool to verify that your Reveal project is not violating any design rules, such as proper module names, number of signals used, and required options set. The design rule check also tells you the total number of EBRs and slices needed for the project.

**To check the Reveal module settings:**

- Choose **Debug > Design Rule Check**.

  The results of the rule check are displayed in the Output tab.

## Saving a Project

Reveal Inserter automatically performs a design rule check whenever a project is saved. The results are shown in the Output tab.

**Note**

Reveal Inserter generates a new “signature,” or tracking mechanism, whenever a Reveal project is saved. Reveal Analyzer reads this signature to ensure that the FPGA has been programmed with the latest Reveal project. If you save the project without re-programming the FPGA, Reveal Analyzer will issue an error message, even if the Reveal project was not changed.

**To save the project settings in the current directory:**

- Choose **File > Save** <*filename*>.

**To save the project settings in another directory:**

- Choose **File > Save** <*filename*> **As**. In the Save Reveal Project dialog box, browse to the desired directory, enter the name of the new .rvl file name in the File Name box, and click **Save**.

**See Also**

- “Inserting the Reveal Modules” on page 353

## Limitations

Reveal Inserter has the following limitations:

**Unsupported VHDL and Verilog Features in Reveal Inserter** The following features are valid in the VHDL and Verilog languages but not supported in Reveal Inserter:

- Array types of two dimensions or more are not shown in the port or node section.
- Undeclared wires attached to instantiated component instances are not shown in the hierarchical design tree. You must declare these wires explicitly if you want to trace or trigger with them.
- Variables used in conditional statements like if-then-else statements are not available for tracing and triggering.
- Variables used in selection statements like the case statement are not available for tracing and triggering.
- If function calls are used in the array declaration, the actual size of the array is unknown to Reveal Inserter.
- Entity and architecture of the same design cannot be in different files.
- In Verilog, you must declare variables at the beginning of a module body to avoid obtaining different results from various synthesis tools.

- In VHDL, you must declare synthesis attributes within an entity, and not within an architecture, to avoid obtaining different results from various synthesis tools.
- Signals used in VHDL “generate” statements are not available for tracing and triggering.
- Signals that are VHDL user-defined enumerated types, integer type, or Boolean type are not available for tracing and triggering.

**Syn_keep and Preserve_signal Attributes** In VHDL, always define the syn_keep and preserve_signal attributes as Boolean types when you declare them in your design. Synplify defines them as Boolean types, and Reveal Inserter will issue an error message if you define them as strings.

**Signals Implemented as Hard Routes** Signals that are implemented as hard routes in the FPGA instead of using the routing fabric are not available for tracing or triggering. Examples are connections to IB and OB components. Many common hard routes are automatically shown as unavailable in Reveal Inserter, but some are not. If you select a signal for tracing or triggering that is implemented as a hard route, an error will occur during the synthesis, mapping, placement, or routing steps.

**Dangling or Unconnected Nets** Dangling, or unconnected, nets in Verilog or VHDL code are available for use with Reveal Inserter.

# Creating Logic Analysis Modules

The major steps in creating a Logic Analysis module are shown below.

The design must be successfully synthesized before running Reveal Inserter.

**Note**

Reveal Inserter has limitations and requirements in how it works with the design. See “Limitations” on page 334 and “Creating Logic Analysis Modules” on page 335.

**To create a Logic Analyzer module:**

1. After opening the design project, choose **Tools > Reveal Inserter**.

   Reveal Inserter launches with the active Reveal project (.rvl) file open. If there are no existing projects, Reveal Inserter creates one.

   Reveal Inserter also parses and statically elaborates the design. In some cases, code that was successfully synthesized is flagged as having an error. In these cases, Reveal Inserter is interpreting the HDL code more strictly than the chosen synthesis tool.

   To correct this problem, see the reveal_error.log file in the implementation directory. With the help of this file, locate and correct the problem in the code. Then synthesize the design and open Reveal Inserter again.

2. Select a module in the Dataset view. If you want to add another module to the project, choose **Debug > Add New Core > Add Logic Analyzer**. See also "Managing the Modules in a Project" on page 331.
3. Set up the trace signals as desired. See "Setting Up Trace Signals" on page 336.
4. Create trigger units, which are collections of signals and the values that are part of causing a trigger signal. See "Setting Up Trigger Units" on page 341.
5. Create trigger expressions, which are logical combinations of trigger units. See "Setting Up Trigger Expressions" on page 345.
6. Select trigger options. See "Event Counter" on page 352 and "Trigger Out" on page 352.
7. Once you have set up of all your modules, add them to the design project by choosing **Debug > Insert Debug** and running the design implementation process. See "Inserting the Reveal Modules" on page 353.

# Setting Up Trace Signals

Setting up the trace signals for a module consists of selecting the signals and buses to watch and dragging them into the Trace Signal Setup tab. You can have up to 512 trace signals in a module.

There is no limit to the bus organization of the design, so you can organize the signals according to how you want to view the data. Order them from top to bottom and organize them into buses, or rearrange the trace signals into your own organization of trace buses. You can manage them using the Debug menu or by right-clicking selected signals.

**To set up trace signals:**

1. Click the **Trace Signal Setup** tab.
2. Select a module in the Dataset view.
3. Select signals in the Design Tree and Trigger Output views and drag the signals to the desired position in the Trace Signal Setup tab. For help finding signals, see "Searching for Signals" on page 332.

   Check the value to the right of the Implementation box to see how adding trace signals affects the consumption of FPGA resources.
4. You can further organize the signals in the Trace Signal Setup tab by ungrouping buses into individual signals and grouping signals into new buses.
   - To break a bus into individual signals, select the bus and choose **Debug > UnGroup Trace Bus**.
   - To create a new trace bus, select the desired signals and buses and choose **Debug > Group Trace Data**. Double-click the new bus and type in the desired name.

You can also drag signals and buses into and out of other buses.

5. To see what happens in the signals that define the trigger units, select **Include trigger signals in trace data** at the bottom of the Trace Signal Setup tab.

   A bus named “Trigger Signals” appears at the top of your list of trace signals. For details, see “Include Trigger Signals in Trace Data” on page 337.

6. Set the trace options. There are several options that enhance the trace signals or control how data is sampled:

   - “Sample Clock” on page 337
   - “Sample Enable” on page 338
   - “Buffer Depth” on page 338
   - “Timestamp” on page 339
   - “Implementation” on page 339
   - “Data Capture Mode” on page 340

**See Also** “Viewing Signals in Design Tree” on page 332

## Include Trigger Signals in Trace Data

In order to monitor what happens on the signals that make up a trigger, you can add all signals used in the trigger units to the trace signals.

This option creates a bus named “Trigger Signals” at the top of your list of trace signals. Trigger Signals contains buses named for each trigger unit and contains the buses and signals used in the trigger units. The Trigger Signals bus cannot be moved or modified in any way.

**To add the trigger signals to the trace signals:**

- Select **Include trigger signals in trace data**.

## Sample Clock

The sample clock determines when the trace signals are sampled. Reveal Analyzer samples the trace signals once every clock cycle on the clock’s rising edge.

**To set the sample clock signal:**

- Find the signal in the Design Tree view and drag it to the Sample Clock box of the Trace Signal Setup tab.

**Note**

On the board, make sure that the sample clock frequency is at least that of the JTAG clock. If the sample clock speed is too slow, you will be unable to complete capturing data with Reveal Analyzer.

The sample clock frequency should be no more than 200 MHz.

**See Also** "Searching for Signals" on page 332

## Sample Enable

The sample enable is a signal that can be used to turn data capture on and off. Normally, data is captured for every sample clock cycle during a specified number of cycles. With sample enable, data capture only happens when the sample enable signal is active. 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.

An example is a design that contains different sections, with some sections only working during certain clock phases. The design uses a master clock and generates different signals for the phases. You can use one of the phase signals as the sample enable.

If the trigger occurs while the sample enable is inactive, Reveal Analyzer cannot accurately calculate the trigger point. Instead of showing the precise trigger point, Reveal Analyzer shows a trigger region that spans five clock cycles. Reveal Analyzer can guarantee that the trigger occurs in this region, but it cannot determine during which clock cycle the trigger occurs.

**To set the sample enable:**

1. Select **Sample Enable**.
2. Find the signal in the Design Tree view and drag it to the box in the Sample Enable section of the Trace Signal Setup tab.
3. In the box to the right of the signal name box, choose whether the signal is:

- **Active High**. Data can be captured when the sample enable is high.
- **Active Low**. Data can be captured when the sample enable is low.

**See Also** "Searching for Signals" on page 332

## Buffer Depth

The buffer depth specifies the size of the trace memory buffer as the maximum number of samples that can be stored. The buffer should be deep enough to hold enough samples, before and after the trigger, for your analysis multiplied by the number of trigger events that you want to see in one test run. Available values are powers of two from 16 to 65,536.

For example, if you want 20 samples from each of five events, choose a buffer depth of at least 128.

**To set the buffer depth:**

1. Determine the maximum number of samples for your test run. Be generous to avoid re-implementing the design to get a larger buffer.
2. In the Buffer Depth box, choose a value that is at least as large as the desired number of samples.

Check the value to the right of the Implementation box to see how changing the buffer depth affects the consumption of FPGA resources.

## Timestamp

The timestamp is a count of sample clock cycles from the beginning of a test run. This is different from the sample index that Reveal Analyzer automatically supplies. The sample index only provides a count of samples within each trigger's data set. The timestamp continues counting between triggers and when the sample enable signal blocks data capture.

However, unlike the sample index, the timestamp contains extra data in each sample and requires a larger trace buffer.

Use the timestamp when you want to know how long the test ran before triggering or how long the sample enable signal blocked data capture. The timestamp can also help associate triggers with external events or with data from another Reveal module using the same sample clock.

**To add timestamps to the trace samples:**

1. Select **Timestamp**.
2. Determine the number of sample clock cycles in the longest test run you want to do.
3. In the pulldown menu next to "Timestamp," select the size of the timestamp in bits. Choose the smallest value that can hold the count for the desired number of sample clock cycles.

   For example, if you want to run a test for 50 thousand cycles, choose 16 bits.

Check the value to the right of the Implementation box to see how changing the timestamp size affects the consumption of FPGA resources.

## Implementation

The implementation specifies what kind of RAM to use for the Reveal module. Normally EBR would be selected, but distributed RAM can be used if you are short of EBR.

**To set the implementation:**

- In the Implementation box, choose a kind of RAM:
  - **EBR** (embedded block RAM)
  - **DistRAM** (distributed RAM)

The number of EBR or slices needed is shown on the screen. This value changes as you add or remove trace signals, or change the buffer depth or timestamp size.

## Data Capture Mode

The data capture mode specifies whether Reveal Analyzer can look for one trigger or multiple triggers in a test run. Multiple Trigger Capture mode provides the greatest flexibility during test runs. Single Trigger Capture mode slightly reduces the amount of FPGA resources needed.

**To set the data capture mode:**

1. In the Data Capture Mode section, select one of the following:
   - **Single Trigger Capture**. Reveal Analyzer captures the data for only one trigger.
   - **Multiple Trigger Capture**. Reveal Analyzer captures the data for multiple triggers.
2. If you select Multiple Trigger Capture, choose the “Minimum samples per trigger.” The number of samples collected for each trigger is set in Reveal Analyzer but cannot be smaller than this value..

# About Trigger Signals

Most of the trigger features need to be set up initially in Reveal Inserter but many of them can be modified in Reveal Analyzer while testing the design. See Table 36.

Changes in Reveal Inserter means the design has to be re-implemented (synthesis, map, place, and route) and reloaded into the FPGA. So it is worthwhile to be generous in defining your trigger units, trigger expressions, and other options. Think of setting up a trigger signal as creating capacity and

access to signals and memory. Try to give yourself capacity to define all the triggering events that you might want to analyze later on.

**Table 36: Where Trigger Features Can Be Changed**

| Feature | | Reveal Inserter | Reveal Analyzer |
|---|---|---|---|
| Trigger Units | Add | ✔ | |
| | Name | ✔ | ✔ |
| | Signals | ✔ | |
| | Operator | ✔ | ✔ |
| | Radix | ✔ | ✔ |
| | Value | ✔ | ✔ |
| Trigger Expressions | Add | ✔ | |
| | Remove | ✔ | ✔ |
| | Name | ✔ | ✔ |
| | Expression | ✔ | ✔ |
| | RAM type | ✔ | |
| | Maximum sequence depth | ✔ | |
| | Maximum event counter | ✔ | |
| Multiple Trigger Capture | Make available | ✔ | |
| | Number of samples per trigger | ✔ | ✔ |
| | Number of triggers | | ✔ |
| Other Features | AND All versus OR All | | ✔ |
| | Final event counter size | ✔ | ✔ |
| | Trace buffer depth | ✔ | |
| | Timestamp | ✔ | |
| | Trigger position | | ✔ |

### See Also

- “Setting Up Trigger Units” on page 341
- “Setting Up Trigger Expressions” on page 345
- “Setting Trigger Options” on page 351
- “Setting Up the Trigger Signals” on page 361

# Setting Up Trigger Units

The trigger unit is used to compare a number of signals to a value. A number of different operators are available for comparison and can be dynamically changed during analysis, along with the comparison value and the trigger unit name.

Each trigger unit can have up to 256 signals. Since there are 16 allowable trigger units, each module can have a maximum of 4096 trigger signals.

Try to minimize the number of trigger units in a module. The more trigger units there are, the longer it takes for Reveal Analyzer to configure the module at the beginning of a test run. More than eight trigger units may cause a delay of 30 seconds or more; 16 trigger units many cause a delay of 5 minutes.

Before you start setting up the trigger units, set the **Default Trigger Radix**. This option is in the Trigger Unit section of the Trigger Signal Setup tab. The default trigger radix is applied automatically to any new trigger units that you create but does not affect the radix of any existing trigger units. You can always change the radix of any trigger unit at any time.

**To set up a trigger unit:**

1. To add a new trigger unit, click **Add** in the Trigger Unit section of the Trigger Signal Setup tab.
2. Specify the signals in the trigger unit by using one of following methods:
   - Drag and drop signals from the Design Tree and Trigger Output views to the Signals column. For more information on using the Design Tree view, see “Viewing Signals in Design Tree” on page 332 and “Searching for Signals” on page 332.
   - Double-click in the Signals column to open the TU Signals dialog box. See “Selecting and Ordering Trigger Signals” on page 342.
3. To change the order of the signals or to remove them, double-click in the **Signals** column.

   The TU Signals dialog box opens. See “Selecting and Ordering Trigger Signals” on page 342.
4. Click in the **Operator** column and choose an operator from the pulldown menu. See “About Trigger Unit Operators” on page 343.

   Both the operator type and the trigger unit value can be changed in Reveal Analyzer during hardware debugging.
5. If you want to change the radix of the Value column, click in the **Radix** column and choose a radix from the pulldown menu. The menu includes token sets whose bit width matches the trigger unit’s signals.

   Token sets are text labels for values that might appear on trace buses. You can create and apply token sets in Reveal Analyzer. See “Creating Token Sets” on page 369.
6. Enter the comparison value in the **Value** column. The form of the value must match the specified radix. If you’ve selected a token set in the Radix

column, a pulldown menu opens in the Value column listing the tokens in the token set.

You can use “x” for a don’t-care value if you selected binary, octal, or hexadecimal in the Radix column and if you selected ==, !=, or serial compare in the Operator column.

## Selecting and Ordering Trigger Signals

In the Trigger Unit section of the Trigger Signal Setup view, double-click in the Signals column. The TU Signals dialog box opens.

**To select and order trigger unit signals:**

1. In the left side of the TU Signals dialog box, select signals that you want to use in the trigger unit. Click on a signal to select it. To select more than one signal, control-click on each one. To select all signals and buses in a range, click on the first signal to select and Shift-click on the last one.
2. Click **>** to add them to the box on the right.
3. To remove signals from the trigger unit, select them in the right box and click **<**.
4. Organize the signals by selecting one and clicking the up or down arrow until the signal is in its desired position. Continue until all the signals are in the desired order from least significant bit (LSB) down to the most significant bit (MSB).
5. Click **OK**.

## About Trigger Unit Operators

Most of the trigger unit operators use standard logical comparisons between the current value of the combined signals of the trigger unit and a specified value.

Operators can be changed in Reveal Analyzer, with the exception of “serial compare”.

**Standard Logical Operators** Reveal includes the following operators:

- == equal to
- != not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to

**Rising-Edge and Falling-Edge Operators** The “rising edge” and “falling edge” operators check for change in the signal value, not the value itself. So

the trigger unit's specified value is a bit mask showing which signals should have a rising or falling edge.

- A **1** means "look for the edge;"
- A **0** means "ignore this bit." A multiple-bit value is true if any of the specified bits has the edge.

For example, consider a trigger unit defined as cout[3:0], rising edge, 1110. This trigger unit is true only when cout[3], cout[2], or cout[1] have a rising edge. What happens on cout[0] does not matter.

- 0000 > 1110

  True because cout[3], cout[2], and cout[1] rose.

- 0000 > 1111

  True for the same reason. It does not matter whether cout[0] rises or not.

- 0000 > 0100

  True because a rising edge on any of the specified bits is sufficient.

- 1000 > 1000

  False because cout[3] is high, but did not rise.

**Serial Compare** The "serial compare" operator checks for a series of values on a single signal. For example, if a trigger unit's specified value is 1011, the "serial compare" operator looks for a **1** on the first clock, a **0** on the second clock, a **1** on the third clock, and a **1** on the last clock. Only after those four conditions are met in the four clock cycles is the trigger unit true.

Serial compare is available only when a single signal is listed in the trigger unit's signal list. The radix is automatically binary.

You can only set the serial compare operator in Reveal Inserter. You cannot change or select it in Reveal Analyzer as you can the other operators.

## Managing Trigger Units

You can add and remove trigger units only in Reveal Inserter. You cannot add them in Reveal Analyzer.

**To add a trigger unit:**

- To add a new trigger unit, click **Add** in the Trigger Unit section of the Trigger Signal Setup tab.

**To rename a trigger unit:**

- Click in the Name column of the Trigger Unit section of the Trigger Signal Setup tab and type in the new name. The name can consist of letters, numbers, and underscores. The first character must be either an underscore or a letter.

**To remove a trigger unit:**

1. In the Trigger Unit section of the Trigger Signal Setup tab, click in any box in the line representing the trigger unit that you want to remove.
2. Click **Remove**.

**See Also** ▶"Setting Up Trigger Units" on page 341

# Setting Up Trigger Expressions

You can set up the initial trigger expressions in Reveal Inserter and change them and their names in Reveal Analyzer. You can also enable or disable trigger expressions in Reveal Analyzer. However, you cannot change the maximum sequence depth or the maximum event counter of the trigger expressions in Reveal Analyzer.

**To set up a trigger expression:**

1. Click **Add** in the Trigger Expression section of the Trigger Signal Setup tab to add a new trigger expression.
2. Enter the trigger expression in the **Expression** column. See "Trigger Expression Syntax" on page 346.

   Reveal Inserter checks the syntax and displays the syntax in red font if it is erroneous.

   Both the trigger units and operators associated with a trigger expression can be changed in Reveal Analyzer during hardware debugging.
3. Click in the **Ram Type** column and choose whether the trigger expression is to be implemented with EBR (embedded block RAM) or slices (distributed RAM). The menu also shows how many of each resource is needed. Choose EBR if available. If short of EBR, choose slices.
4. Click in the **Max Sequence Depth** column and choose the maximum number of sequences, or trigger units connected by THEN operators, that can be used in the trigger expression.

   The maximum sequence depth must be at least as large as the number in the Sequence Depth column, which shows the number of sequences currently used by the trigger expression. Increasing the maximum sequence depth allows you to use more sequences if you change the trigger expression in Reveal Analyzer, but it also uses more FPGA resources. Consider the largest chain of sequences you might want to use in your test and choose a value at least that large. Reveal supports up to 16 sequence levels.

   The Max Sequence Depth value is set statically in Reveal Inserter and cannot be changed in Reveal Analyzer.
5. Click in the **Max Event Counter** box and choose the maximum size of the count in the trigger expression (the count is how many times a sequence must occur before a THEN statement).

   The Max Event Counter value must be at least as large as the largest counter value used in the trigger expression. Increasing the size of the

event counter allows you to use larger counts if you change the trigger expression in Reveal Analyzer, but it also uses more FPGA resources. Consider the largest count you might want to use in your test and choose a value at least that large. The maximum is 65,536.

The Max Event Counter setting can only be changed it in Reveal Inserter.

**See Also** "Example Trigger Expressions" on page 349

## Trigger Expression Syntax

Trigger expressions are combinations of trigger units. Trigger units can be combined in combinatorial, sequential, and mixed combinatorial and sequential patterns. A trigger expression can be dynamically changed at any time. Each core supports up to 16 trigger expressions that can be dynamically enabled or disabled in Reveal Analyzer. Trigger expressions support AND, OR, XOR, NOT, parentheses (for grouping), THEN, NEXT, # (count), and ## (consecutive count) operators. Each part of a trigger expression, called a sequence, can also be required to be valid a number of times before continuing to the next sequence in the trigger expression.

**Detailed Trigger Expression Syntax** Trigger expressions in both Reveal Inserter and Reveal Analyzer use the same syntax.

**Operators** You can use the following operators to connect trigger units:

- & (AND) – Combines trigger units using an AND operator.
- | (OR) – Combines trigger units using an OR operator.
- ^ (XOR) – Combines trigger units using a XOR operator.
- ! (NOT) – Combines a trigger unit with a NOT operator.
- Parentheses – Groups and orders trigger units.
- THEN – Creates a sequence of wait conditions. For example, the following statement:

  ```
  TU1 THEN TU2
  ```

  means "wait for TU1 to be true, then wait for TU2 to be true."

  The following expression:

  ```
  (TU1 & TU2) THEN TU3
  ```

  means "wait for TU1 and TU2 to be true, then wait for TU3 to be true."

  Reveal supports up to 16 sequence levels.

  See "Sequences and Counters" on page 347 for more information on THEN statements.
- NEXT – Creates a sequence of wait conditions similar to THEN, except the second trigger unit must come immediately after the first. That is, the second trigger unit must occur in the next clock cycle after the first trigger unit. See "Sequences and Counters" on page 347 for more information on NEXT statements.

- # (count) – Inserts a counter into a sequence. See "Sequences and Counters" on page 347 for information on counters.
- ## (consecutive count) – Inserts a counter into a sequence similar to Like # (count) except that the trigger units must come in consecutive clock cycles. That is, one trigger unit immediately after another with no delay between them. See "Sequences and Counters" on page 347 for information on counters.

**Case Sensitivity** Trigger expressions are case-insensitive.

**Spaces** You can use spaces anywhere in a trigger expression.

**Sequences and Counters** Sequences are sequential states connected by THEN or NEXT operators. A counter counts how many times a state must occur before a THEN or NEXT statement or the end of the sequence. The maximum value of this count is determined by the Max Event Counter value. This value must be specified in Reveal Inserter and cannot be changed in Reveal Analyzer.

The following is an example of a trigger expression with a THEN operator:

```
TU1 THEN TU2
```

This trigger expression is interpreted as "wait for TU1 to be true, then wait for TU2 to be true."

Here's the same example written with a NEXT operator:

```
TU1 NEXT TU2
```

it is interpreted as "wait for TU1 to be true, then wait *one clock cycle* for TU2 to be true." If TU2 is not true in the next clock cycle, the sequence fails and starts over, waiting for TU1 again.

The next trigger expression:

```
TU1 THEN TU2 #2
```

is interpreted as "wait for TU1 to be true, then wait for TU2 to be true for two sample clocks." TU2 may be true on consecutive or non-consecutive sample clocks and still meet this condition.

The following statement:

```
TU1 ##5 THEN TU2
```

means that TU1 must occur for five consecutive sample clocks before TU2 is evaluated. If there are any extra delays between any of the five occurrences of TU1, the sequence fails and starts over.

The next expression:

```
(TU1 & TU2)#2 THEN TU3
```

means "wait for the second occurrence of TU1 and TU2 to be true, then wait for TU3."

The last expression:

```
TU1 THEN (1)#200
```

means “wait for TU1 to be true, then wait for 200 sample clocks.” This expression is useful if you know that an event occurs a certain time after a condition.

You can only use one count (# or ##) operator per sequence. For example, the following statement is not valid, because it uses two counts in a sequence:

```
TU1 #5 & TU2 #2
```

Multiple count values are allowed for a single trigger expression, but only one per sequence. For two count operators to be valid in a trigger expression, the expression must contain at least one THEN or NEXT operator, as in the following example:

```
(TU1 & TU2) #5 THEN TU2 #2
```

This expression means “wait for TU1 and TU2 to be true for five sample clocks, then wait for TU2 to be true for two sample clocks.”

Also, the count operator must be applied to the entire sequence expression, as indicated by parentheses in the expression just given. The following is not allowed:

```
TU1 #5 & TU2 THEN TU2 #2
```

The count (#) operator cannot be used as part of a sequence following a NEXT operator. A consecutive count (##) operator may be used after a NEXT operator. The following is not allowed:

```
TU1 NEXT TU2 #2
```

The count (# or ##) operators can only be used in one of two areas:

- Immediately after a trigger unit or parentheses(). However, if the trigger unit is combined with another trigger unit without parentheses, a # cannot be used.
- After a closing parenthesis.

**Precedence** The symbols used in trigger expression syntax take the following precedence:

- Because it inserts a sequence, the THEN and NEXT operators always take the highest precedence in trigger expressions.
- Between THEN or NEXT statements, the order is defined by parentheses that you insert. For example, the following trigger expression:

  ```
  TU1 & (TU2|TU3)
  ```

  means “wait for either TU1 and TU2 or TU1 and TU3 to be true.”

  If you do not place any parentheses in the trigger expression, precedence is left to right until a THEN or NEXT statement is reached.

  For example, the following trigger expression:

```
TU1 & TU2|TU3
```

is interpreted as “wait for TU1 & TU2 to be true or wait for TU3 to be true.”

- The precedence of the ^ operator is same as that of the & operator and the | operator.
- The logic negation operator (!) has a higher precedence than the ^ operator, & operator, or | operator, for example:

```
!TU1 & TU2
```

means “not TU1 and TU2.”

- The # and ## operators have the same precedence as the ^ operator, & operator, or | operator. However, they can only be used in one of two areas:
  - Immediately after a trigger unit or trigger units combined in parentheses. However, if the trigger unit is combined with another trigger unit without parentheses, a # or ## operator cannot be used.

    Here is an example of correct syntax using the count (#) operator:

    ```
    TU1 #2 THEN TU3
    ```

    This statement means “wait for TU1 to be true for two sample clocks, then wait for TU3.”

    However, the following syntax is incorrect, because the count operator is applied to multiple trigger units combined without parentheses:

    ```
    TU1 & TU2#2 THEN TU3
    ```

  - Use parentheses to combine multiple trigger units and then apply a count, as in the following example:

    ```
    (TU1 & TU2)#2 THEN TU3
    ```

    This statement means “wait for the combination of TU1 and TU2 to be true for two sample clocks, then wait for TU3.”

**See Also** “Example Trigger Expressions” on page 349

## Example Trigger Expressions

The following is a series of examples that demonstrate the flexibility of trigger expressions.

**Example 1: Simplest Trigger Expression-**

TU1

This trigger expression is true, causing a trigger to occur when the TU1 trigger unit is matched. The value and operator for the trigger unit is defined in the trigger unit, not in the trigger expression.

**Example 2: Combinatorial Trigger Expression-**

TU1 & TU2 | TU3

This trigger expression is true when (TU1 and TU2) or TU3 are matched. If no precedence ordering is specified, the order is left to right.

**Example 3: Combinatorial Trigger Expression with Precedence Ordering-**

TU1 & (TU2 | TU3)

This trigger expression gives different results than the previous one. In this case, the trigger expression is true if (TU1 and TU2) or (TU1 and TU3) are matched.

**Example 4: Simple Sequential Trigger Expression-**

TU1 THEN TU2

This trigger expression looks for a match of TU1, then waits for a match on TU2 a minimum of one sample clock later. Since this expression uses a THEN statement, it is considered to have multiple sequences. The first sequence is “TU1,” since it must be matched first. The second sequence is “TU2,” because it is only checked for a match after the first sequence has been found. The “sequence depth” is therefore 2.

The sequence depth is an important concept to understand for trigger expressions. Since the debug logic is inserted into the design, logic must be used to support the required sequence depth. Matching the depth to the entered expression can be used to minimize the logic. However, if you try to define a trigger expression that has a greater sequence depth than is available in the FPGA, an error will prevent the trigger expression from running. The dynamic capabilities of the trigger expression can therefore be limited. To allow more flexibility, you can specify the maximum sequence depth when you set up the debug logic in Reveal Inserter. You can reserve more room for the trigger expression than is required for the trigger expression currently entered. If you specify multiple trigger expressions, each trigger expression can have its own maximum sequence depth.

**Example 5: Mixed Combinatorial and Sequential Trigger Expression-**

TU1 & TU2 THEN TU3 THEN TU4 | TU5

This trigger expression only generates a trigger if (TU1 AND TU2) match, then TU3 matches, then (TU4 or TU5) match. You can set precedence for any sequence, but not across sequences. The expression (TU1 & TU2) | TU3 THEN TU4 is correct. The expression (TU1 & TU2 THEN TU3) | TU4 is invalid and is not allowed.

**Example 6: Sequential Trigger Expression with Sequence Counts-**

The next trigger expression shows two new features: the sequence count and a true operator to count sample clocks:

(TU1 & TU2)#2 THEN TU3 THEN TU4#5 THEN (1)#200

This trigger expression means wait for (TU1 and TU2) to be:

1. True twice
2. Wait for TU3 to be true
3. Wait for TU4 to be true five times
4. Wait 200 sample clocks

The count (# followed by number) operator can only be applied to a whole sequence, not part of a sequence. When the count operator is used in a sequence, the count may or may not be contiguous. The always true operator (1) can be used to wait or delay for a number of contiguous sample clocks. It is useful to know an event you want to capture occurs a certain time after a condition, but you did not know the state of the trigger signals at that time.

However, there is a limitation on the maximum size of the counter. This depends on how much hardware is reserved for the sequence counter. When you define a trigger expression, the Max Event Counter setting in the Trigger Expression section of Reveal Inserter and Reveal Analyzer specifies how large a count value is allowed in the trigger expression. Each trigger expression can have a unique Max Event Counter setting.

**See Also** ▶"Trigger Expression Syntax" on page 346

## Managing Trigger Expressions

Trigger expressions are combinatorial or sequential equations of trigger units or both. Trigger expressions can be defined during insertion and changed in Reveal Analyzer. You can add up to 16 trigger expressions.

You can add trigger expressions in Reveal Inserter but not in Reveal Analyzer. You can dynamically enable or disable individual trigger expressions before triggering is activated during hardware debugging.

**To add a trigger expression:**

- In the Trigger Expression section of the Trigger Signal Setup tab, click **Add**.

**To rename a trigger expression:**

- Click in the Name column of the Trigger Expression section of the Trigger Signal Setup tab, and type in the new name. The name can consist of letters, numbers, and underscores. The first character must be either an underscore or a letter.

**To remove a trigger expression:**

1. Click in any box in the line representing the expression that you want to remove.
2. Click **Remove**.

**See Also** ▶Setting Up Trigger Expressions

# Setting Trigger Options

In addition to the trigger units and trigger expressions, there are two other aspects of triggers: the final event counter and enabling the trigger signal for other Reveal modules.

## Event Counter

The final event counter allows a counter to be added to the final trigger of one or more trigger expressions. In order to use the final event counter during logic analysis, you must specify it during insertion, along with the maximum count allowed. The actual count used by the counter during triggering can be dynamically changed during logic analysis.

**To add a counter to the output of the final trigger:**

1. Select **Enable final trigger counter** in the lower left portion of the Trigger Signal Setup tab.
2. In the **Event Counter Value** pulldown menu, choose the maximum size of the count of all the trigger expression outputs combined. You can choose powers of two between 2 and 65536.

   Clearing the “Enable final trigger counter” option is equivalent to setting the counter to a value of 1.

   You can change the value of this parameter in Reveal Analyzer, but you cannot make the value bigger, so be sure to reserve enough space for the count that you think you will need.

## Trigger Out

To support triggers based on multiple sample clocks, cross-triggering is available between different debug modules. Reveal provides an optional trigger-out signal in the triggering section for every module.

If a design has multiple modules, trigger-out signals from other modules are listed as an available signal. To use a trigger-out signal as an input to another module, you must specify it as a NET or BOTH type. The IO type is only used for connecting the trigger-out to an external I/O. Trigger-out signals are listed in the Trigger Output view.

**To create a trigger output signal:**

1. Select **Enable Trigger Out** from the bottom of the Trigger Signal Setup view.
2. In the Net pulldown menu, choose one of the following:
   - **IO** – Creates an I/O signal that can connect to an I/O pin.
   - **NET** – Creates a net signal that can be used in another module.
   - **BOTH** – Creates a signal that can be used on another module and can connect to an I/O pin.

3. If you want to change the signal's name, double-click the desired name next to the Net menu and type in the new name.
4. In the Polarity menu, choose whether the signal is **Active High** or **Active Low**.
5. In the "Minimum pulse width" box, enter the minimum pulse width of the trigger output signal, measured in cycles of the sample clock. You can input any value of 0 or greater as the minimum pulse.

**Note**

For soft IP or single core devices, only **IO** is selectable in Net pulldown menu.

# Inserting the Reveal Modules

When you finish setting up the trace and trigger signals, you can insert the Reveal modules into the design.

**Note**

Interactive and stand-alone synthesis are not compatible with Reveal modules. Reveal Inserter automatically uses the integrated synthesis option. Make sure your design project is set up for integrated synthesis if not done already.

**To insert the debug logic modules into the design:**

1. Choose **Debug >** **Insert Debug**.
2. In the Insert Debug to Design dialog box, select the modules to insert.
3. Select **Activate Reveal file in design project**.

   If the .rvl file is not active in the design project, the Reveal modules will not be included during synthesis.
4. Click **OK**.

   Reveal Inserter performs a design rule check and saves the Reveal (.rvl) file. The Output view shows resource requirements and the DRC report for the modules. The .rvl file is listed in the File List pane under Debug Files.
5. Implement the design in the usual way. See "Implementing the Design" on page 161.

   Adding Reveal modules can sometimes cause design implementation to fail. If you have any problems, see "Troubleshooting Design Implementation Errors" on page 353 and "Limitations" on page 334.

## Troubleshooting Design Implementation Errors

If the design implementation process fails after inserting a Reveal module, check this section for solutions.

**Signals Implemented as Hard Routes** Some signals that are implemented as hard routes in the FPGA instead of using the routing fabric are not available for tracing or triggering. Examples are connections to IB and OB components. If you select a signal for tracing or triggering that is implemented as a hard route, an error will occur during the synthesis, mapping, placement, or routing steps. However, the error can take many forms. Following are some examples.

**Example 1:** An example error message from the synthesis log file, <*cktname*>.log:

*@E:"f:\cws\bugs\cr37986\reveal_workspace\tmprveal\rx_ddr_rvl.vhd":648:8:648:12|Port 'serin' on Chip 'RX_DDR' drives 1 PAD loads and 1 non PAD loads*

**Example 2:** In this example error message, the serin signal is a hard route, but serin is not the name of the original signal that was traced. The hierarchical path shown is for the Reveal module that was generated. It is not part of the original design and is not displayed during the debug module insertion. The error message does not specify which trace or trigger signal is causing the problem.

To manually determine which signal is causing this error, you can use two approaches:

1. Remove signals one by one in Reveal Inserter to see which caused the error. If you have only a few signals, this would be the best approach.
2. Manually look through the design to determine the problem. If you have many signals, this approach would be the best.

   The error message refers to the temporary HDL design that is generated during debug logic insertion. Normally this HDL source is deleted after the database is built. To save this temporary HDL source in the case of errors in mapping, you must set an environment variable by doing the following:

   a. In the System control panel, click on the **Advanced** tab, then click on **Environment Variable** at the bottom of the window.
   b. Create a new environment variable named KEEP_REVEAL_TEMP. The value can be anything, but it is normally set to TRUE.
   c. Once this variable is set, exit the Radiant software.
   d. Open the Radiant software and synthesize the design.

   You can now open the generated and explore the HDL with a text editor or HDL Explorer to determine which signal caused the error. The top-level generated file is located at <*project_directory*>/reveal_workspace/tmpreveal/<*project_name*>_rvl.<*v or vhd*>.

**Example 3:**

*ERROR - map: IO register/latch FF_inst cannot be implemented in PIC.*

In this case, the input of a register is being traced. But the register is being implemented as an input flip-flop because of a preference, USE DIN. Allowing

the register to be implemented as an internal flip-flop by removing the preference resolves the issue.

**See Also** ▶"Limitations" on page 334

## Removing Reveal Modules from the Design

When you want to remove the Reveal modules from your design, you can either remove the .rvl file from the design project or set it as inactive, leaving it easily available for future use. Otherwise, the modules will continue to be inserted.

**To remove the Reveal modules from the design:**

1. In the File List view, right-click the .rvl file.
2. Do one of the following:
   - To remove the Reveal modules but keep the project, choose **Set as Inactive**.
   - To delete the Reveal project, choose **Remove**.
3. Implement the design in the usual way. See "Implementing the Design" on page 161.

# Performing Logic Analysis

Once you have created one or more Reveal modules in Reveal Inserter, you can use Reveal Analyzer to capture the trace signal data from your evaluation board and view that data as waveforms. As you run your tests and start getting results, you can modify the trigger units and expressions and the use of the trace buffer to test other conditions.

**To perform logic analysis:**

1. If you are working in a secured lab (without a network connection to the design project files), make a copy of the required files and install them in the lab computer. See "Working in a Secured Lab" on page 356.
2. Connect your evaluation board and program the FPGA. See "Programming the FPGA" on page 357.
3. Start Reveal Analyzer by selecting **Create a new file** in the Reveal Analyzer Startup Wizard. See "Starting Reveal Analyzer" on page 357.
4. Set up the trigger signals for each module that you want to use. See "Setting Up the Trigger Signals" on page 361.
5. Capture data. See "Capturing Data" on page 370.
6. View the resulting waveforms for each module. See "Viewing Waveforms" on page 372.

7. Optionally, you can save the data in a Reveal Analyzer (.rva) file, a value change dump (.vcd) file for use in third-party tools, or in an ASCII text (.txt) file. See "Saving the Reveal Analyzer Settings and Data" on page 377.

**To just view waveforms captured earlier:**

1. Start Reveal Analyzer by selecting **Open an existing file** in the Reveal Analyzer Startup Wizard. See "Starting Reveal Analyzer" on page 357.
2. View the waveforms. See "Viewing Waveforms" on page 372.
3. Optionally, you can save the data in a Reveal Analyzer (.rva) file, a value change dump (.vcd) file for use in third-party tools, or in an ASCII text (.txt) file. See "Saving the Reveal Analyzer Settings and Data" on page 377.

# Working in a Secured Lab

The usual process of Reveal logic analysis assumes that you are performing logic analysis on the same computer on which you created the Reveal modules, or that you are using a computer on the same network with easy access to the files. However, if your evaluation system is in a secured lab that is off the network, use the stand-alone Reveal Analyzer and the stand-alone Programmer in the lab.

To perform logic analysis, copy the files needed to program the FPGA and to run Reveal Analyzer over to the lab computer. After you finish collecting trace data, you may want to copy settings or the trace data to bring them out of the lab.

The files you need to run Reveal Analyzer in the lab are:

- Bitstream (.bin or .hex) file produced by the Export Files stage of the design implementation process. This file is required to program the FPGA.
- Reveal project (.rvl) file produced by Reveal Inserter. This is the defining file for a Reveal project and its modules. The .rvl file identifies the trace and trigger signals, and stores the trace and trigger options.
- Reveal settings (.rvs) file produced by Reveal Inserter. This file contains settings that can be changed in Reveal Analyzer while running tests. These settings include important parts of trigger units and trigger expressions.

Files to Bring Back:

After collecting data with Reveal Analyzer, you may want to bring one or more files back from the lab either to update the Reveal project or to analyze the data somewhere else.

If you made changes to the settings for trigger units or trigger expressions in Reveal Analyzer and want to update the project in Reveal Inserter with them, copy the .rvs file.

To analyze the trace data outside the lab using Reveal Analyzer's LA Waveform view, choose **File >** **Save** <*file*> and copy the following two files:

- Reveal Analyzer (.rva) file defines the Reveal Analyzer project. This file also contains data about the display of signals in the LA Waveform view.
- Reveal trace (.trc) file contains data recorded from the last test run. This file loads Reveal Analyzer's LA Waveform view.

To analyze the trace data outside the lab using another tool, save the data as either a value change dump (.vcd) or text (.txt) file and copy that file. See "Saving to Other Formats" on page 378.

# Programming the FPGA

For an evaluation board using a single FPGA, program the FPGA in the usual way.

For an evaluation board using multiple FPGAs in a JTAG daisy chain, Reveal Analyzer has the following requirements:

- Only one device can be debugged at a time.
- The .xcf file must be in the design project directory.

**To program an FPGA with a Reveal module in a daisy chain:**

1. In Programmer, uncheck the Process column of the devices that do not get the Reveal module. This sets the operation of these devices to Bypass mode.
2. Ensure that the Process column of the device that does get the Reveal module is checked.
3. Double-click in the row of the device that does get the Reveal module.

   The Device Properties dialog box opens.
4. In the Device Properties dialog box, choose **Fast Program** from the pulldown menu.
5. Click **OK**.

For an FPGA that contains its own SPI flash, flash programming is also an option.

**See Also**

- "Programming the FPGA" on page 258

# Starting Reveal Analyzer

Before starting Reveal Analyzer you need to decide if you want to work with a new Reveal Analyzer (.rva) file or an existing one. The .rva file defines the Reveal Analyzer project and contains data about the display of signals in the

LA Waveform view. You may want to start Reveal Analyzer with a new file to set up a new test. Start with an existing file to rerun a test, to set up a new test based on existing settings, or to just view the waveforms from an earlier test.

How you start Reveal Analyzer also depends on whether you are using it integrated with the Radiant software or using the stand-alone version, and on your operating system.

## Starting with a New File

Before you can start Reveal Analyzer with a new .rva file, you need to be connected to your evaluation board with a download cable and have the board's power turned on.The following steps can also be seen in Figure 37

**To start Reveal Analyzer with a new file:**

1. Issue the start command. To start:
   - For integrated with the Radiant software, go to the Radiant software main window and choose **Tools > Reveal Analyzer**.
   - For stand-alone in Windows, go to the Windows Start menu and choose **Programs > Lattice Radiant Reveal > Reveal Logic Analyzer**.
   - For stand-alone in Linux, go to a command line and enter the following:

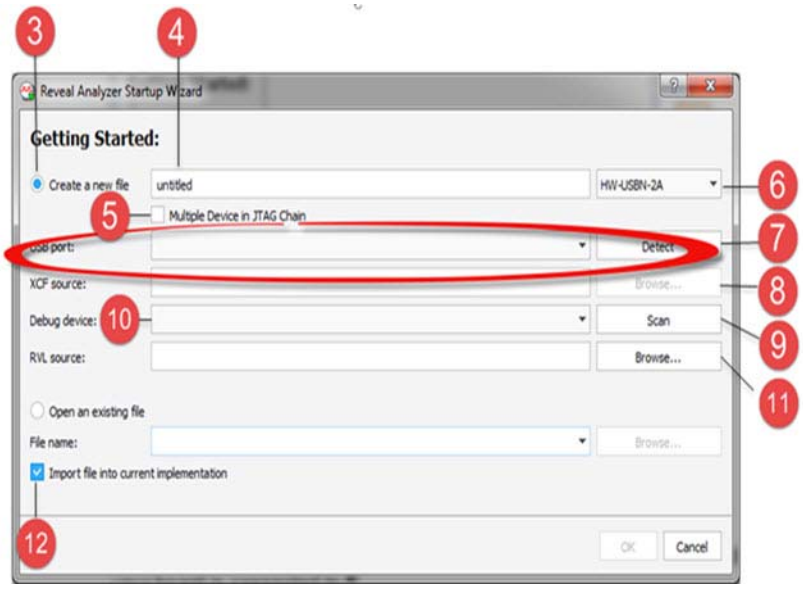
     *<Reveal install path>*/bin/lin64/**revealrva**

   The Reveal Analyzer Startup Wizard dialog box appears.
2. If Reveal Analyzer opens with an existing file, choose **File > Save As**.

   The Save Reveal Analyzer File dialog box opens. Change the filename and click **Save**. You now have a new .rva file ready to work with.
3. In the Reveal Analyzer Startup Wizard dialog box, Select **Create a new file** (at the upper-left of the dialog box).

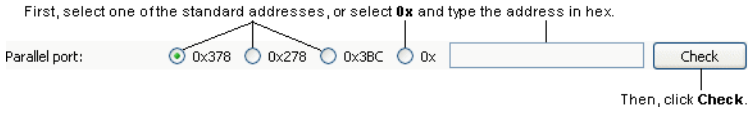
   The dialog box presents a few rows of boxes that need to be filled in. In the figure below, the numbers match up with the steps in this procedure and show where the boxes, buttons, and menus are for each step.
4. In the first second row, type in the base name of the file. The extension is added automatically.
5. If there are daisy-chained devices, select **Multiple Device in JTAG Chain**.
6. To the right of this row is a pulldown menu. Choose the type of cable that your board is connected to.
7. Select the port. The method depends on the cable type:
   - If USB, click **Detect**. Then choose from the active ports found. The following figure shows the row after choosing a USB type.
   - If parallel, select the port address. If it's not one of the standard addresses given, select **0x** and type in the hexadecimal address.

**Figure 37: Startup Wizard with Steps for a New File**





Then click **Check** to verify that the connection is working. The following figure shows the row after choosing a parallel type.




8. If there are daisy-chained devices, click **Browse** in the XCF source row to find the XCF source file.
9. Click **Scan** to find the FPGA.
10. If there is more than one FPGA on your board, go to the “Debug device” menu and choose one that has a Reveal icon. The icon indicates the presence of a Reveal module.
11. Click **Browse** in the RVL source row to find the Reveal Inserter project (.rvl) file.
12. To add the new .rva file to the File List view, select **Import file into current implementation**. The .rva file works the same either way.
13. Click **OK**.

### See Also

- “Creating a New Source File” on page 10

## Starting with an Existing File

If you want to start with an existing file, you just need to have that .rva file in the design project. You need to be connected to the evaluation board only if you want to run a test and capture data.

**To start Reveal Analyzer with an existing file:**

1. Issue the start command. To start:

   - In the Radiant software main window, choose **Tools > Reveal Analyzer**.
   - The stand-alone Reveal Analyzer in Windows, go to the Windows Start menu and choose **Programs > Lattice Radiant Reveal > Reveal Logic Analyzer**.
   - The stand-alone Reveal Analyzer in Linux, enter the following on a command line:

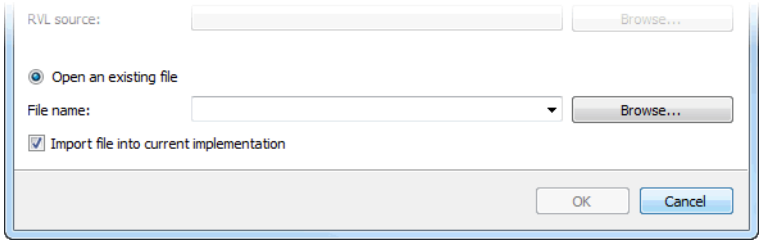
     <*Reveal install path*>/bin/lin64/**revealrva**

   The Reveal Analyzer Startup Wizard dialog box appears.

   If Reveal Analyzer opens with the .rva file you want to use, you're ready to go. Otherwise continue with the following steps.

2. In the Reveal Analyzer Startup Wizard dialog box, elect **Open an existing file** (in the lower part of the dialog box). See Figure 38.

**Figure 38: Existing File Part of Startup Wizard**




3. In the "File name" box, choose one of the available .rva files.
4. If the file you want is not in the menu, click **Browse** and browse to the desired .rva file.
5. To add the .rva file to the File List view, select **Import file into current implementation**. The .rva file works the same either way.
6. Click **OK**.

If the connection to your evaluation board has changed, either in the cable type or the computer port used, you need to tell Reveal Analyzer about the new connection. See "Changing the Cable Connection" on page 361.

## Changing the Cable Connection

If you need to change how your evaluation board is connected to your computer, go ahead and make the change. Then go through the following procedure to change the Reveal Analyzer project.

**To change the cable setting in a Reveal Analyzer project:**

1. Make sure your evaluation board is connected and that its power is on.
2. If Reveal Analyzer is not already open, start it as described in "Starting with an Existing File" on page 360.
3. Choose **Design > Cable Connection Manager**.

   The Cable Connection Manager dialog box opens.
4. In the dialog box, choose the cable type.

   The second row in the dialog box changes to select the specific port.
5. Select the specific port. The method depends on the port type:
   - If USB, click **Detect**. Then choose from the active ports found.
   - If parallel, select the port address. If it's not one of the standard addresses given, select **0x** and type in the hexadecimal address. Then click **Check** to verify that the connection is working.
6. To change the clock speed of the cable connection, adjust the value of TCK Low Pulse Width Delay.
7. Click **OK**.

# Setting Up the Trigger Signals

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 operators, radices, and values. 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.

**To modify the trigger of a module.**

1. Click on the **LA Trigger** tab.
2. Choose the module from the pulldown menu in the Reveal Analyzer toolbar.

   The definition of the module's trigger units and trigger expressions appear. Grayed out cells and options cannot be changed in Reveal Analyzer. To change them, you must go back to Reveal Inserter.
3. Modify the trigger units as desired. If you want to change:
   - Operator: Choose an operator from the pulldown menu. You cannot change or select serial compare in Reveal Analyzer. See "About Trigger Unit Operators" on page 363.

- Radix of the value: Choose a radix from the pulldown menu. The menu includes token sets whose bit width matches the trigger unit's signals.
- Value: Select it and type in a new comparison value. The form of the value must match the specified radix. If you selected a token set in the Radix column, a pulldown menu opens in the Value column listing the tokens in the token set.

  You can use "x" for a don't-care value if you selected binary, octal, or hexadecimal in the Radix column and if you selected ==, !=, or serial compare in the Operator column.

4. Modify the trigger expressions as desired. Trigger expressions can be completely rewritten provided that you do not exceed the maximum sequence depth or the maximum event counter. See "Trigger Expression Syntax" on page 364.
5. In the Trigger Expression Name column, select the trigger expressions that you want to use in the next test.
6. In the Trigger Options section, at Enable TE, choose how to combine the selected trigger expressions to form the trigger event:
   - **AND All** means that all of the trigger expressions must be satisfied to form the trigger event.
   - **OR All** means that only one of the trigger expressions must be satisfied to form the trigger event.
7. If you want the trigger event, as specified in the Enable TE option, to happen more than once before capturing trace data, select **Final Event Counter** in the Trigger Position section, and then specify the number of times the event is to happen to produce the actual trigger on the trigger signal. This option is available only if the event counter was enabled for the module in Reveal Inserter.
8. In the Trigger Options section, specify the number of samples per trigger and the number of triggers you want to record. The number of samples multiplied by the number of triggers cannot be greater than the trace buffer depth that was specified in Reveal Inserter. Reveal Analyzer adjusts the values when needed.
9. In the Trigger Position section, specify the trigger position relative to the trace data. The numbers in the section title show the current position. For example, "8/128" means the trigger happens on sample 8 out of the 128 samples per trigger. Select either:
   - **Pre-selected** and choose one of the standard positions:
     - Pre-Trigger: 1/16 of the way from the beginning of the samples.
     - Center-Trigger: 1/2 of the way from the beginning of the samples.
     - Post-Trigger: 15/16 of the way from the beginning of the samples.
   - **User-selected** and choose a position with the slider.

It should be noted that for each of the Pre-, Center- and Post-Trigger samples, a user is only guaranteed to see the waveforms from that point on.

For example if you choose Post-Trigger (15/16), then a user will only see the sample waveforms for the last 1/16 of 256 sample data.

## About Trigger Unit Operators

Most of the trigger unit operators use standard logical comparisons between the current value of the combined signals of the trigger unit and a specified value. But some of the operators are unusual and need some explanation.

With the exception of “serial compare,” the operators can be changed in Reveal Analyzer.

**Standard Logical Operators** Reveal includes the following operators:

- == equal to
- != not equal to
- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to

**Rising-Edge and Falling-Edge Operators** The “rising edge” and “falling edge” operators check for change in the signal value, not the value itself. So the trigger unit’s specified value is a bit mask showing which signals should have a rising or falling edge. A 1 means “look for the edge;” a 0 means “ignore this bit.” A multiple-bit value is true if any of the specified bits have the edge.

For example, consider a trigger unit defined as cout[3:0], rising edge, 1110. This trigger unit will be true only when cout[3], cout[2], or cout[1] have a rising edge. What happens on cout[0] does not matter.

- 0000 > 1110

  True because cout[3], cout[2], and cout[1] rose.

- 0000 > 1111

  True for the same reason. It does not matter whether cout[0] rises or not.

- 0000 > 0100

  True because a rising edge on any of the specified bits is sufficient.

- 1000 > 1000

  False because cout[3] did not rise. It just stayed high.

**Serial Compare** The “serial compare” operator checks for a series of values on a single signal. For example, if a trigger unit’s specified value is 1011, the “serial compare” operator looks for a 1 on the first clock, a 0 on the next clock, a 1 on the clock after that, and a 1 on the last clock. Only after those four conditions are met in those four clock cycles is the trigger unit true.

Serial compare is available only when a single signal is listed in the trigger unit’s signal list. The radix is automatically binary.

You can only set the serial compare operator in Reveal Inserter. You cannot change it or select it in Reveal Analyzer as you can the other operators.

## Trigger Expression Syntax

Trigger expressions are combinations of trigger units. Trigger units can be combined in combinatorial, sequential, and mixed combinatorial and sequential patterns. A trigger expression can be dynamically changed at any time. Each core supports up to 16 trigger expressions that can be dynamically enabled or disabled in Reveal Analyzer. Trigger expressions support AND, OR, XOR, NOT, parentheses (for grouping), THEN, NEXT, # (count), and ## (consecutive count) operators. Each part of a trigger expression, called a sequence, can also be required to be valid a number of times before continuing to the next sequence in the trigger expression.

**Detailed Trigger Expression Syntax** Trigger expressions in both Reveal Inserter and Reveal Analyzer use the same syntax.

**Operators** You can use the following operators to connect trigger units:

- & (AND) – Combines trigger units using an AND operator.
- | (OR) – Combines trigger units using an OR operator.
- ^ (XOR) – Combines trigger units using a XOR operator.
- ! (NOT) – Combines a trigger unit with a NOT operator.
- Parentheses – Groups and orders trigger units.
- THEN – Creates a sequence of wait conditions. For example, the following statement:

  ```
  TU1 THEN TU2
  ```

  means “wait for TU1 to be true, then wait for TU2 to be true.”

  The following expression:

  ```
  (TU1 & TU2) THEN TU3
  ```

  means “wait for TU1 and TU2 to be true, then wait for TU3 to be true.”

  Reveal supports up to 16 sequence levels.

  See “Sequences and Counters” on page 347 for more information on THEN statements.
- NEXT – Creates a sequence of wait conditions, like THEN, except the second trigger unit must come immediately after the first. That is, the second trigger unit must occur in the next clock cycle after the first trigger unit. See “Sequences and Counters” on page 347 for more information on NEXT statements.
- # (count) – Inserts a counter into a sequence. See “Sequences and Counters” on page 347 for information on counters.
- ## (consecutive count) – Inserts a counter into a sequence. Like # (count) except that the trigger units must come in consecutive clock cycles. That is, one trigger unit immediately after another with no delay between them. See “Sequences and Counters” on page 347 for information on counters.

**Case Sensitivity** Trigger expressions are case-insensitive.

**Spaces** You can use spaces anywhere in a trigger expression.

**Sequences and Counters** Sequences are sequential states connected by THEN or NEXT operators. A counter counts how many times a state must occur before a THEN or NEXT statement or at the end of the sequence. The maximum value of this count is determined by the Max Event Counter value. This value must be specified in Reveal Inserter and cannot be changed in Reveal Analyzer.

Here is an example of a trigger expression with a THEN operator:

```
TU1 THEN TU2
```

This trigger expression is interpreted as “wait for TU1 to be true, then wait for TU2 to be true.”

If the same example were written with a NEXT operator:

```
TU1 NEXT TU2
```

it is interpreted as “wait for TU1 to be true, then wait *one clock cycle* for TU2 to be true.” If TU2 is not true in the next clock cycle, the sequence fails and starts over, waiting for TU1 again.

The next trigger expression:

```
TU1 THEN TU2 #2
```

is interpreted as “wait for TU1 to be true, then wait for TU2 to be true for two sample clocks.” TU2 may be true on consecutive or non-consecutive sample clocks and still meet this condition.

The following statement:

```
TU1 ##5 THEN TU2
```

means that TU1 must occur for five consecutive sample clocks before TU2 is evaluated. If there are any extra delays between any of the five occurrences of TU1, the sequence fails and starts over.

The next expression:

```
(TU1 & TU2)#2 THEN TU3
```

means “wait for the second occurrence of TU1 and TU2 to be true, then wait for TU3.”

The last expression:

```
TU1 THEN (1)#200
```

means “wait for TU1 to be true, then wait for 200 sample clocks.” This expression is useful if you know that an event occurs a certain time after a condition.

You can only use one count (# or ##) operator per sequence. For example, the following statement is not valid, because it uses two counts in a sequence:

```
TU1 #5 & TU2 #2
```

Multiple count values are allowed for a single trigger expression, but only one per sequence. For two count operators to be valid in a trigger expression, the expression must contain at least one THEN or NEXT operator, as in the following example:

```
(TU1 & TU2) #5 THEN TU2 #2
```

This expression means “wait for TU1 and TU2 to be true for five sample clocks, then wait for TU2 to be true for two sample clocks.”

Also, the count operator must be applied to the entire sequence expression, as indicated by parentheses in the expression just given. The following is not allowed:

```
TU1 #5 & TU2 THEN TU2 #2
```

The count (#) operator cannot be used as part of a sequence following a NEXT operator. A consecutive count (##) operator may be used after a NEXT operator, however. The following is not allowed:

```
TU1 NEXT TU2 #2
```

The count (# or ##) operators can only be used in one of two areas:

- Immediately after a trigger unit or parentheses(). However, if the trigger unit is combined with another trigger unit without parentheses, a # cannot be used.
- After a closing parenthesis.

**Precedence** The symbols used in trigger expression syntax take the following precedence:

- Because it inserts a sequence, the THEN and NEXT operators always take the highest precedence in trigger expressions.
- Between THEN or NEXT statements, the order is defined by parentheses that you insert. For example, the following trigger expression:

  ```
  TU1 & (TU2|TU3)
  ```

  means “wait for either TU1 and TU2 or TU1 and TU3 to be true.”

  If you do not place any parentheses in the trigger expression, precedence is left to right until a THEN or NEXT statement is reached.

  For example, the following trigger expression:

  ```
  TU1 & TU2|TU3
  ```

  is interpreted as “wait for TU1 & TU2 to be true or wait for TU3 to be true.”
- The precedence of the ^ operator is same as that of the & operator and the | operator.
- The logic negation operator (!) has a higher precedence than the ^ operator, & operator, or | operator, for example:

```
!TU1 & TU2
```

means “not TU1 and TU2.”

- The # and ## operators have the same precedence as the ^ operator, & operator, or | operator. However, they can only be used in one of two areas:
  - Immediately after a trigger unit or trigger units combined in parentheses. However, if the trigger unit is combined with another trigger unit without parentheses, a # or ## operator cannot be used.

    Here is an example of correct syntax using the count (#) operator:

    ```
    TU1 #2 THEN TU3
    ```

    This statement means “wait for TU1 to be true for two sample clocks, then wait for TU3.”

    However, the following syntax is incorrect, because the count operator is applied to multiple trigger units combined without parentheses:

    ```
    TU1 & TU2#2 THEN TU3
    ```
  - After a closing parenthesis. Use parentheses to combine multiple trigger units and then apply a count, as in the following example:

    ```
    (TU1 & TU2)#2 THEN TU3
    ```

    This statement means “wait for the combination of TU1 and TU2 to be true for two sample clocks, then wait for TU3.”

**See Also** “Example Trigger Expressions” on page 367

## Example Trigger Expressions

The following is a series of examples that demonstrate the flexibility of trigger expressions.

**Example 1: Simplest Trigger Expression** Following is the simplest trigger expression:

TU1

This trigger expression is true, causing a trigger to occur when the TU1 trigger unit is matched. The value and operator for the trigger unit is defined in the trigger unit, not in the trigger expression.

**Example 2: Combinatorial Trigger Expression** An example of a combinatorial trigger expression is as follows:

TU1 & TU2 | TU3

This trigger expression is true when (TU1 and TU2) or TU3 are matched. If no precedence ordering is specified, the order is left to right.

**Example 3: Combinatorial Trigger Expression with Precedence Ordering** In the following example of a combinatorial trigger expression, precedence makes a difference:

TU1 & (TU2 | TU3)

This trigger expression gives different results than the previous one. In this case, the trigger expression is true if (TU1 and TU2) or (TU1 and TU3) are matched.

**Example 4: Simple Sequential Trigger Expression** The following is an example of a simple sequential trigger expression:

TU1 THEN TU2

This trigger expression looks for a match of TU1, then waits for a match on TU2 a minimum of one sample clock later. Since this expression uses a THEN statement, it is considered to have multiple sequences. The first sequence is "TU1," since it must be matched first. The second sequence is "TU2," because it is only checked for a match after the first sequence has been found. The "sequence depth" is therefore 2.

The sequence depth is an important concept to understand for trigger expressions. Since the debug logic is inserted into the design, logic must be used to support the required sequence depth. Matching the depth to the entered expression can be used to minimize the logic. However, if you try to define a trigger expression that has a greater sequence depth than is available in the FPGA, an error will prevent the trigger expression from running. The dynamic capabilities of the trigger expression can therefore be limited. To allow more flexibility, you can specify the maximum sequence depth when you set up the debug logic in Reveal Inserter. You can reserve more room for the trigger expression than is required for the trigger expression currently entered. If you specify multiple trigger expressions, each trigger expression can have its own maximum sequence depth.

**Example 5: Mixed Combinatorial and Sequential Trigger Expressions** Here is an example showing how you can mix combinatorial and sequential elements in a trigger expression:

TU1 & TU2 THEN TU3 THEN TU4 | TU5

This trigger expression only generates a trigger if (TU1 AND TU2) match, then TU3 matches, then (TU4 or TU5) match. You can set precedence for any sequence, but not across sequences. The expression (TU1 & TU2) | TU3 THEN TU4 is correct. The expression (TU1 & TU2 THEN TU3) | TU4 is invalid and is not allowed.

**Example 6: Sequential Trigger Expression with Sequence Counts** The next trigger expression shows two new features, the sequence count and a true operator to count sample clocks:

(TU1 & TU2)#2 THEN TU3 THEN TU4#5 THEN (1)#200

This trigger expression means wait for (TU1 and TU2) to be true two times, then wait for TU3 to be true, then wait for TU4 to be true five times, then wait 200 sample clocks. The count (# followed by number) operator can only be applied to a whole sequence, not part of a sequence. When the count operator is used in a sequence, the count may or may not be contiguous. The

always true operator (1) can be used to wait or delay for a number of contiguous sample clocks. It is useful if you know that an event that you want to capture occurs a certain time after a condition but you did not know the state of the trigger signals at that time.

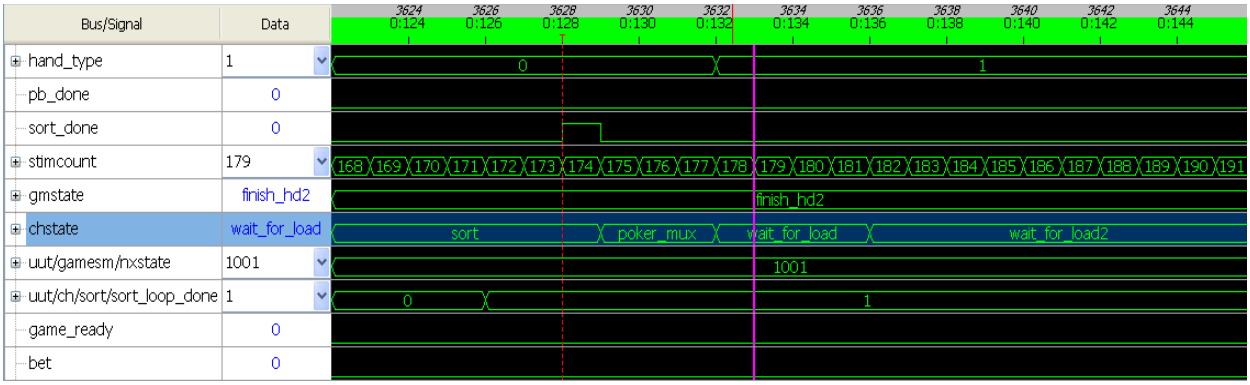
However, there is a limitation on the maximum size of the counter. This depends on how much hardware is reserved for the sequence counter. When you define a trigger expression, the Max Event Counter setting in the Trigger Expression section of Reveal Inserter and Reveal Analyzer specifies how large a count value is allowed in the trigger expression. Each trigger expression can have a unique Max Event Counter setting.

**See Also** ▶"Trigger Expression Syntax" on page 364

# Creating Token Sets

You can create sets of "tokens," or text labels, for values that might appear on trace buses. You can create tokens such as ONE, TWO, THREE, or Reset, Boot, and Load. Tokens can make reading the waveforms in Reveal Analyzer easier and can highlight the occurrence of key values. See Figure 39 for an example. The row for the chstate bus uses tokens.

**Figure 39: LA Waveform View Using Tokens**




**To create or modify a token set:**

1. Choose **Design > Token Set Manager**.

   The Token Manager dialog box opens. If the Reveal project already has token sets defined, they are listed in the dialog box.

2. If you want to use token sets that were previously saved to a separate file, right-click in the dialog box and choose **Import**. In the Import Tokens dialog box, browse to the token (.rvt) file and click **Open**.

   The token sets in the .rvt file are added to the list in Token Manager.

3. To create a new token set, click **Add Set**.

   A new token set is started with default values. But it has no tokens defined yet.

4. To change the size of the token values, double-click the value in the Num. of Bits column and type in the new width (up to 256) in bits. The width must be the same as the bus that the token set will be used with.

   The Num. of Bits value can only be changed when the token set is empty. If there are any tokens, you will get an error message.

5. To create a new token, select a token set. Then click **Add Token**.

   A new token is created with default values. Repeat for as many new tokens as needed.

6. You can modify token sets by doing any of the following:

   - To change the name of a token or token set, double-click the name and type a new name. The name can consist of letters, numbers, and underscores (_). It must start with a letter.
   - To change the value of a token, double-click the value and type in a new value. Token values must be prefixed by one of the radix indicators shown in the following table:

| Radix | Prefix | Example |
|---|---|---|
| Binary | b’ | b’110x0 |
| Octal | o’ | o’53 |
| Decimal | d’ | d’123 |
| Hexadecimal | h’ | x’0F2 |

     If a value does not have a prefix, its radix is assumed to be binary. You can use an “x” in binary numbers as a don’t-care value.

   - To remove a token or token set, select it and click **Remove**.

7. You can save the collection of token sets showing in the dialog box to a separate file for use in another project. To save the token sets, right-click and choose **Export**. In the Export Tokens dialog box, browse to the desired location and type in the name of the new token (.rvt) file. Click **Save**.

8. When you are done, click **Close** to close the dialog box. The token sets are automatically applied to the current Reveal project.

# Capturing Data

After you have configured trigger settings in the LA Trigger tab, you can capture data.

Before capturing data, however, your evaluation board must be connected and the design downloaded. See “Programming the FPGA” on page 357.

**To capture data:**

1. In the Reveal Analyzer toolbar, select the modules you want to use.

2. 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 correct trigger condition, then waits for the trigger conditions to occur. When a trigger occurs, the data is uploaded to your computer. The resulting waveforms appear in the LA Waveform tab.

If the trigger condition is not met, Reveal Analyzer continues running. In that case, you can use manual triggering, described in "Using Manual Triggering" on page 371.

**See Also** ▶"Stopping Data Capture" on page 371

## Stopping Data Capture

You can stop capturing data at any time.

**To stop data capture:**

1. Choose a module from the pulldown menu in the Reveal Analyzer tool bar.
2. Click the Stop button in the Reveal Analyzer toolbar.

This command only stops the data capture for the current module. You must stop each module separately.

## Using Manual Triggering

If triggering fails to occur or you want to trigger manually instead of triggering when a signal condition occurs, you can use manual triggering to collect data. The data may then help you find out why triggering did not occur as you originally intended.

When you select manual triggering, Reveal Analyzer fills the buffer with data captured from that moment. In single-trigger capture mode, it fills the buffer and stops. In multiple-trigger capture mode, it captures one trigger and data. You can then continue to manually trigger as many times as the original triggering setup specified. If you want to capture fewer triggers, you can manually trigger the desired number of times, then click the Stop button. Then the buffer starts uploading the data.

**To use manual triggering:**

1. After you start capturing data with the Run button, choose a module from the pulldown menu in the Reveal Analyzer tool bar.
2. Click the Manual Trigger button.

   This command only applies to the data capture on the current module. You must start each module separately.

3. When you have captured the desired number of triggers in multiple trigger capture mode, click the Stop button.

# Viewing Waveforms

After capturing data, you can view the trace data in waveform format in the LA Waveform tab. Whenever the trace stops, Reveal Analyzer reads the trace samples and automatically updates the signal waveforms.

**To view a waveform:**

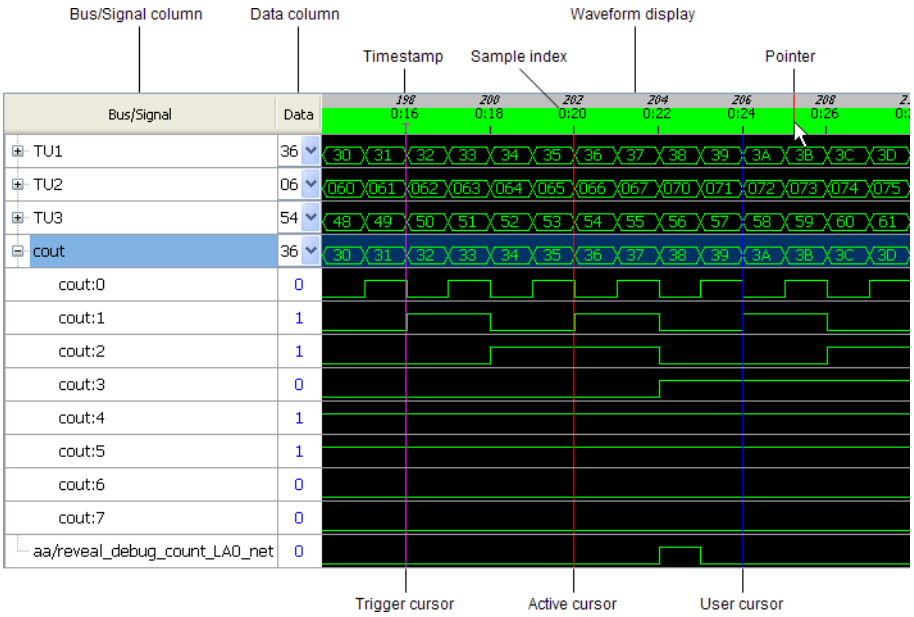
1. Click the **LA Waveform** tab.
2. Choose a module from the pulldown menu in the Reveal Analyzer tool bar.

Sometimes the waveform includes fewer clock cycles than you expect; in particular, fewer before the trigger. This happens if the trigger occurs so quickly after the test starts that there are not enough clock cycles before the trigger to fill that part of the trace buffer.

## About the LA Waveform View

Waveforms are presented in a grid layout as shown in Figure 40 along with several features to help you find and analyze the data.

**Figure 40: Elements of the LA Waveform View**




**Bus/Signal Column** Displays the names of the trace buses and signals in the selected module.

**Data Column** Displays the value of the bus or signal at the active cursor (a solid, red line that you can set in the waveform display). Buses also have a pulldown menu for setting the radix used in the Data column and in the waveform display. The menu includes token sets whose bit width matches the bus. See "Setting a Trace Bus Radix" on page 376.

**Waveform Display** Displays the trace data in waveform format. When there is room, bus values are included the display using the radix set in the Data column. You can zoom in and out, pan, and jump to various points.

The waveform display includes several other elements to help you read the display and analyze the data:

- Timestamp. The gray bar at the top of the display shows "timestamps" of the trace frames (actually, a simple count of the clock cycles). Timestamps are shown only if the Timestamp trace option was selected for the module in Reveal Inserter. See "Timestamp" on page 339.
- Sample Index. The green bar near the top of the display shows a count of triggers and trace samples within each trigger's data set. The sample indexes have the form <*trigger*>:<*sample*>. For example, 0:2 indicates the first trigger and the third trace sample for that trigger (the counts are zero-based). 2:10 indicates the third trigger and the eleventh trace sample for that trigger.
- Pointer. A red line that cuts across the timestamp and sample index bars, the pointer follows the horizontal movement of the mouse pointer across the waveform display. Use the pointer to see where you are in time as you examine the waveform.
- Cursors. Vertical lines cutting through all the signals, cursors mark moments in the waveform. See "About Cursors" on page 373.

**See Also**

- "Zooming In and Out" on page 375
- "Moving around the Waveform" on page 375

## About Cursors

The LA Waveform view comes with three types of "cursors" to highlight moments in the waveform. The cursors are vertical lines cutting through all the signals at the leading edge of a clock cycle. See Figure 40 on page 372. The three types are:

- Trigger. A purple line with a "T" at the top, trigger cursors are automatically placed at the moment of each final trigger event. If the module used a sample enable signal and the exact moment of the trigger is unknown, the waveform shows a trigger cursor five clock cycles before the sample enable signal turned inactive and sampling stopped.
- Active. A red line appears wherever you click in the waveform. The Data column shows the values of the signals and buses at the moment highlighted by the active cursor.

- User. A blue line can be placed anywhere you want. Use these cursors to mark moments of interest. You can also use these cursors to maneuver about a long waveform with the **Go to Cursor** command. See "Working with User Cursors" on page 374.

## Working with User Cursors

You can create any number of user cursors that can be moved or deleted. You can also jump the display to any one of them.

Most cursor functions require that the LA Waveform view be in Select mode. Do so by right-clicking in the LA Waveform view and choosing **Select Mode**.

**To create a user cursor:**

1. Click in the desired clock cycle.

   The active cursor appears. Make sure it is where you want the user cursor to be.

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

**To move a user cursor:**

1. Zoom in so you can easily see and click in individual samples.
2. Click in the desired location.

   The active cursor appears. Make sure it is where you want the user cursor to be.

3. Carefully click in the sample to the right of the user cursor.

   You must click on or to the right of the user cursor. Otherwise you are just moving the active cursor to a neighboring sample.

   The user cursor and the active cursor exchange locations.

**To jump to a user cursor:**

- Right-click in the waveform and choose **Go to Cursor >** <*cursor*>. Cursors are identified by the sample index as shown in the green bar at the top of the waveform display.

**To remove a user cursor:**

1. Click on or near the cursor.

   The active cursor appears. Make sure it is on or next to the user cursor you want to remove.

2. Right-click and choose **Remove Cursor**.

**To remove all user cursors:**

- Right-click in the waveform and choose **Clear All Cursors**.

**See Also** "Moving around the Waveform" on page 375

## Zooming In and Out

You can zoom in to expand the waveform and see more detail, or zoom out to see more of the waveform.

**To zoom in on a waveform:**

- Choose **View >** **Zoom In**.

**To zoom in on a specified area:**

1. Right-click the waveform and choose **Zoom Mode**.

   The pointer changes to a cross: ┼.

2. Hold down the left mouse button and drag the pointer across the area you want to zoom in on.

   A shaded area appears on the waveform display.

3. Release the mouse button.

   The shaded area expands to fill the display.

**To zoom out on a waveform:**

- Choose **View >** **Zoom Out**.

**To show the entire waveform in the window:**

- Choose **View >** **Zoom Fit**.

**See Also** "Moving around the Waveform" on page 375

## Moving around the Waveform

The waveform is usually much wider than the display, especially if you zoom in enough to see individual trace samples. To see nearby sections of the waveform, you can pan by sliding it left and right (described below) or by using the horizontal scroll bar at the bottom of the LA Waveform view. You can also jump to various points in the waveform including any cursors you've placed, the trigger point, the start of the display, and the end.

**To pan the waveform display:**

1. Right-click in the waveform and choose **Pan Mode**.
2. Press the left mouse button and drag to the left or the right.

**To jump to a location in the waveform:**

- Right-click in the waveform and choose one of the following from the menu. To jump to the:

- User cursor, choose **Go to Cursor >** <*cursor*>. Cursors are identified by the sample index as shown in the green bar at the top of the waveform display.
- Trigger point, choose **Zoom > Zoom Trigger**
- Start of the display, choose **Zoom > Zoom Start**
- End of the display, choose **Zoom > Zoom End**

The display changes to show the selection. The zoom level may change to keep the display filled.

**See Also**

- "About Cursors" on page 373
- "Working with User Cursors" on page 374
- "Zooming In and Out" on page 375

## Setting a Trace Bus Radix

You can set the radix of a trace bus displayed in the LA Waveform tab. You can choose a binary, octal, decimal, or hexadecimal radix. You can also use any token set whose bit width matches the bus.

**To set the bus radix of a signal or bus:**

1. In the LA Waveform tab, click in the Data cell of the signal or bus.

   A menu appears showing the different radixes and any token sets that fit.
2. Choose the desired radix or token set.

**To set the bus radix of multiple signals and buses:**

This method can set several signals to the same radix but cannot use tokens.

1. In the LA Waveform tab, select one or more buses.

   Click on a bus to select it. To select multiple buses, Control-click on each one. To select all buses in a range, click on one end of the range and Shift-click the other end. If you want to change all the signals in the waveform to the same radix, you do not need to select anything.
2. Right-click in one of the selected waveforms and choose **Set Bus Radix**. Be careful to click in the same row as one of your selections, or you will change the selection.

   The Set Bus Radix dialog box opens.
3. In the pulldown menu, choose the radix.
4. In the Range pulldown menu, choose **Selected signals** or, if you want to change all the signals in the waveform to the same radix, choose **All signals**.
5. Click **OK**.

## Changing LA Waveform Colors

You can change the colors used by the LA Waveform view.

**To change the colors:**

1. Choose **Tools > Options**.
2. In the Options dialog box, choose **Colors > Reveal Analyzer**.
3. Double-click on the color sample for the desired part of the LA Waveform view.

   The Select Color dialog box opens.
4. Select a color.
5. In the Select Color dialog box, click **OK**.
6. To see the effect of the change, click **Apply**.
7. Change other colors if desired.
8. Click **OK**.

## Counting Samples

You can easily count the number of samples in a range on the display.

**To count samples:**

- Click where you want to start counting and drag to the end of the range.

  While you're dragging, the LA Waveform view shows two red lines and the number of samples between the lines.

# Saving the Reveal Analyzer Settings and Data

You can save the waveform data in the following formats:

- Reveal Analyzer (.rva) file, which will also include the trigger settings and waveform setup for future use
- Value change dump (.vcd) file, which can be imported by third-party tools such as ModelSim or Active-HDL
- Text (.txt) file

Save to an .rva file if you want to see the data in the LA Waveform view again or if you want to use the trigger settings. See "Saving a Reveal Analyzer File" on page 378. To save to a different format, see "Saving to Other Formats" on page 378.

## Saving a Reveal Analyzer File

You can save the waveform along with the trigger settings and waveform setup in a Reveal Analyzer (.rva) file that you can use in the future. You can also save an existing .rva file in a file with a different name.

**To save changes in the current file:**

- Choose **File >** **Save** <*file*>.

**To save the file with a different name:**

1. Choose **File > Save** <*file*> **As**.

   The Save Reveal Analyzer File dialog box appears.
2. Browse to the directory in which you want to save the project.
3. In the File name box, type the file name.
4. Click **Save**.

## Saving to Other Formats

You can export the data captured for individual modules to a value change dump (.vcd) file, which can be imported by third-party tools such as ModelSim or Active-HDL, or to an ASCII text (.txt) file.

**To export data:**

1. Choose the module from the pulldown menu in the Reveal Analyzer tool bar.
2. If you want the data to include an approximate measure of time instead of a simple count of clock cycles, right-click the waveform and choose **Set Clock Period**. See "Specifying the Clock Period" on page 379.
3. If you want to export only some of the signals, select them in the waveform. You can only export whole buses. If you select only some of the signals in a bus, you get the whole bus.
4. Right-click in the waveform and choose **Export Waveform**.

   The Export Waveform dialog box opens.
5. Browse to the location where you want to export the file.
6. Type in a name in the **File name** box.
7. Choose a file type.
8. If you are exporting only some of the signals, choose **Selected signals** in the Range box.
9. If you are exporting to .vcd, type in a module name. This will form the title in the .vcd file. If you leave the field empty, the module name will be "<unknown>".
10. Click **Save**.

## Specifying the Clock Period

By default, Reveal Analyzer refers to time by a count of the sample clock cycles. You can convert this to an approximate measure of time, in nanoseconds or picoseconds, by specifying the length of the clock period. When the waveform is exported, this measure of time is shown instead of a simple count of cycles.

**To set the clock period:**

1. Right-click the waveform and choose **Set Clock Period**.
2. Specify the scale by choosing a unit for the period in the pulldown menu on the right side of the dialog box. If you are specifying a frequency in the megahertz range, choose **ns**. If you are specifying a frequency in the gigahertz range, choose **ps**.
3. Place the cursor in either the Period or Frequency text box and type in the desired value. The other text box is filled in automatically.

   Only integers are allowed. If you try to specify a frequency that requires a non-integer period, the period is truncated to an integer and the frequency is automatically adjusted. For example, typing 150 in the Frequency text box gives you a period of six and a frequency of 166.
4. Click **OK**.

# Strategy Reference Guide

A strategy provides a unified view of all the options related to implementation tools such as synthesis, map, and place and route. Strategy options are listed in the Strategy dialog box. Open the dialog box by double-clicking a strategy name in the File List view.

For information about an option, select it. A brief description appears at the bottom of the dialog box. Press **F1** to open this guide and see the full description.

The descriptions of the strategy options are grouped by their processes (such as Synplify Pro or Map Timing Analysis).

For detailed information on how to apply strategies to your project, see “Using Strategies” on page 21.

**Synplify Pro Options** Synplify Pro strategy options are listed below:

**Lattice Synthesis Engine (LSE) Options** LSE strategy options are listed below:

**Post Synthesis** Post-synthesis strategy options are listed below:



**MAP Design Options** MAP Design strategy options are listed below:



**MAP Timing Analysis Options** MAP Timing Analysis strategy options are listed below:



**Place & Route Design Options** Place & Route Design strategy options are listed below:

**Place & Route Timing Analysis Options** Place & Route Timing Analysis strategy options are listed below:



**Timing Simulation Options** Timing Simulation strategy options are listed below:



**Bitstream Options** Bitstream strategy options are listed below:

# Synplify Pro Options

This page lists all the strategy options associated with the Synplify Pro Synthesis process. For information on their use in Synplify Pro, see the Synopsis Synplify Pro for Lattice Reference Manual.

**Allow Duplicate Modules (for Synplify Pro)** Allows the use of duplicate modules in your design.

When it is set to True, the last definition of the module is used by the software and any previous definitions are ignored. The default is False.

**Area (for Synplify Pro)** Specifies optimization preference for area reduction over timing delay reduction.

The True option specifies the area reduction mode. When set to True, this setting overrides the setting in Frequency (for Synplify Pro).

The default is False.

This option is equivalent to the “set_option -frequency 1” command in Synplify Pro.

**Arrange VHDL Files** Allows Synplify Pro to reorder the VHDL source files for synthesis.

The default is True for VHDL or VHDL design entry type projects, and False for other projects. When this is set to False, Synplify Pro will use the file order in the Radiant software File List view.

**Clock Conversion** Controls gated and generated clock conversion.

Values are True and False.

**Command Line Options (for Synplify Pro)** Enables additional command line options for the Synplify Pro Synthesis process.

**To enter a command line option:**

1. In the Strategy dialog box, select **Synplify Pro** in the Process list.
2. Double-click the Value column for the Command line Options option.
3. Type in the option and its value (if any) in the text box.
4. Click **Apply**.

**Default Enum Encoding** (For VHDL designs) Defines how enumerated data types are implemented.

The type of implementation affects the performance and device utilization. Available options are:

- Default – Automatically assigns an encoding style based on the number of states:

  - Sequential: 0-4 enumerated types
  - Onehot: 5-40 enumerated types
  - Gray: more than 40 enumerated types
- Gray – Only one bit of the state register changes at a time, but because more than one bit can be hot, the value must be decoded to determine the state. For example: 000, 001, 011, 010, 110
- Onehot – Only two bits of the state register change (one goes to 0; one goes to 1) and only one of the state registers is hot (driven by a 1) at a time. For example: 0000, 0001, 0010, 0100, 1000
- Sequential – More than one bit of the state register can change at a time, but because more than one bit can be hot, the value must be decoded to determine the state. For example: 000, 001, 010, 011, 100

This option is equivalent to the “set_option -default_enum_encoding default | onehot | gray | sequential” command in Synplify Pro.

**Disable IO Insertion (for Synplify Pro)** Controls whether the synthesis tool will add I/O buffers into your design.

If this is set to True, Synplify Pro will not add I/O buffers into your design. If it is set to False (default), the synthesis tool will insert I/O buffers into your design.

This option is equivalent to the “set_option -disable_io_insertion 1 | 0” command in Synplify Pro.

**Export Radiant Software Settings to Synplify Pro GUI** Controls whether the strategy settings are exported to Synplify Pro during interactive synthesis (opening Synplify Pro through the Tools menu). After opening Synplify Pro, you can change settings in Synplify Pro's interface. This option has no effect with integrated or stand-alone synthesis.

Available options are:

- No – Synplify Pro opens with its own defaults, ignoring the strategy settings.
- Yes (default) – Synplify Pro opens with the strategy settings every time. Options set and saved in a previous Synplify Pro session are ignored.
- Only on First Launch – Synplify Pro opens with the strategy settings the first time only. After that, Synplify Pro opens with settings saved in a previous session or with its own defaults. After the first time, the strategy settings are ignored.

For more information, see “Interactive Synthesis” on page 569.

**FSM Compiler (for Synplify Pro)** Enables or disables the FSM Compiler and controls the use of FSM synthesis for state machines.

When Synplify Pro is selected as the synthesis tool, it enables or disables the FSM Compiler and controls the use of FSM synthesis for state machines. When this is set to True (default), the FSM Compiler automatically recognizes

and optimizes state machines in the design. The FSM Compiler extracts the state machines as symbolic graphs, and then optimizes them by re-encoding the state representations and generating a better logic optimization starting point for the state machines.

This option is equivalent to the “set_option -symbolic_fsm_compiler 1 | 0” command in Synplify Pro.

**Fanout Guide** Controls fanout during synthesis. When the specified fanout limit is achieved, logic will be duplicated.

The default is 1000.

This option is equivalent to the “set_option -maxfan <*number*>” command in Synplify Pro.

**Force GSR (for Synplify Pro)** Forces Global Set/Reset Pin usage.

Available options are:

- Auto – Allows the software to decide whether to infer Global Set/Reset in your design.
- False (default) – Does not infer Global Set/Reset in your design.
- True – Always infers Global Set/Reset in your design.

This option is equivalent to the “set_option -force_gsr auto | yes | no” command in Synplify Pro.

**Frequency (for Synplify Pro)** Specifies the global design frequency (in MHz). Nothing in the Value column means "auto" (the default) and Synplify Pro will try to maximize the frequency of the clocks.

The setting is ignored when Area (for Synplify Pro) is set to True.

This option is equivalent to the “set_option -frequency <*number*> | auto” command in Synplify Pro.

**Number of Critical Paths (for Synplify Pro)** Specifies the number of critical timing paths to be reported in the timing report.

This option is equivalent to the “set_option -num_critical_paths <*number*>” command in Synplify Pro.

**Number of Start/End Points** Specifies the number of start and end points you want the software to report in the critical path section of the timing report.

This option is equivalent to the “set_option -num_startend_points <*number*>” command in Synplify Pro.

**Pipelining and Retiming** Enables the pipelining and retiming features to improve design performance.

Values are:

- None – Disables the pipelining and retiming features.
- Pipelining Only (default) – Runs the design at a faster frequency by moving registers into the multiplier, creating pipeline stages.
- Pipelining and Retiming – When enabled, registers may be moved into combinational logic to improve performance.

This option is equivalent to the “setup_option -pipe 1 | 0 -retiming 1 | 0” command in Synplify Pro.

**Push Tristates** When this is set to True, the Synplify Pro compiler pushes tristates through objects such as muxes, registers, latches, buffers, nets, and tristate buffers, and propagates the high impedance state.

The high-impedance states are not pushed through combinational gates such as ANDs or ORs.

The default is False.

This option is equivalent to the “set_option -compiler_compatible 1 | 0” command in Synplify Pro.

**Resolve Mixed Drivers (for Synplify Pro)** If a net is driven by a VCC or GND and active drivers, setting this option to True will connect the net to the VCC or GND driver.

This option is equivalent to the “set_option -resolve_multiple_driver 1 | 0” command in Synplify Pro.

**Resource Sharing (for Synplify Pro)** When this is set to True (default), the synthesis tool uses resource sharing techniques to optimize area.

With resource sharing, synthesis uses the same arithmetic operators for mutually exclusive statements; for example, with the branches of a case statement. Conversely, you can improve timing by disabling resource sharing, but at the expense of increased area.

This option is equivalent to the “set_option -resource_sharing 1 | 0” command in Synplify Pro.

**Update Compile Point Timing Data** Determines whether (True) or not (False) changes inside a compile point can cause the compile point (or top-level) containing it to change accordingly.

When this is set to False (default), Synplify Pro keeps the top level module the same, which is desired by incremental flow.

When this is set to True, changes in low level partitions will be propagated to top partitions up to top module. Synplify Pro will possibly optimize timing data and certainly will write a new timestamp onto the partition for the top level module.

This option is equivalent to the “set_option -update_models_cp 1 | 0” command in Synplify Pro.

**Use Clock Period for Unconstrained I/O** Controls whether to forward annotate constraints for I/O ports without explicit user-defined constraints.

When this is set to True, only explicit I/O port constraints are forward annotated. When it is set to False (the default), all I/O port constraints are forward annotated.

This option is equivalent to the “set_option -auto_constraint_io 1 | 0” command in Synplify Pro.

**VHDL 2008 (for Synplify Pro)** When this is set to True, VHDL 2008 is selected as the VHDL standard for the project.

**Verilog Input** Specifies the Verilog standard used for the project.

The default is Verilog 2001.

This option is equivalent to the “set_option -vlog_std v2001 | v95 | sysv” command in Synplify Pro.

For information about Verilog 2001, refer to the Synopsys Synplify Pro for Lattice Reference Manual. Go to the “Verilog 2001 Support” section in the “Verilog Language Support” chapter.

# LSE Options

This page lists all the strategy options associated with the LSE synthesis process.

**Allow Duplicate Modules (for LSE)**

When set to True, allows the design to keep duplicate modules. LSE issues a warning and uses the last definition of the module. Any previous definitions are ignored. The default is False, which causes an error if there are duplicate modules.

**Carry Chain Length** Specifies the maximum number of carry chain cells (CCUs) that get mapped to a single carry chain. Default is 0, which is interpreted as infinite length.

This option is equivalent to the “-carry_chain_length” option in the SYNTHESIS command.

This option is equivalent to the “-allow_duplicate_modules” option in the SYNTHESIS command.

**Command Line Options (for LSE)** Enables additional command line options for the LSE Synthesis process.

**To enter a command line option:**

1. In the Strategy dialog box, select **LSE** in the Process list.

2. Double-click the Value column for the Command line Options option.
3. Type in the option and its value (if any) in the text box.
4. Click **Apply**.

For detailed description on LSE command line options, see “Running SYNTHESIS from the Command Line” on page 570.

**DSP Style** Specifies how DSP modules should be implemented: with DSP resources or with Logic (LUTs).

This option is equivalent to the “-use_dsp” option in the SYNTHESIS command.

**DSP Utilization** Specifies the percentage of DSP sites that LSE should try to use.

This option is equivalent to the “-dsp_utilization” option in the SYNTHESIS command.

**Decode Unreachable States** When set to True, synthesis infers safe recovery logic from unreachable states in all the state machines of the design.

This option is equivalent to the “-decode_unreachable_states” option in the SYNTHESIS command.

**Disable Distributed RAM** When set to True, inferred memory will not use the distributed RAM of the PFUs.

**EBR Utilization** Specifies EBR utilization target setting in percent of total vacant sites. LSE will honor the setting and do the resource computation accordingly. Default is 100 (in percentage).

This option is equivalent to the “-bram_utilization” option in the SYNTHESIS command.

**FSM Encoding Style** Specifies the encoding style to use with the design.

This option is equivalent to the “-fsm_encoding_style” option in the SYNTHESIS command. Valid options are auto, one-hot, gray, and binary. The default value is auto, meaning that the tool looks for the best implementation.

**Note**

The encoding type “gray” only works with less than or equal to four machine states. When the number of machine states is large than four, LSE will use other encoding styles and issue the following warning message:

***WARNING - Gray encoding is not supported for state machines with more than four states.***

**Fix Gated Clocks** When set to True, LSE changes standard gated clocks to forms more effective for FPGAs. Clocks are gated with AND or OR gates to conserve power, but in FPGAs such clocks cause skew and prevent global clock resources from being used. The Fix Gated Clocks option is ignored if the Optimization Goal option is set to Area.

The gated clocks must be specified in the .ldc file with create_clock constraints. All inputs of the gating logic must be driven by primary inputs and the gating logic must be decomposable. Instantiated primitives and black boxes are not affected.

Converted clocks and the associated registers are reported in the synthesis.log file.

**Hardware Evaluation (for LSE)** Enables or disables the ability to temporarily test IP in a device without an IP license. If enabled, a timer is added to the design that allows unlicensed IP to function for about 4 hours in a device. If disabled, you cannot generate a bitstream if there are any unlicensed IP in the design.

You might want to disable this option to refine your design while waiting for the license. You will not be able to generate a bitstream, but you will be able to see how resources are used (without the timer) and close timing. When you get the license, you can then generate the bitstream.

Regardless of how this option is set, if there are any unlicensed IP in the design, some features of the Radiant software, such as gate level simulation and EPIC, are blocked.

This option is equivalent to the “-dt” option in the SYNTHESIS command.

**Intermediate File Dump** If you set this to True, LSE will produce intermediate encrypted Verilog files. If you supply Lattice with these files, they can be decrypted and analyzed for problems. This option is good for analyzing simulation issues.

This option is equivalent to the “-ifd” option in the SYNTHESIS command.

**Loop Limit** Specifies the maximum number of iterations of "for" and "while" loops in the source code. The limit is applied when the loop index is a variable, not when it is a constant. The higher the loop_limit, the longer the run time. The default value is 1950. Setting a higher value may cause stack overflow during some of the optimizations during synthesis. A lower value will be ignored and the default used instead.

This option is equivalent to the “-loop_limit” option in the SYNTHESIS command.

**Macro Search Path (for LSE)** Allows you to specify a path (or paths) to locate physical macro files used in a given design. The software will add the specified paths to the list of directories to search when resolving file references. The option can also be used for indicating the directories containing include files that are specified in the RTL design files.

You don't need to specify a search path if the necessary .ngo or .nmc file is in the directory containing the top-level .ngo file or if the FILE attribute in the design gives a complete path name for the file (instead of a relative path name).

The software follows the following order to search for .ngo files:

1. Current implementation directory
2. Project directory
3. Directories where the LPC or IPX source files reside
4. User-specified macro search paths

To specify a macro search path, double-click the Value box, and directly enter the path or click the **...** button to browse for one or more paths.

This option is equivalent to the "-p" option in the SYNTHESIS command.

**Max Fanout Limit** Specifies the maximum fanout setting. LSE will make sure that any net in the design is not exceeding this limit. Default is 1000 fanouts.

This option is equivalent to the "-max_fanout" option in the SYNTHESIS command.

**Memory Initial Value File Search Path (for LSE)** Allows you to specify a path (or paths) to locate memory initialization file (.mem) used in a given design. The software will add the specified path(s) to the list of directories to search when resolving file references.

To specify a search path, double-click the Value box, and directly enter the path or click the **...** button to browse for one or more paths.

This option is equivalent to the "-p" option in the SYNTHESIS command.

**Number of Critical Paths (for LSE)** Specifies the number of critical timing paths to be reported in the timing report.

This option is equivalent to the "-twr_paths" option in the SYNTHESIS command.

**Optimization Goal** Enables LSE to optimize the design for area, speed, or balanced.

Valid options are:

- Area – Optimizes the design for area by reducing the total amount of logic used for design implementation.

When Optimization Goal is set to Area, LSE honors the LDC constraints if there are any. If Use IO Registers is set to Auto, LSE packs input and output registers into I/O pad cells.

**Note**

With the Area setting, LSE also ignores all SDC constraints. These constraints are not used by LSE and are not added to an .lpf file for use by the later stages of implementation.

- Timing – Optimizes the design for speed by reducing the levels of logic.

  When Optimization Goal is set to Timing and a create_clock constraint is available in an .ldc file, LSE ignores the Target Frequency setting and uses the value from the create_clock constraint instead.

  If there are multiple clocks, and if not all the clocks use create_clock constraint, then LSE will assign 200 MHz constraint on the remaining clocks in Timing Mode.

  If Use IO Registers is set to Auto, LSE does not pack input and output registers into I/O pad cells.

- Balanced – Optimizes the design for both area and timing.

  When Optimization Goal is set to Balanced, all timing driven optimizations based on static timing analysis will run depending on LDC constraints. If Use IO Registers is set to Auto, LSE does not pack input and output registers into I/O pad cells.

The default setting depends on the device type.

For more information, see “Optimizing LSE for Area and Speed” on page 164.

This option is equivalent to the “-optimization_goal” option in the SYNTHESIS command.

**Propagate Constants** When set to True (default), enables constant propagation to reduce area, where possible. LSE will then eliminate the logic used when constant inputs to logic cause their outputs to be constant.

You can turn off the operation by setting this option to False.

This option is equivalent to the “-propagate_constants” option in the SYNTHESIS command.

**RAM Style** Sets the type of random access memory globally to distributed, embedded block RAM, or registers.

The default is Auto which attempts to determine the best implementation, that is, the synthesis tool will map to technology RAM resources (EBR/Distributed) based on the resource availability.

This option will apply a syn_ramstyle attribute globally in the source to a module or to a RAM instance. To turn off RAM inference, set its value to Registers.

- Registers – Causes an inferred RAM to be mapped to registers (flip-flops and logic) rather than the technology-specific RAM resources.
- Distributed – Causes the RAM to be implemented using the distributed RAM or PFU resources.
- Block_RAM – Causes the RAM to be implemented using the dedicated RAM resources. If your RAM resources are limited, for whatever reason, you can map additional RAMs to registers instead of the dedicated or distributed RAM resources using this attribute.

This option is equivalent to the “-ramstyle” option in the SYNTHESIS command.

**ROM Style** Allows you to globally implement ROM architectures using dedicated, distributed ROM, or a combination of the two (Auto).

This applies the syn_romstyle attribute globally to the design by adding the attribute to the module or entity. You can also specify this attribute on a single module or ROM instance.

Specifying a syn_romstyle attribute globally or on a module or ROM instance with a value of:

- Auto (default) – Allows the synthesis tool to choose the best implementation to meet the design requirements for speed, size, and so on.
- Logic – Causes the ROM to be implemented using the distributed ROM or PFU resources. Specifically, the logic value will implement ROM to logic (LUT4) or ROM technology primitives (such as ROM16X1, ROM32X1, ROM64X1, and so on).
- EBR – Causes the ROM to be mapped to dedicated EBR block resources. ROM address or data should be registered to map it to an EBR block. If your ROM resources are limited, for whatever reason, you can map additional ROM to registers instead of the dedicated or distributed RAM resources using this attribute.

Infer ROM architectures using a CASE statement in your code. For the synthesis tool to implement a ROM, at least half of the available addresses in the CASE statement must be assigned a value. For example, consider a ROM with six address bits (64 unique addresses). The CASE statement for this ROM must specify values for at least 32 of the available addresses.

This option is equivalent to the “-romstyle” option in the SYNTHESIS command.

**Remove Duplicate Registers** Specifies the removal of duplicate registers.

When set to True (default), LSE removes a register if it is identical to another register. If two registers generate the same logic, the second one will be deleted and the first one will be made to fan out to the second one's destinations. LSE will not remove duplicate registers if this option is set to False.

This option is equivalent to the “-remove_duplicate_regs” option in the SYNTHESIS command.

**Remove LOC Properties (for LSE)** Setting this to On removes LOC properties in the synthesized design before building the Native Generic Database (.ngd) file.

**Resolve Mixed Drivers (for LSE)** If a net is driven by a VCC or GND and active drivers, setting this option to True connects the net to the VCC or GND driver.

**Resource Sharing (for LSE)** When this is set to True (default), the synthesis tool uses resource sharing techniques to optimize area.

With resource sharing, synthesis uses the same arithmetic operators for mutually exclusive statements; for example, with the branches of a case statement. Conversely, you can improve timing by disabling resource sharing, but at the expense of increased area.

This option is equivalent to the “-resource_sharing” option in the SYNTHESIS command.

**Target Frequency** Specifies the target frequency setting. This frequency applies to all the clocks in the design. If there are some clocks defined in an .ldc file, the remaining clocks will get this frequency setting. When a create_clock constraint is available in an .ldc file, LSE ignores the Target Frequency setting for that clock and uses the value from the create_clock constraint instead.

This option is equivalent to the “-frequency” option in the SYNTHESIS command.

**Timing Analysis Options (for LSE)** Specifies the analysis type.

- Hold Analysis – Performs hold analysis.
- Standard Setup Analysis – Performs setup time checks.
- Standard Setup and Hold Analysis – Performs both the Standard Setup Analysis and the Hold Analysis.

**Use Carry Chain** Turns on (True) or off (False) carry chain implementation for adders. Default is True.

This option is equivalent to the “-use_carry_chain” option in the SYNTHESIS command.

**Use IO Insertion** When set to True, LSE uses I/O insertion and GSR.

This option is equivalent to the “-use_io_insertion” option in the SYNTHESIS command.

**Use IO Registers** When True, this option forces the synthesis tool to pack all input and output registers into I/O pad cells based on the timing requirements for the target device family. Auto, the default setting, enables this register packing if Optimization Goal is set to Area. If Optimization Goal is Timing or Balanced, Auto disables register packing.

This option is equivalent to the "-use_io_reg" option in the SYNTHESIS command.

You can also control packing on individual registers and ports. See "syn_useioff" on page 477.

**VHDL 2008 (for LSE)** When this is set to True, VHDL 2008 is selected as the VHDL standard for the project.

# Post Synthesis Options

This page lists all strategy options associated with the Post Synthesis process. The options available for user setting are dependent on the target device of your project.

**External Module Files (.udb)**

Allows the user to supply already synthesized modules for design assembly into the top level design. These modules must be already synthesized in Unified Database (.udb) format.

For example:

c:\case1\ip1.udb; d:\example\aaa\abc.udb

Multiple .udb files can be separated by ';'

This internal process, Post Synthesis, performed after logic synthesis resolves and assembles lower level modules to top level to complete the design contents.

All modules contents must be resolved at this stage before the flow can continue to the next stage.

# Map Design Options

This page lists all strategy options associated with the Map Design process. The options available for user setting are dependent on the target device of your project.

**Command Line Options (for Map Design)** Enables additional command line options for the associated process.

**To enter a command line option:**

1. In the Strategy dialog box, select the associated process in the Process list.
2. Double-click the Value column for the Command line Options option.
3. Type in the option and its value (if any) in the text box. For example: -exp parPathBased=ON
4. Click **Apply**.

To reference more information about command line options for the Map Design process, type map -h <architecture> in a command line window. For detailed options description, refer to Running MAP from the Command Line.

**Infer GSR** Enables or disables the GSR inferencing.

The default is True. By default, if the input design (NGD) does not include a GSR buffer, MAP will infer one based on the signal that drives the most set/reset loads.

If this is set to False, all the GSR_NET constraints will be ignored.

**Report Signal Cross Reference** When this is set to True, the map report (.mrp) will show where nets in the logical design were mapped in the physical design.

The default is False.

**Report Symbol Cross Reference** When this is set to True, the map report (.mrp) will show where symbols in the logical design were mapped in the physical design.

The default is False.

**Unclip Unused Instances** Setting this strategy option to True instructs Map to not remove unused logic from your design. When this option is set to False, the mapper will remove unused logic from your design. The default is False.

# Map Timing Analysis Options

This page lists all strategy options associated with the Map Timing Analysis process.

**Number of End Points** Controls the number of endpoints in the critical endpoint summary.

**Number of Paths Per Constraint (for Map Timing Analysis)** Lists detailed logic and route delays for all constrained paths and nets in the design.

**Number of Paths Per Endpoint (for Map Timing Analysis)** Controls maximum number of paths that could be reported for each endpoint.

**Number of Unconstrained Paths (for Map Timing Analysis)** Reports paths not covered by a timing preference.

**Report Constraints (for Map Timing Analysis)** Controls reporting of SDC constraints

**Timing Analysis Options (for Map Timing Analysis)** Specifies the analysis type.

- Hold Analysis – Performs hold analysis.
- Standard Setup Analysis – Performs setup time checks.
- Standard Setup and Hold Analysis – Performs both the Standard Setup Analysis and the Hold Analysis.

# Place & Route Design Options

This page lists all strategy options associated with the Place & Route Design process. The options available for user setting are dependent on the target device of your project.

**Command Line Options (for Place & Route Design)** Allows you to specify options from the par command without directly using the command line. Type in a string of options without the par command. For example: `-exp parPathBased=ON:parHold=1`

For detailed descriptions of placement, routing, and PAR explorer (-exp) command line options, see Running PAR from the Command Line.

**Disable Auto Hold Timing Correction (for Place & Route Design)** When the switch is used, PAR will not check the hold timing of the design, so no correction of potential hold timing violations will be performed.

**Disable Timing Driven (for Place & Route Design)** Enables or disables the timing-driven option for the PAR run.

When this is set to True, the timing-driven option for the PAR run will not be used. If this is set to False, PAR automatically uses the timing-driven option if the Timing Wizard is present and if any timing constraints are found in the preference file. If selected, the timing-driven option is not invoked in any case and cost-based placement and routing are done instead.

Two examples of situations in which you might disable this option are:

- You have timing constraints specified in your preference file, but you want to execute a quick PAR run without using the timing-driven option to give you a rough idea of how difficult the design is to place and route.
- You only have a single license for the timing-driven option but you want to use this license for another application (for example, to perform timing-driven routing within EPIC) that will run at the same time as PAR. This option keeps the license free for the other application.

**Multi-Tasking Node List** Allows you to specify the node file name for the multi-tasking PAR.

The multi-tasking PAR allows you to use multiple machines (nodes) that are networked together for a multi-run PAR job, significantly reducing the total amount of time for completion.

For more information on multi-tasking PAR, see Running Multiple PAR Jobs in Parallel.

**Pack Logic Block Util.** Sets the relative density (of available slices) at which the slices within a device are to be packed, in terms of a percentage of the available slices in the device.

This option has great control of the packing density. The value range is 0-100 percent (100 = minimum packing; 0 = maximum density).

If this option is not specified (blank), it defaults to 97 percent. The result will be a less dense packing, depending on the size of the design relative to the number of available slices in the device. If the design is large compared with the number of available slices in the device, the mapper will make a reasonable effort to pack the design so that it fits in the device.

If you specify a density value for this option, the mapper will attempt to pack the device to that density. The “0” setting results in the densest mapping. If the design is large compared with the target density, the mapper will make an aggressive packing effort to meet your target. However, this may adversely impact the design $f_{MAX}$ performance.

**Path-based Placement** Allows you to apply path-based placement. Path-based placement gives better performance and more predictable results.

Options are Off and On (default).

**Placement Iteration Start Pt** Specifies the cost table to use (from 1-100) to begin the PAR run.

The default is 1. Cost tables are not an ordered set. There is no correlation between a cost table's number and its relative value. If cost table 100 is reached, placement does not begin at 1 again, even if command options specify that more placements should be performed.

**Placement Iterations** Specifies the maximum number of placement/routing passes (0-100, 0 = run until solved) to be run (regardless of whether they complete) at the Placement Effort Level.

Each iteration uses a different cost table when the design is placed and will produce a different Uniform Database (.udb) file. If you specify a Starting Cost Table, the iterations begin at that table number.

**Placement Save Best Run** Determines the number (1-100) of best outputs of the Place and Route run to save (defaults to 1).

If no number is specified, all output designs produced by the PLACE & ROUTE run are saved. The best outputs are determined by a scoring system described in the section titled Scoring the Routed Design.

This option does not care how many iterations you performed or how many effort levels were used. It compares every result to every other result.

**Placement Sort Best Run** Specifies how to sort the results of the Place and Route run in the PAR report. Results are sorted first by the number of unrouted connections. Then the results are ranked by timing. Choose whether you want to use the worst-slack value or the timing score to rank the results.

The default is Worst Slack.

**Prioritize Hold Correction Over Setup Performance (for Place & Route Design)** During hold timing correction, there may be situations when correcting a hold timing violation would cause a setup requirement to become violated. By default, we do not correct the hold violation on such connections so as to preserve the setup performance, but when this switch is used, we will attempt to correct the hold violation and let the setup requirement become violated.

**Remove Previous Design Directory** When this is set to True (default), the software removes the contents of the project design directory before Place and Route Design is rerun.

This prevents the accumulation of files from multiple Place & Route Design processes in the same directory.

**Run Placement Only** Setting this to True prevents the design from being routed. PAR will output a placed, but not routed Unified Design Database(.udb) file.

This option defaults to False.

**Set Speed Grade for Hold Optimization (for Place & Route Design)** This overrides the default speed grade used to perform hold timing analysis.

**Stop Once Timing is Met (for Place & Route Design)** Setting this to True forces the Place and Route Design process to stop as soon as the timing requirement is satisfied. This option has no effect if the "Generate Timing Analysis report for each iteration" option is set to True or if using Run Manager to produce multiple place-and-route runs.

# Place & Route Timing Analysis Options

This page lists all strategy options associated with the Place & Route Timing Analysis process.

**Number of End Points (for Place & Route Timing Analysis)** Controls the number of endpoints in the critical endpoint summary.

**Number of Paths Per Constraint (for Place & Route Timing Analysis)**

Lists detailed logic and route delays for all constrained paths and nets in the design.

**Number of Paths Per Endpoint (for Place & Route Timing Analysis)**

Controls maximum number of paths that could be reported for each endpoint.

**Number of Unconstrained Paths (for Place & Route Timing Analysis)** Reports paths not covered by a timing preference.

**Report Constraints (for Place & Route Timing Analysis)** Controls reporting of SDC constraints.

**Timing Analysis Options (for Place & Route Timing Analysis)** Specifies the analysis type.

- Hold Analysis – Performs hold analysis.
- Standard Setup Analysis – Performs setup time checks.
- Standard Setup and Hold Analysis – Performs both the Standard Setup Analysis and the Hold Analysis.

# Timing Simulation Options

This page lists all strategy options associated with the Timing Simulation process. The options available for user setting are dependent on the target device of your project.

**Generate PUR in the Netlist** When this is set to False, the timing simulation file generation process will not write PUR instance in the Verilog/ VHDL back-annotation netlist. Then you have to instantiate PUR in the test bench.

**Generate X for Setup/Hold Violation** When this is set to True, the Timing Simulation process will place X notifiers in the output file on flip-flops with setup and/or hold time violations.

**Hold Check Min Speed Grade** Setting this to True replaces all timing information for back annotation with the minimum timing for all paths.

This option is used for simulation of hold time requirements. Separate simulations are required for hold time verification (-min switch) and delay time verification (normal output).

**Multichip Module Prefix** Adds a prefix to module names to make them unique for multi-chip simulation.

**Negative Setup-Hold Times** Allows you to select negative setup time and negative hold time for better accuracy.

The default is True. You can set it to False for those simulators that might not be able to handle negative setup- hold times.

**Retarget Speed Grade** Retargets back annotation to a different performance grade than the one used to create the Uniform Database (.udb) file.

You are limited to those performance grades available for the device used in the UDB file.

**Timing Simulation Max Delay between Buffers (ps)** Distributes routing delays by splitting the signal and inserting buffers. The delay value assigned represents the maximum delay number in picoseconds between each buffer (1000 ps by default).

**Verilog Hierarchy Separator** Specifies the hierarchy separator character which will be used in name generation when the design hierarchy is flattened.

You can specify an alpha-numeric character or special character as the hierarchy separator.

For alpha-numeric characters, just enter the character as is in the edit box. For special characters (such as +, -, and so on), encapsulate the character with double quotes, for example, “+”, “-”.

The option is only available for Verilog designs.

# Bitstream Options

This page lists all strategy options associated with the bitstream generation process. The options available for user setting are dependent on the target device of your project.

**Initialize EBR Quadrant 0** Write the EBR initialization data into the bitstream for quadrant 0.

**Initialize EBR Quadrant 1** Write the EBR initialization data into the bitstream for quadrant 1.

**Initialize EBR Quadrant 2** Write the EBR initialization data into the bitstream for quadrant 2.

**Initialize EBR Quadrant 3** Write the EBR initialization data into the bitstream for quadrant 3.

**No header** Don't write the bitstream header section.

**Output Format** Specifies the type of bitstream to create for an FPGA device.

The following options are available:

- Bit File (Binary) – Generates a binary configuration file (.bit) that contains the default outputs of the Bit Generation process.
- Raw Bit File (ASCII) – Generates an ASCII raw bit text file (.rbt) of ASCII ones and zeros that represent the bits in the bitstream file. If you are using a microprocessor to configure a single FPGA, you can include the Raw Bit file in the source code as a text file to represent the configuration data. The sequence of characters in the Raw Bit file is the same as the bit sequence that will be written into the FPGA.
- Mask and Readback File (ASCII) – Generates an ASCII bitstream file that is used to compare bit locations and execute a readback of configuration data within an operating FPGA. When you select this option, BITGEN generates a an ASCII mask file (.mska) for the input design and an ASCII readback file (.rbka) for verifying the configuration data.
- Mask and Readback File (Binary) – The mask file is used to compare bit locations and execute a readback of configuration data within an operating FPGA. When you select this option, BITGEN generates a binary mask file (.msk) for the input design and a binary readback file (.rbk) for verifying the configuration data.

**Run DRC** When this is set to True, the software runs a physical design rule check and saves the output to the Bit Generation report file (.bgn).

Running DRC before a bitstream or JEDEC file is produced will detect any errors that could cause the FPGA to function improperly. If no fatal errors are detected, it will produce a bitstream or JEDEC file. Run DRC is the default.

# Constraints Reference Guide

Constraints are instructions applied to design elements that guide the design toward desired results and performance goals. They are critical to achieving timing closure or managing reusable intellectual property (IP). The most common constraints are those for timing and pin assignments, but constraints are also available for placement, routing, and many other functions. In the Radiant software design flow, constraints include logical constraints and HDL attributes.

You can assign constraints using one or both of the following methods:

- Assign Lattice design (timing) constraints (.PDC) via the Device Constraint editor tool.

  See the “Device Constraint Editor” on page 127 for complete descriptions of each constraint, including syntax rules and examples.

- Assign Lattice design timing constraints (.LDC) via the “Timing Constraint Editor” on page 136.

- Assign HDL or schematic-based attributes using design source files. These attributes are used to direct Map and Place & Route.

  See the “HDL Attributes” on page 404 for complete descriptions of each attribute, including conventions and examples.

If you are using the Lattice Synthesis Engine (LSE), constraints will also include Synopsys Design Constraints (SDC) as well as HDL attributes and directives that influence the optimization or structure of the output netlist. See the “Lattice Synthesis Engine Constraints” on page 416 for descriptions of these LSE constraints.

**See Also** “Applying Design Constraints” on page 115

- “Integrated Synthesis” on page 166

# HDL Attributes

HDL attributes are constraints that are attached as text to design objects and interpreted by the Radiant software. A design object can be a specific port, component pin, net, instance, instantiation, or even an entire design. An attribute provides information about the object. For example, an attribute might specify where a component in the logical design must be placed in the physical device, or it might specify a frequency constraint for a net that timing-driven place and route will attempt to meet.

The HDL attributes described in this section are those most commonly used as constraints in HDL source files. They generally apply to all designs and are not specific to any architecture.

HDL attributes are typically declared using comment notation in Verilog HDL or the VHDL Attribute keyword..

**Note**

A comprehensive guide to library element HDL attributes is available in the "FPGA Libraries Reference Guide" on page 490. For specific applications, refer to the technical notes that are available on the Lattice web site. Editing these library element attributes is not recommended, as they are usually generated from an array of choices that are made for module generation using IP Catalog.

## BBOX

**Convention** BBOX= x,x

**Description** Indicates the bounding box or the area given in number of rows and columns for a given GRP. The attribute must appear on the same block as the GRP attribute. This has replaced the old PBBOX attribute which is also still valid for backwards compatibility.

**Device Support** All

**Syntax** BBOX= H,W

where *H* is height, *W* is width.

For example, the following parameter indicates a BBOX that is three rows high and seven columns wide.

**Figure 41: Example of a BBOX Parameter**

```
BBOX= 3,7
```

You will see BBOX with the GRP definition below it.

# DRIVE

**Convention** DRIVE= NA, 2, 4, 8 (default), 12, 16, 24

**Description** Attached to bidirectional and output buffers (e.g., BB, BBPD, OB, OBW), the drive strength attribute is available for output standards that support programmable drive strength. Refer to the SysIO Usage Guides to see the IOTYPES that support valid drive strength standards.

The programmable drive available on a pad depends on the VCCIO. Also, not all drive strengths are available on an alternate PIO pad. Both the single-ended driver and differential current source driver have programmable drive strength. Only three drive settings are available when operating at 3.3 volts.

**Table 37: Valid Drive Strength Table**

| Drive Strength (mA) | VCCIO 1.2V | VCCIO 1.5V | VCCIO 1.8V | VCCIO 2.5V | VCCIO 3.3V |
|---|---|---|---|---|---|
| 2 | X | | | | |
| 4 | X | X | X | X | |
| 8 | X | X | X | X | X |
| 12 | X | X | X | X | |
| 16 | | X | X | X | X |
| 24 | | | | | X |

**Device Support** All

**VHDL Syntax** ATTRIBUTE DRIVE : string;
ATTRIBUTE DRIVE OF [pin_name]: SIGNAL IS "[drive_strength]";

**VHDL Example Code**
```
ATTRIBUTE DRIVE OF portD: SIGNAL IS "8";
```

**Verilog Syntax – Synplify** PinType [pin_name] /* synthesis IO_TYPE="[type_name]" DRIVE="[drive_strength]" PULLMODE="[mode]" SLEWRATE="[value]"*/;
[drive_strength] = 2, 4, 8, 12, 16, 20

**Note**: Example given for Pin I/O Type configuration.

**Verilog Example Code – Synplify**
```
//output  mypin /* synthesis IO_TYPE="LVTTL33D" DRIVE="16" PULLMODE="UP" SLEWRATE="FAST"*/;
```

**.ldc TCL command**
```
ldc_set_port -iobuf {DRIVE=16} get_ports [mypin]
```

**See Also** ▶IO_TYPE

# GRP

**Convention** GRP = <identifier>

**Description** Universal grouping construct. Use the GRP attribute to group blocks within different hierarchies or with no hierarchy.

You can add GRP anchor and bounding box information to the HDL with BBOX attribute and anchoring it with a SDC constraint. Alternatively, you can allow a GRP to float within a REGION.

**Device Support** All

**Syntax** GRP="<GRP_name>"

[BBOX="<height>,<width>"]

where:

<*GRP_name*> is a user-defined name.

<site_name> is a row/column Slice D location of the target device in the format R<n>C<m>D, an EBR site in the format EBR_R<n>C<m>, or a DSP site in the format DSP_R<n>C<m>.

<height> is the number of device rows of a rectangular bounding box.

<width> is the number of device columns of a rectangular bounding box.

The VHDL and Verilog examples show anchored GRPs. The current mechanism is to define the BBOX and GRP name in the attribute and the anchor location must be set via the SDC constraint ldc_set_location.

**VHDL Syntax** attribute GRP: string;
attribute GRP of <object>: label is "<GRP_name>";
attribute BBOX: string;
attribute BBOX of <object>: label is "<height>,<width>";

**VHDL Example Code**
```
attribute GRP: string;
attribute GRP of x: label is "reg_group";
attribute BBOX: string;
attribute BBOX of <object>: label is "9,7";
```

**Verilog Syntax – Synplify** /* synthesis GRP= "<GRP_name>"
BBOX= "<h,w>"
*/;

**Verilog Example Code – Synplify**
```
"module serial_reg_custom(D,
CLK, CE, RST, Q)
```

```
 /* synthesis GRP= ""reg_group""  GLOC= ""R5C19D""  BBOX=
""6,4*/;
serial_reg_custom inst1A(.D(A), .CLK(CLK), .CE(CE), .RST(RST),
.Q(adder1_in1));
serial_reg_custom inst1B(.D(B), .CLK(CLK), .CE(CE), .RST(RST),
.Q(adder1_in2));
 serial_reg_custom inst1C(.D(adder1_sum), .CLK(CLK), .CE(CE),
.RST(RST), .Q(SUM));"
*/;
```

**Note**

Synthesis directives to preserve hierarchy are typically needed to retain the instances constrained by the GRP attribute.

**.ldc TCL command** `"ldc_create_group  -name "reg_GROUP" -bbox {6 4} [get_cells {inst1A inst1B inst1C}]`
`ldc_set_location -site "R5C19D" [ldc_get_group "reg_GROUP"]`

**Required to set Anchor location using SDC constraint**

**Figure 42: Anchor Location using SDC Constraint**

```
ldc_set_location -site R16C6A [ldc_get_groups reg_group]
```

**GRP Attribute Usage Rules and Restrictions** Observe the following conditions for proper GRP attribute usage:

- All elements within the GRP block belong to that particular GRP.
- Nested GRP blocks are considered as unique individual GRPs.
- All blocks belong to the GRP attached to their nearest ancestor in the hierarchy.
- Unlike GRP, the mapper will not append the hierarchical path plus the block instance name.

# HYSTERESIS

**Convention** HYSTERESIS=SMALL | LARGE | NA

**Description** The ratioed input buffers have 2 input hysteresis settings. The HYSTERESIS option can be used to change the amount of hysteresis for the PCI, LVTTL and LVCMOS input and bidirectional I/O standards, except for the LVCMOS12 inputs.

The LVCMOS25R33, LVCMOS18R25, LVCMOS18R33, LVCMOS15R25, and LVCMOS15R33 input types do not support HYSTERESIS so this setting is always disabled for these types. HYSTERESIS is not supported for LVCMOS I/O types when used in an under-drive or over-drive mode.

The HYSTERESIS option for each of the input pins can be set independently when hysteresis is supported for the IO_TYPE assigned to the input pin.

**Device Support** All

**VHDL Syntax – Synplify** ATTRIBUTE HYSTERESIS: string;
ATTRIBUTE HYSTERESIS OF [signal_name]: SIGNAL IS "[ SMALL | LARGE | NA]";

**VHDL Example Code** ATTRIBUTE HYSTERESIS OF q_lvttl33_17: SIGNAL IS "LARGE ";

**Verilog Syntax – Synplify** /* synthesis attribute HYSTERESIS [of] signal_name [is] [SMALL | LARGE | NA] */;

**Verilog Example Code** /* synthesis attribute HYSTERESIS of q_lvttl33_17 is LARGE */;

**.ldc TCL command** ldc_set_port -iobuf {HYSTERESIS=SMALL} [get_ports q_lvttl33_17] */;

# INBUF

**Convention** INBUF= OFF | ON

**Description** INBUF is a global setting. All unused input buffers are disabled when INBUF is OFF to save power. When you need to perform boundary scan testing, you must turn ON the INBUF attribute. Turning ON this INBUF attribute will turn on the input buffers.

**Device Support** All

**VHDL Syntax** ATTRIBUTE INBUF : string;
ATTRIBUTE INBUF OF [module_name]: entity IS "[value]";

**VHDL Example Code**
```
ATTRIBUTE INBUF : string;
ATTRIBUTE INBUF OF behave: entity IS "OFF";
```

**Verilog Syntax – Synplify** module [module_name] (ports) /* synthesis INBUF= "[value]"*/;

**Verilog Example Code – Synplify**
```
module pci_top (portA, portB) /*
synthesis INBUF= "OFF" */;
```

**.ldc TCL command**
```
ldc_set_port -iobuf {INBUF=OFF} [get_ports
portA]
```

# INIT

**Convention** INIT= value
INIT=ENABLED | DISABLED

**Description** Initializes the look-up table values for all device families.

**Device Support** All

**VHDL Syntax** ATTRIBUTE INIT : string;
ATTRIBUTE INIT OF [module_name]: entity IS "[value]";

**VHDL Example Code**
```
ATTRIBUTE INIT : string;
ATTRIBUTE INIT OF behave: entity IS "ENABLED";
```

**Verilog Syntax – Synplify** module [module_name] (ports) /* synthesis INIT= "[value]"*/;

**Verilog Example Code – Synplify**
```
module pci_top (portA, portB) /*
synthesis INIT= "ENABLED" */;
```

**.ldc TCL command**
```
ldc_set_port -iobuf {INIT=ENABLED}
[get_ports portA]
```

# IO_TYPE

**Convention** IO_TYPE= buffer type

**Description** This is used to set the I/O standard for an I/O (input, output, and bidirectional buffers, e.g., IB, OB, and BB). Multiple values may be required that differ slightly depending on which element you are using. The VCCIO required to set these IO standards are embedded in the attribute names. There is no separate attribute to set the VCCIO requirements.

This attribute is used in conjunction with the IOBUF constraint. Refer to IOBUF to see how global constraints are honored for IO_TYPE. For valid buffer types, refer to the specific element in the FPGA Libraries Help. See the sysIO usage guides for legal IO_TYPE for inputs and outputs.

Examples of attribute usage for VHDL and Verilog are included in this topic.

**Device Support** All

**Values** LVDS, BLVDS25, MLVDS25, RSDS, LVPECL25, LVPECL33, HSTL15_I, HSTL15_II, HSTL15_III, HSTL15_IV, HSTL15D_I, HSTL15D_II, HSTL18_I, HSTL18_II, HSTL18_III, HSTL18_IV, HSTL18D_I, HSTL18D_II, SSTL18_I, SSTL18_II, SSTL18D_I, SSTL18D_II, SSTL25_I, SSTL25_II, SSTL25D_I, SSTL25D_II, SSTL33_I, SSTL_II, SSTL33D_I, SSTL33D_II, GTLPLUS15, GTL12, LVTTL33, LVTTL33D, LVCMOS33, LVCMOS25, LVCMOS18, LVCMOS15, LVCMOS12, PCI33, PCIX33, PCIX15, AGP1X33,

AGP2X33, LVCMOS25D, LVCMOS33D, LVCMOS18D, LVCMOS15D, LVCMOS12D

**Default** LVCMOS25

**VHDL Syntax** ATTRIBUTE IO_TYPE : string;
ATTRIBUTE IO_TYPE OF [pin_name]: SIGNAL IS "[type_name]";

**Example Code:**
```
ATTRIBUTE IO_TYPE OF portA: SIGNAL IS "PCI33";
ATTRIBUTE IO_TYPE OF portB: SIGNAL IS "LVCMOS33";
ATTRIBUTE IO_TYPE OF portC: SIGNAL IS "SSTL33_II";
ATTRIBUTE IO_TYPE OF portD: SIGNAL IS "LVCMOS25";
```

**Verilog Syntax – Synplify** PinType [pin_name] /* synthesis IO_TYPE="[type_name]" DRIVE="[drive_strength]" PULLMODE="[mode]" SLEWRATE="[value]"*/;
[value] = (valid I/O type, e.g., LVCMSO18)

**Note**: Example given for Pin I/O Type configuration.

**Verilog Example Code – Synplify** output [4:0] portA /* synthesis IO_TYPE="LVTTL33" DRIVE="16" PULLMODE="UP" SLEWRATE="FAST"*/;

**.ldc TCL command**
```
ldc_set_port -iobuf {IO_TYPE=LVTTL33}
[get_ports portA[1]]
```

# LOC

**Convention** LOC= site_name (PIN | POS)

**Description** Specifies a site location for the component that is created when this block is mapped. Attaching to multiple blocks indicates that these blocks are to be mapped together in the specified site.

When the device is mapped, the Map Design process (**map**) first maps blocks to PLCs without looking at LOC attributes. It then attempts to merge all PLCs with the same LOC attribute into a single PLC, and then it assigns this PLC to the site with the specified site name. If all of the blocks with the LOC attribute cannot be mapped into a single component, MAP groups together as many as possible.

When it encounters a LOC constraint, MAP also writes a LOCATE constraint for the component into the constraint file. The LOCATE constraint locks the component to the site. When you run PAR, the component cannot be unplaced, moved, swapped, or deleted.

The LOC attribute can be attached to anything that will end up on an I/O cell, and to clocks and internal flip-flops, but it should not be attached to combinational logic that will end up on a logic cell. Doing so could fail the generation of a locate constraint. The LOC attribute overrides register ordering.

**Device Support** All

**VHDL Syntax** ATTRIBUTE LOC : string;
ATTRIBUTE LOC OF [pin_name]: SIGNAL IS "[site_name]";

**VHDL Example Code**
```
ATTRIBUTE LOC : string;
ATTRIBUTE LOC OF output_vector : SIGNAL IS "H5";
```

**Verilog Syntax – Synplify** PinType [pin_name] /* synthesis LOC="[site_name]"*/;

**Verilog Example Code – Synplify**
```
//input rst  /* synthesis LOC="H5" */ ;
```

**.ldc TCL command**
```
ldc_set_location -site H5 [get_ports rst]

ldc_define_attribute -attr loc -value A12 -object_type port -object {clk1}
```

The following examples show slice locations for packing registers:

**VHDL Example Code for Register Packing**
```
ATTRIBUTE LOC : string;
ATTRIBUTE LOC OF inreg : SIGNAL IS "R2C9D R2C9D R2C8D R2C8D R2C7D R2C7D R2C6D R2C6D R2C5D R2C5D R2C4D R2C4D R2C3D R2C3D R2C2D R2C2D";
```

# NOCLIP

**Convention** NOCLIP=1

**Description** Assigned to a net or an instance, indicates that the net or instance cannot be removed as unused logic.

When the design is mapped, the default MAP setting removes any unused logic from the design. In most cases, you will want to do this, since unused logic takes up device components and routing resources unnecessarily. However, when you want to keep unused logic in the design, place a NOCLIP attribute on selected nets, and the nets on which you place the attribute are protected from removal, along with any valid logic connected to these nets.

For a path to be considered valid, it must have a valid input and a valid output.

Valid path inputs include:

- An input pad or the input portion of a bidirectional pad.
- A pullup or pulldown resistor.
- An oscillator primitive.
- A logic 1 or logic 0 primitive.
- A net with a NOCLIP attribute on it.

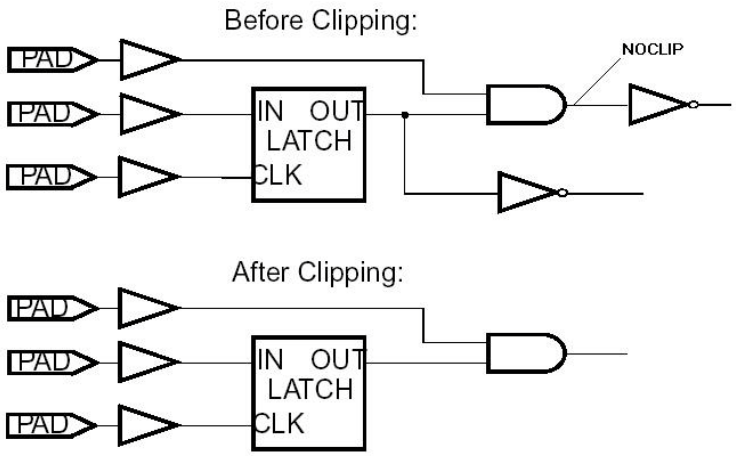
Valid path outputs include:

- An output pad or the output portion of a bidirectional pad.
- A net with a NOCLIP attribute on it.

If a path is not valid, the MAP program removes its logic as unused. To protect all or part of an invalid path from being removed, add a NOCLIP attribute to the appropriate net. The attribute can act as either a valid input or a valid output, depending on what is necessary to complete the path.

Figure 43 shows an example of how a NOCLIP attribute affects clipping:

**Figure 43: Example of NOCLIP Attribute Affects Clipping**




Some notes about using the NOCLIP attribute:

- If you design an on-chip oscillator (by building one from components in the device instead of using an oscillator primitive), the MAP program assumes the logic is unused and removes it because the oscillator logic does not contain a valid driver. To retain your oscillator logic, protect it by placing NOCLIP on the appropriate nets.
- If you design a circuit with a feedback loop and the looped signal is not driving any other loads but the loop itself, the loop is eliminated as unused logic. If you want to keep the loop in the design, make sure you protect the loop with a NOCLIP attribute.

**Device Support** All

**Verilog example** `wire net_a /* synthesis syn_keep=1 */ /* synthesis NOCLIP=1 */;`

or

```
wire net_a /* synthesis syn_keep=1  NOCLIP=1 */;
```

"syn_keep" is a logic synthesis directive. "NOCLIP" applies to physical design. Both need to be specified if you want to keep the net throughout the flow and only use syn_keep for synthesis cross-probing without NOCLIP.

**VHDL example**

```
signal net_a: std_logic;

attribute syn_keep : boolean;
attribute NOCLIP : boolean;

attribute syn_keep of net_a : signal is true;
attribute NOCLIP of net_a : signal is true;
```

Although NOCLIP is declared as boolean in VHDL, synthesis tool writes NOCLIP=1 in synthesis output.

**.ldc TCL command**

```
ldc_set_attribute {NOCLIP} [get_nets net_a]
```

# NOMERGE

**Convention** NOMERGE= <net_name>

**Description** Assigned to a net, specifies that the net cannot be absorbed into a logic block when the design is mapped. This may happen, for example, if the components connected to each side of a net are mapped into the same logic block. The net may then be absorbed into the block containing the components.

If you want to ensure that a net remains outside of a logic block, attach a NOMERGE attribute to the net. The MAP program then maps the net in a way that ensures you do not “lose” the net inside a logic block.

**Device Support** All

**VHDL Syntax** ATTRIBUTE NOMERGE: string;
ATTRIBUTE NOMERGE OF [signal_name]: SIGNAL IS "ON";

**VHDL Example Code**

```
ATTRIBUTE DRIVE OF q_lvttl33_17: SIGNAL IS "ON";
```

**Verilog Example Code – Synplify**

```
output q_lvttl33_17 /* synthesis NOMERGE="ON" */;
```

**.ldc TCL command**

```
ldc_set_location {NOMERGE} [get_nets q_lvttl33_17]
```

**See Also** ▶“IO_TYPE” on page 409

# PULLMODE

**Convention** PULLMODE= UP (default), DOWN, NONE, KEEPER, PCICLAMP

**Description** Attached to output buffer elements (e.g., OB, OBW) and bidirectional buffers, the PULLMODE attribute mode parameters are UP, DOWN, NONE, KEEPER, PCICLAMP. The PULLMODE options can be enabled for each I/O independently. The default is UP.

**Device Support** All

**VHDL Syntax** ATTRIBUTE PULLMODE: string;
ATTRIBUTE PULLMODE OF [pin_name]: SIGNAL IS "[mode]";

**Example Code:**
```
ATTRIBUTE PULLMODE OF md: SIGNAL IS "PCICLAMP";
```

**Verilog Syntax – Synplify** PinType [pin_name] /* synthesis IO_TYPE="[type_name]" DRIVE="[drive_strength]" PULLMODE="[mode]" SLEWRATE="[value]"*/;

[mode] = UP (default), DOWN, NONE, KEEPER, PCICLAMP

**Note**: Example given for Pin I/O Type configuration.

**Verilog Example Code Synplify**
```
output [4:0] portA /* synthesis IO_TYPE="LVTTL33_OD" DRIVE="16" PULLMODE="UP" SLEWRATE="FAST"*/ ;
```

**.ldc TCL command**
```
ldc_set_port -iobuf {PULLMODE=UP [get_ports port A[4]]}
```

There are also example files for other FPGA architectures available in corresponding directories in the <install_dir>/ispcpld/examples path.

**See Also** ▶"IO_TYPE" on page 409

# RBBOX

**Convention** RBBOX = H,W (where *H* is height, *W* is width)

**Description** RBBOX indicates the area size a region. This attribute must appear on the same block as REGION attribute (old attribute is PREGION). Required if REGION exists. This has replaced the old PRBBOX attribute which is also still valid for backwards compatibility.

**Device Support** All

**See Also** ▶"GRP" on page 406 (attribute)

# REGION

**Convention** REGION = <identifier>

**Description** REGION indicates the region to which a given GRP belongs. This attribute must appear on a block that has a GRP attribute. This has replaced the old PREGION attribute which is also still valid for backwards compatibility.

**Device Support** All

**VHDL Syntax** attribute REGION: string;
attribute REGION of <object>: label is "<REGION_name>";
attribute RLOC: string;
attribute RLOC of <object>: label is "<site>";
attribute RBBOX: string;
attribute RBBOX of <object>: label is "<height>,<width>";
attribute GRP: string;
attribute GRP of <object>: label is "<reg_GRP>";
attribute GLOC: string;
attribute GLOC of <object>: label is "<site>";
attribute BBOX: string;
attribute BBOX of <object>: label is "<height>,<width>";

**Example VHDL Code:**

```
attribute REGION: string;
attribute REGION of x: label is "reg_group";
attribute RLOC: string;
attribute RLOC of <object>: label is "R5C19D";
attribute RBBOX: string;
attribute RBBOX of <object>: label is "20,15";
attribute GRP: string;
attribute GRP of <object>: label is "reg_GRP";
attribute GLOC: string;
attribute GLOC of <object>: label is "R10C20D";
attribute BBOX: string;
attribute BBOX of <object>: label is "5,5";
```

**Verilog Syntax – Synplify** PinType [pin_name] /* synthesis IO_TYPE="[type_name]" DRIVE="[drive_strength]" PULLMODE="[mode]" SLEWRATE="[value]"*/;

[mode] = UP (default), DOWN, NONE, KEEPER, PCICLAMP

**Note**: Example given for Pin I/O Type configuration.

**Verilog Example Code Synplify**

```
"module serial_reg_custom(D, CLK,
CE, RST, Q) /*  synthesis REGION="reg_REGION"
 RLOC= "R5C19D"
 RBBOX= "20,15"
 GRP= "reg_GRP"
 GLOC="R10C20D"
 BBOX= "5,5"*/;

serial_reg_custom inst1A(.D(A), .CLK(CLK), .CE(CE), .RST(RST),
.Q(adder1_in1));
serial_reg_custom inst1B(.D(B), .CLK(CLK), .CE(CE), .RST(RST),
.Q(adder1_in2));
 serial_reg_custom inst1C(.D(adder1_sum), .CLK(CLK), .CE(CE),
.RST(RST), .Q(SUM));
```

```
count count_inst(A,CLK,RST) /* synthesis REGION="reg_REGION" */
;
```

**.ldc TCL command** `"ldc_create_region -name "reg_REGION" -site "R5C19D" -width 20 -height 15`

```
ldc_create_group  -name "reg_GROUP" -bbox {5 5} [get_cells {inst1A inst1B inst1C}]
ldc_set_location -site "R10C20D" [ldc_get_group "reg_GROUP"]
ldc_set_location -region "reg_REGION" [get_cells  {inst1A inst1B inst1C count_inst}]
```

**See Also** ▶"GRP" on page 406 (attribute)

# Lattice Synthesis Engine Constraints

The LSE, enables you to use constraints that are directly interpreted by the synthesis engine. This new category of constraints includes "Synopsys Design Constraints" on page 416, and "Lattice Synthesis Engine-Supported HDL Attributes" on page 430. These constraints influence the optimization or structure of the output netlist.

**See Also** ▶"Integrated Synthesis" on page 166

▶ "Applying Lattice Pre-Synthesis Timing Constraints" on page 123

## Synopsys Design Constraints

This section describes the SDC language elements for timing-driven synthesis that are supported by the LSE. When you use LSE, these SDC constraints are saved to a Lattice Design Constraints file (.ldc). A new .ldc file can be created and edited using the LDC Editor or the Source Editor. You can select the .ldc file to be the active synthesis constraint file for an implementation.

The SDC constraints will drive optimization of the design if LSE's Optimization Goal is set for timing in the active strategy file.

**Considerations about LSE Timing** The module delay in LSE is based on the primitive level, before logic is packed into a slice. Therefore, the delay might be different from the slice-based MAP or PAR Timing analysis report (.twr). Also, some elements are modeled as black boxes in LSE; for example, EFB in MachXO2.

The routing delay algorithm in LSE is different from the estimated routing delay in Map or Place.

The current LSE timing does not take the PLL/DLL frequency or phase shift properties into account. It also does not model the different IO_TYPE in the PIO. Therefore, it is necessary to adjust the timing constraint. For example, you can explicitly include a timing constraint on the PLL outputs with the phase-shift property.

**See Also** "Design Objects" on page 417

- "create_clock" on page 419
- "create_generated_clock" on page 421
- "set_clock_groups" on page 422
- "set_clock_latency" on page 423
- "set_clock_uncertainty" on page 424
- "set_false_path" on page 425
- "set_input_delay" on page 425
- "set_max_delay" on page 426
- "set_min_delay" on page 427
- "set_multicycle_path" on page 428
- "set_output_delay" on page 429
- "Integrated Synthesis" on page 166
- "Applying Lattice Pre-Synthesis Timing Constraints" on page 123

## Design Objects

Design objects can be referred to in the SDC as a single object, or as a collection of objects. Single objects must be referred to as a collection of a single object. The current implementation of the SDC commands allows only single objects in the collection. The exception to this are the all_* commands.

In the examples below, <*target name*> is a regular expression that matches one object only.

### Clock Object

| SDC Collection | Description |
|---|---|
| [get_clocks <*target name*>] | <*target name*> is the name of the clock. Clock is either given a name or gets its name from the port/net on which it is defined. |
| [all clocks] | Collection of all clocks in the design. |

**Examples:** [get_clocks clock_fast]

[all_clocks]

### Port Object

| SDC Collection | Description |
|---|---|
| [get_ports <*target name*>]: | Collection of a design port that matches <*target name*> |
| [all_inputs] | Collection of all design input ports |
| [all_outputs] | Collection of all design output ports |

**Examples:** [get_ports indata*]

[all_inputs]

### Cell Object

| SDC Collection | Description |
|---|---|
| [get_cells <*target name*>]: | Collection of a design instance that matches <*target name*> |

### Net Object

| SDC Collection | Description |
|---|---|
| [get_nets <*target name*>]: | Collection of a design net that matches <*target name*><br>Hierarchy is specified using regular expression. |

### Pin Object

| SDC Collection | Description |
|---|---|
| [get_pins <*instance name*> \| <*pin name*>]: | Collection of a design pin that matches <*target name*><br>Hierarchy is specified using regular expression. |

The get_nets command can used to define the target for a create_generated_clock command.

### Wildcard support

Only one wildcard (*) is supported. In addition, the wildcard should be at the end or beginning of the object name string. For example:

ab* matches abc, ab, abdefg, etc.

*bc matches abc, bbc, debc, etc.

The following is not supported:

a*b or a*b*c

**See Also** ▶"Synopsys Design Constraints" on page 416

## create_clock

Creates a clock and defines its characteristics.

**Note**

In LSE timing, interclock domain paths are always blocked for create_clock. However, the interclock domain path is still valid for constraints such as set_false_path and set_multicycle_path.

**Syntax** create_clock -period <period>[-name <clock name>] [-waveform <value1 value2>] [[get_ports | get_pins | get_nets source_object]]

**Arguments** -name *name*

The name string specifies the name of the clock. If this parameter is not given, the name of the source object is used as the name of the clock.

-period *period_value*

This value is required and it specifies the clock period in nanoseconds. The value you specify is the minimum time over which the clock waveform repeats. The value specified for the period must be positive as the period of a clock must be greater than zero. The duty cycle of the clock is 50 percent.

*-waveform {value1 value2}*

The values are a list of edge values. Only two edges are supported. Floating values are accepted. Value1 must be less than value2, and the difference must be less than the clock period. Note that each value must also be less than the clock period.

*source_object*

The source object is the object on which the clock constraint is defined. The source object can be a port object or a net object in the design. The object is obtained by using one of the get_ports or get_nets commands. If you specify a clock constraint on a source object that already has a clock, the new clock replaces the existing one. Only one source object is accepted. Wildcards are accepted as long as the resolution shows one port or net object.

**Example** The following example creates two clocks on ports CK1 and CK2 with a period of 6:

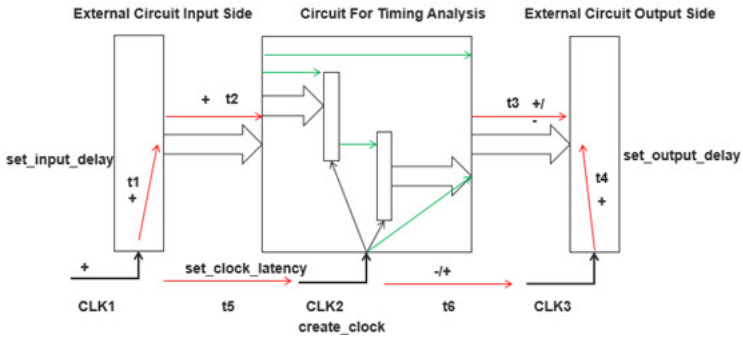
```
create_clock -name my_user_clock -period 6 [get_ports CK1]
create_clock -name my_other_user_clock -period 6 [get_ports CK2]
```

**Example** The following example creates a clock on port CK3 with a period of 7.1, and has two edges at 0 and 4.1:

```
create_clock -period 7.1 -waveform {0 4.1} [get_ports CK3]
```

**Example** Radiant Virtual Clocks are external to the FPGA and only trigger flip flops that are ouside the FPGA. These clocks do not trigger flip flops from inside the FPGA and are only used as the clocks are constrained with set_input_delay and set_output_delay.

**Example** In the following example, CLK1 and CLK3 are virtual clocks.




```
Input and output delay constraints must take external delays into
consideration.
Input side:
create_clock -period T -name CLK1
create_clock -period T -name CLK2 [get_ports CLK]
set_clock_latency t5 [get_clocks CLK2]
set_input_delay t1 + t2 -clock [get_clocks CLK1] [all_inputs]

Output side:
create_clock -period T -name CLK3
set_output_delay t3 + tr - t6 -clock [get_clocks CLK3]
[all_outputs]

If ther is clock uncertainty:
set_clock_uncertainty tu [get_clocks CLK2]
```

**See Also** ▶"Synopsys Design Constraints" on page 416

## create_generated_clock

Creates an internally generated clock and defines its characteristics. This command is used when the clock being created is related to another clock. The generated clock is considered a clock when defining constraints such as input_delay.

**Syntax** create_generated_clock –source [get_ports | get_pins/get_nets <*reference_object*> ] [ [ [-divide_by <factor>] [-multiply_by <factor>] [-duty_cycle <value>] ] | [-edges <edge specs>] ] [-invert] [-name <clock name>] [get_pins | get_nets <pin/net name>] net_object

**Arguments** -source *reference_object*

The reference object is an object on which the source clock of the generated clock is defined. The source object can be a net object or a port object. The period of the generated clock is derived from the clock on the reference object using the multiply and divide factors.

-divide_by *factor*

This factor is the frequency division factor. The frequency of the generated clock is equal to the frequency of the source clock divided by this factor, if the multiply by factor is not specified. For instance, if this factor is equal to 2, the generated clock period is twice the reference clock period. Default value is 1.

-multiply_by *factor*

This factor specifies the frequency multiplication number to be used when finding the generated clock frequency. For instance, if the factor is equal to 2, the generated clock period is half the reference clock period. If both multiply_by and divide_by factors are used, the frequency is obtained by using both factors. Default value is 1.

-duty_cycle *value*

This value specifies the duty cycle in percentage of the clock period. The value can be floating point and ranges from 0 to 100. The default value is 50.

-edges *value*

Rising or falling edge.

-invert

This inverts the clock edge.

*net_object*

The net_object specifies the source of the clock constraint. This is usually an internal -net of the design. If you specify a clock constraint on a net that already has a clock, the new clock replaces the existing clock. Only one source is accepted. Wildcards are accepted as long as the resolution shows one net.

This command creates a generated clock in the current design at a declared net_object by defining its frequency with respect to the frequency at the reference object. The static timing analysis tool uses this information to compute and propagate the generated clock's waveform

across the clock network to the clock pins of all sequential elements driven by this target

**Examples** The following example creates a generated clock on pin pll1/CLKOP with a period twice as long as the period at the reference port CLK:

```
create_generated_clock -divide_by 2 -source [get_ports CLK]
[get_pins pll1/CLKOP]
```

The following example creates a generated clock at the primary output of myPLL with a period three quarters of the period at the reference pin clk:

```
create_generated_clock -divide_by 3 -multiply_by 4  -source
[get_ports clk]  [get_pins myPLL/CLK1]
```

The following example shows a clock with a duty cycle of 60 percent:

```
create_generated_clock -duty_cycle 60 -source [get_ports clk]
[get_pins myPLL/CLK1]
```

**See Also** ▶"Synopsys Design Constraints" on page 416

## set_clock_groups

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design so that the paths between these clocks are not considered during timing analysis.

**Syntax** set_clock_groups *-asynchronous* | *-exclusive* -group *clock_objects* [-group *clock_objects*]

set_clock_groups [-logically_exclusive | -physically_exclusive | -asynchronous] –group [get_clocks <*clock_objects*>] -group [get_clocks <*clock_objects*>]

**Arguments** -asynchronous

Specifies that the clock groups are asynchronous to each other (while the Radiant software assume all clocks are asynchronous). Two clocks are asynchronous with respect to each other if they have no phase relationship at all.

-pyhsically_exclusive

Specifies that clocks are mutually exclusive. Only one clock group is active at any given time.

-logically_exclusive

Specifies that clocks are mutually exclusive and will not reach the flip flops at the same time due to logic implementation.

-group *clock_object*

Specifies the clock objects in a group. Specifying one group indicates that the clocks in that group are exclusive or asynchronous with all other clocks in the design. A default other group is created for this single group. Whenever a new clock is created, it is automatically included in this group.

**Examples** The following example specifies two clock ports (clka and clkb) are asynchronous to each other.

```
create_clock -period 10.000 -name clka_port [get_ports clka]
create_clock -period 10.000 -name clkb_port [get_ports clkb]
# Set clka_port and clkb_port to be mutually exclusive clocks.
set_clock_groups -asynchronous -group [get_clocks clka_port] -
group [get_clocks clkb_port]
# The previous line is equivalent to the following two
commands.
set_false_path -from [get_clocks clka_port] -to [get_clocks
clkb_port]
set_false_path -from [get_clocks clkb_port] -to [get_clocks
clka_port]
```

The following example specifies four clock constraints that won't be active at the same time:

```
create_clock -period 10.000 -name clka_port [get_ports clka]
create_clock -period 10.000 -name clkb_port [get_ports clkb]
create_clock -period 10.000 -name clkc_port [get_ports clkc]
set_clock_groups -exclusive -group [get_clocks {clka_port
clkb_port}] -group [get_clocks clkc_port]
```

**See Also** ▶"Synopsys Design Constraints" on page 416

## set_clock_latency

Specifies the behavior of the clock outside of the FPGA.

**Syntax** set_clock_latency <value> -source [-rise] [-fall] [-early] [-late] [get_clocks <clock name>]

The –source option of the constraint is required because only source latencies are supported by the timer. The value of the latency is the delay of the clock outside the FPGA. The value could be a rise, fall, early or late value. The max and min flags indicate delays that will be used for setup and hold requirement calculations. The early delay will be used as the capture clock delay for setup calculations and the launch delay for hold time calculations. The late delay will be used on the launch clock for setup and on the capture

clock for hold. The rise and fall options indicate the delay of the rising and falling edges of the clock.

**Examples:**

```
set_clock_latency 3 -source -early [get_clocks clk1]
set_clock_latency 4 -source -late [get_clocks clk1]
```

Setup analysis always uses longest path and hold analysis uses shortest path. In setup analysis, “early" value is used to calculate required time, "late" value is to calculate arrival time. While in hold analysis, the "late" value is used to calculate the required time and "early" is used to calculate arrival time.

In the example above, 3ns is in the destination clock of setup and source clock of hold; while 4ns is in source clock of setup and destination clock of hold.

## set_clock_uncertainty

This constraint indicates that the clock of interest has uncertainties in its period.

**Syntax** set_clock_uncertainty <value> [-from <from clock>] [-to <to clock>] [-setup] [-hold] [get_clocks <clock list>]

This constraint requires at least one of the clock options. The number given by value, is the amount by which the clock period in uncertain. The timer will use this value to add additional requirement to all paths affected by the selected clock. The timer requires the ‘from’ option to be accompanied by a ‘to’ option. The setup option indicates the constraint for setup analysis and the hold constraint for hold analysis.

**Examples** The following two examples specify clock uncertainty, one with a from/to.

```
set_clock_uncertainty 0.5 [get_clocks clk1]
set_clock_uncertainty 0.4 -from [get_clocks clk2] -to
[get_clocks clk3]
```

The first constraint sets the uncertainty of clk1 to half a nanosecond and the second constraint sets the uncertainty to 400 picoseconds when a path is launched by clk2 and captured by clk3.

## set_false_path

Identifies paths that are considered false and excluded from timing analysis.

The command requires a minimum of one from, through or to option. This constraint defines paths that should be removed from consideration during timing analysis. All paths that start at one of the 'from' objects, pass through one of the 'through' objects and arrive at one of the 'to' objects will be treated as an unconstrained path. If any of the options are missing, all objects in the design that satisfy the criteria of the missing option will be used as objects of the missing option. A missing "from" option, for instance, implies that all timing starting points are part of the 'from' list for the constraint.

**Syntax** set_false_path [(-from | rise_from | fall_from)<object_list>] [-through object_list] [(-to | -rise_to | -fall_to) <object_list>]

<object_list> = [ (get_clocks | get_ports | get_pins | get_nets | get_cells) <names>]

**Arguments** -from | rise_from | fall _from *from object_list*

Specifies the timing path start point. A valid timing starting point is a clock, a primary input, a combinational logic cell, or a sequential cell (clock-pin).

-to *to object_list*

Specifies the timing path end point. A valid timing end point is a primary output, a combinational logic cell, or a sequential cell (data-pin).

-through *object_list*

Specifies a net through which the paths should be blocked.

**Examples** The following example specifies all paths from clock pins of the registers in clock domain clk1 to data pins of a specific register in clock domain clk2 as false paths:

```
set_false_path –from [get_ports clk1] –to [get_cells reg_2]
```

The following example specifies all paths through the net UO/sigA as false:

```
set_false_path –through [get_nets UO/sigA]
```

**See Also** ►"Synopsys Design Constraints" on page 416

## set_input_delay

Defines the arrival time of an input relative to a clock.

**Syntax** set_input_delay (-clock | -clock_fall) <*clock_object*> [-min | -max] [-add_delay] <*delay_value*> <*input_port_object*>

<*input_port_object*> = [get_ports <names>] | [all_inputs]

**Arguments** *delay_value*

Specifies the arrival time in nanoseconds that represents the amount of time for which the signal is available at the specified input after a clock edge.

-max

Specifies that the delay value is the maximum delay.

-min

Specifies that the delay value is the minimum delay.

-clock | clock_fall *clock_object*

Specifies the clock reference to which the specified input delay is related. clock_fall is on the falling edge of clock. This is a mandatory argument.

*input_port_object*

Provides one or more input ports in the current design to which delay_value is assigned. You can also use the keyword “all_inputs” to include all input ports.

*add_delay*

Used to define more than one output delay on a given port.

**Example** The following example sets an input delay of 1.2 ns for port data1 relative to the rising edge of CLK1:

```
set_input_delay 1.2 -clock [get_clocks CLK1] [get_ports data1]

set_output_delay 4.0 –clock [get_clocks vclk] –add_delay
[get_ports out2]
```

**Example** The following example sets an input delay of 1.2 ns minimum and 1.5 ns maximum for port data1 relative to the rising edge of CLK1:

```
set_input_delay 1.2 -min -clock [get_clocks CLK1] [get_ports
data1]
set_input_delay 1.5 -max -clock [get_clocks CLK1] [get_ports
data1]
```

**See Also** ►“Synopsys Design Constraints” on page 416

## set_max_delay

Specifies the maximum delay for the timing paths.

**Syntax** set_max_delay [(-from)<*port_object or cell_object*>] [-through *port_object or cell_object*] [(-to) <*port_object or cell_object*>] <*delay_value*>

<*port_object or cell_object*> = [ (get_ports | get_pins | get_nets | get_cells) <names>]

**Arguments** *delay_value*

Specifies a floating point number in nanoseconds that represents the required maximum delay value for specified paths.

If the path ending point is on a sequential device, the tool includes library setup time in the computed delay.

-from *from port_object or cell_object*

Specifies the timing path start point. A valid timing start point is a clock, a primary input, a combinational logic cell, or a sequential cell (clock pin).

-to *to port_object or cell_object*

Specifies the timing path end point. A valid timing end point is a primary output, a combinational logic cell, or a sequential cell (data pin).

-through *port_object or cell_object*

Specifies the timing path pass through point. The timing path must go through this object.

**Examples** The following example sets a maximum delay by constraining all paths from ff1a:CLK to ff2e:D with a delay less than or equal to 5 ns:

```
set_max_delay -from [get_cells ff1a] -to [get_cells ff2e] 5.0

set_max_delay –from [get_pins ff1/Q] –through [get_pins and1/B]
–to [get_pins ff2/D] 7.0
```

**See Also** ▶"Synopsys Design Constraints" on page 416

## set_min_delay

Specifies the minimum delay for the timing paths.

**Syntax** set_min_delay [(-from)<*port_object or cell_object*>] [-through *port_object or cell_object*] [(-to) <*port_object or cell_object*>] <delay_value>

<object_list> = [ (get_ports | get_pins | get_nets | get_cells) <names>]

**Arguments** *delay_value*

Specifies a floating point number in nanoseconds that represents the required minimum delay value for specified paths.

If the path ending point is on a sequential device, the tool includes library hold time in the computed delay.

-from *from port_object or cell_object*

Specifies the timing path start point. A valid timing start point is a clock, a primary input, a combinational logic cell, or a sequential cell (clock pin).

-to *to port_object or cell_object*

Specifies the timing path end point. A valid timing end point is a primary output, a combinational logic cell, or a sequential cell (data pin).

-through *port_object or cell_object*

Specifies the timing path pass through point. The timing path must go through this object.

**Examples** The following example sets a minimum delay by constraining all paths from ff1a:CLK to ff2e:D with a delay greater than or equal to 5 ns:

```
set_min_delay -from [get_cells ff1a] -to [get_cells ff2e] 5.0
set_min_delay –from [get_pins ff1/Q] –through [get_pins and1/B]
–to [get_pins ff2/D] 7.0
```

**See Also** ▶"Synopsys Design Constraints" on page 416

## set_multicycle_path

Defines a path that takes multiple clock cycles.

**Syntax** set_multicycle_path *ncycles* [-from *from net_object or cell_object*] [-to *to net_object or cell_object*]

set_multicycle_path <value> [(-from | rise_from | fall_from)<object_list>] [-through object_list] [(-to | -rise_to | -fall_to) <object_list>] <delay_value>

<object_list> = [ (get_clocks | get_ports | get_pins | get_nets | get_cells) <names>]

**Arguments** *ncycles*

Specifies a value that represents the number of cycles the data path must have for setup check. The value is relative to the ending point clock and is defined as the delay required for arrival at the ending point.

-from *from object_list*

Specifies the timing path start point. A valid timing start point is a sequential cell (clock pin) or a clock net (signal). You can also use the keyword "all_registers" to include all registers' clock inputs.

-to *to object_list*

Specifies the timing path end point. A valid timing end point is a sequential cell (data-pin) or a clock-net (signal). You can also use the keyword "all_registers" to include all registers' data inputs.

-through *object_list*

Specifies the timing path pass through point. The timing path must go through this object.

**Example** The following example sets all paths between reg1 and reg2 to 3 cycles for setup check. Hold check is measured at the previous edge of the clock at reg2.

```
set_multicycle_path 3 –from [get_cells reg1] –to [get_cells
reg2]
set_multicycle_path 3 –from [get_pins ff1/Q] –through [get_pins
and1/B] –to [get_pins ff2/D
```

**See Also** ▶"Synopsys Design Constraints" on page 416

## set_output_delay

Defines the output delay of an output relative to a clock.

**Syntax** set_output_delay *delay_value* [-max |-min] -clock *clock_object output_port_object*

set_output_delay (-clock | -clock_fall) <clock_name> [-min | -max] [-add_delay] <delay_value> <port list>

<port_list> = [get_ports <names>] | [all_outputs]

**Arguments** *delay_value*

Specifies the amount of time from a reference clock to a primary output port.

-max

Specifies that the delay value is the maximum delay.

-min

Specifies that the delay value is the minimum delay.

-clock | clock_fall *clock_object*

Specifies the clock reference to which the specified input delay is related. clock_fall is on the falling edge of clock and is a mandatory argument.

*output_port_object*

Provides one or more (by wildcard) output ports in the current design to which delay_value is assigned. Use the keyword "all_outputs" to include all output ports.

*add_delay*

Used to define more than one output delay on a given port.

**Example** The following example sets an output delay of 1.2 ns for all outputs relative to clki_c:

```
set_output_delay 1.2 -clock [get_clocks CLK1] [get_ports OUT1]
set_output_delay 1.2 -clock [get_clocks CLK1] [all_outputs]
```

**Example** The following example sets an output delay of 1.2 ns minimum and 1.5 ns maximum for port data1 relative to the rising edge of CLK1:

```
set_output_delay 1.2 -min -clock [get_clocks CLK1] [get_ports
data1]
set_output delay 1.5 -max -clock [get_clocks CLK1] [get_ports
data1]
```

**See Also** ▶"Synopsys Design Constraints" on page 416

# Lattice Synthesis Engine-Supported HDL Attributes

This section describes the Synplify Lattice Attributes that are supported by the LSE. These attributes are directly interpreted by the engine and influence the optimization or structure of the output netlist.

All HDL attributes have priority over Strategy settings.

**See Also** ▶"black_box_pad_pin" on page 432


- "full_case" on page 434
- "loc" on page 435
- "parallel_case" on page 436
- "syn_black_box" on page 438
- "syn_encoding" on page 439
- "syn_force_pads" on page 444
- "syn_hier" on page 446
- "syn_insert_pad" on page 447
- "syn_keep" on page 449
- "syn_maxfan" on page 451
- "syn_multstyle" on page 452
- "syn_noprune" on page 454
- "syn_pipeline" on page 456
- "syn_preserve" on page 458
- "syn_ramstyle" on page 460

## black_box_pad_pin

This attribute specifies pins on a user-defined black box module. The pins are defined as I/O pads that are visible outside of the black box. If there is more than one port that is an I/O pad, the ports are listed inside double-quotes (") separated by commas (,), and without enclosed spaces. This attribute must be used in conjunction with the syn_black_box attribute.

**Verilog Syntax** object /* synthesis syn_black_box black_box_pad_pin = "portList" */ ;

where *object* is a module declaration, and *portList* is a space free, comma-separated list of the black box port names that are I/O pads.

**Verilog Example**

```
module black_box_pad_pin2(
                    input[4:0] in1,
                    input[4:0] in2,
                    input clk,
                    output[4:0] q
                    )/*          synthesis          syn_black_box
                    black_box_pad_pin="in1(4:0),q" */;

  reg [4:0] q;
  always @(posedge clk)
    begin
       q <= in1 + in2;
    end
endmodule

module black_box_pad_pin_instan(
  input[4:0] in1,
  input[4:0] in2,
  input[4:0] in3,
  input clk,
  output[5:0] q_out
  );

  wire [4:0] q;
  reg [5:0] q_out;
  black_box_pad_pin2 test_123(
                           .in1(in1),
                           .in2(in2),
                           .clk(clk),
                           .q(q)
                           );

  always @(posedge clk)
    begin
       q_out <= q + in3;
    end
endmodule
```

**VHDL Syntax** attribute black_box_pad_pin of object : architecture is "portList" ;

The object is the architecture name of a black box, while the data type is string. The portList is a space free, comma-separated list of the black box port names that are I/O pads.

**VHDL Example**

```
entity BBDLHS is
  port (D: in std_logic;
    E: in std_logic;
    GIN : in std_logic_vector(2 downto 0);
    Q : out std_logic );
end;

architecture BBDLHS_behav of BBDLHS is
end bl_box_behav;
attribute syn_black_box : boolean;
attribute syn_black_box of BBDLHS_behav : architecture is true;
attribute black_box_pad_pin : string;
attribute black_box_pad_pin of BBDLHS_behav : architecture is
"GIN(2:0),Q";
```

## full_case

Directive. For Verilog designs only. When used with a case, casex, or casez statement, this directive indicates that all possible values have been given, and that no additional hardware is needed to preserve signal values.

**Verilog Syntax** object /* synthesis full_case */

**Verilog Example**

```
module full_case1 (q, in1, in2, in3, in4, sel);
output q;
input in1, in2, in3, in4;
input [3:0] sel;
reg q;
always @(sel or in1 or in2 or in3 or in4)
begin
casez (sel) /* synthesis full_case */
4'b11??: q = in4;
4'b?1??: q = in3;
4'b???1: q = in1;
4'b??1?: q = in2;
default: q = 'bx;
endcase
end
endmodule
```

## loc

The loc attribute specifies pin locations for Lattice I/Os, instances, and registers, and forward-annotates them to the place-and-route tool. Refer to the Lattice databook for valid pin location values. If the attribute is on a bus, the software writes out bit-blasted constraints for forward-annotation.

**Verilog Syntax** object /* synthesis loc = "pinLocations" */ ;

In the syntax, pin_locations is a space free, comma-separated list of pin locations.

**Verilog Example**

```
I/O pin location
input [3:0]DATA0 /* synthesis loc="p10,p12,p11,p15" *;
Register pin location
reg data_in_ch1_buf_reg3 /* synthesis loc="R40C47" */;
Vectored internal bus
reg [3:0] data_in_ch1_reg /*synthesis loc =
"R40C47,R40C46,R40C45,R40C44" */;
```

**VHDL Syntax** attribute loc of object : objectType is "pinLocations" ;

In the syntax, pinLocations is a space free, comma-separated list of pin locations.

**VHDL Example**

```
entity mycomp is port(DATA0 : in std_logic_vector (3 downto 0);
                .
                .
                .
                );
attribute loc : string;
attribute loc of DATA0 : signal is "p10,p12,p11,p15";
```

## parallel_case

Directive. For Verilog designs only. Forces a parallel-multiplexed structure rather than a priority-encoded structure. This is useful because case statements are defined to work in priority order, executing only the first statement with a tag that matches the select value.

If the select bus is driven from outside the current module, the current module has no information about the legal values of select, and the software must create a chain of disabling logic so that a match on a statement tag disables all following statements. However, if you know the legal values of select, you can eliminate extra priority-encoding logic with the parallel_case directive.

In the following example, the only legal values of select are 4'b1000, 4'b0100, 4'b0010, and 4'b0001, and only one of the tags can be matched at a time. Specify the parallel_case directive so that tag-matching logic can be parallel and independent, instead of chained.

**Note**

Designers should be aware that it is possible for the priority of overlapping cases in post-synthesis simulation to mismatch with the priority behavior in RTL simulation when using this pragma.

**Verilog Syntax** You specify the directive as a comment immediately following the select value of the case statement.

object /* synthesis parallel_case */

where *object* is a case, casex or casez statement declaration.

### Verilog Example

```
module parallel_case1 (q, in1, in2, in3, in4, sel);
output q;
input in1, in2, in3, in4;
input [3:0] sel;
reg q;
always @(sel or in1 or in2 or in3 or in4)
begin
casez (sel) /* synthesis parallel_case */
4'b11??: q = in4;
4'b?1??: q = in3;
4'b???1: q = in1;
4'b??1?: q = in2;
default: q = 'bx;
endcase
end
endmodule
```

If the select bus is decoded within the same module as the case statement, the parallelism of the tag matching is determined automatically, and the parallel_case directive is unnecessary.

## syn_black_box

This attribute specifies that a Verilog module or VHDL architecture declaration is for a black box. Only the module's interface is defined for synthesis. The contents of a black box cannot be optimized during synthesis. A module can be a black box whether it is empty or not. However, the syn_black_box attribute cannot be used with the top-level module or architecture of a design. Additionally, the syn_black_box attribute is not supported for instances in Verilog or components in VHDL.

This attribute has an implicit Boolean value of 1 or true.

If any of the ports are I/O pads, add the black_box_pad_pin attribute. See "black_box_pad_pin" on page 432.

**Verilog Syntax** object /* synthesis syn_black_box */ ;

where *object* is a module declaration.

**Verilog Example**

```
module bl_box(out,data,clk) /* synthesis syn_black_box */;
```

**VHDL Syntax** attribute syn_black_box of object : architecture is true ;

where *object* is an architecture name. Data type is Boolean.

**VHDL Example**

```
entity bl_box is
  port (data : in std_logic_vector (7 downto 0);
    clk : in std_logic;
    out : out std_logic);
end;

architecture bl_box_behav of bl_box is
end bl_box_behav;
attribute syn_black_box : boolean;
attribute syn_black_box of bl_box_behav : architecture is true;
```

## syn_encoding

This attribute specifies the encoding style for a finite state machine (FSM), overriding the default LSE encoding. The default encoding is based on the number of states in the FSM. This attribute takes effect only when LSE infers an FSM. This attribute has no effect when syn_state_machine is 0, which blocks inference of an FSM.

Values for syn_encoding are as follows:

- Sequential – More than one bit of the state register can change at a time, but because more than one bit can be hot, the value must be decoded to determine the state. For example: 000, 001, 010, 011, 100
- Onehot – Only two bits of the state register change (one goes to 0; one goes to 1) and only one of the state registers is hot (driven by a 1) at a time. For example: 0000, 0001, 0010, 0100, 1000
- Gray – Only one bit of the state register changes at a time, but because more than one bit can be hot, the value must be decoded to determine the state. For example: 000, 001, 011, 010, 110

  There can be no more than four states for gray encoding. If the FSM has more than four states, LSE switches to sequential encoding.
- Safe – If the state machine enters an invalid state, additional logic will drive the state machine into its reset state. The design must have a defined reset state.

  Safe encoding can be combined with either sequential or onehot encoding (not with gray encoding) as in:

  ```
  syn_encoding = "safe,onehot"
  ```

  If the safe value is given by itself, it combines with the encoding method of a preceding syn_encoding statement or the default method.

**Verilog Syntax** Object /* synthesis syn_encoding = "value" */;

Where *object* is an enumerated type and value is from the list above.

**Verilog Example**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity syn_state_machine2 is
  port(
    clk : in std_logic;
    reset: in std_logic;
    en   : in std_logic;
    q  : out std_logic_vector(1 downto 0)
    );

end entity;

architecture behave of syn_state_machine2 is
  type state_type is(state0,state1,state2,state3);
  signal state,next_state:state_type;
  attribute syn_state_machine : boolean;
  attribute syn_state_machine of behave : architecture is true;
  attribute syn_encoding : string;
  attribute syn_encoding of state,next_state: signal is
"binary";
begin
  process(clk,reset)
  begin
    if reset = '1' then
      state <= state0;
    elsif clk'event and clk = '1' then
      state <= next_state;
    end if;
  end process;
  process(state)
  begin
    case state is
      when state0 =>
        if (en = '1') then
           q <= "00";
        end if;
        next_state <= state1;
      when state1=>
        if (en = '1') then
           q <= "01";
        end if;
        next_state <= state2;
      when state2 =>
        if (en = '1') then
           q <= "10";
        end if;
        next_state <= state3;
      when state3 =>
        if (en = '1') then
           q <= "11";
        end if;
        next_state <= state0;
    end case;
  end process;
end behave
```

**VHDL Syntax** attribute syn_encoding of object: objectType is "value";

Where *object* is an enumerated type and value is from the list above.

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity syn_encoding1 is
  port(
    clk : in std_logic;
    reset: in std_logic;
    en   : in std_logic;
    q  : out std_logic_vector(1 downto 0)
    );
end entity;

architecture behave of syn_encoding1 is
  signal state : std_logic_vector(3 downto 0);
  constant state0 : std_logic_vector(3 downto 0) := "1000";
  constant state1 : std_logic_vector(3 downto 0) := "0100";
  constant state2 : std_logic_vector(3 downto 0) := "0010";
  constant state3 : std_logic_vector(3 downto 0) := "0001";
  attribute syn_encoding : string;
  attribute syn_encoding of state : signal is "safe,onehot";
begin
  process(clk,reset,en)
  begin
    if reset = '1' then
      state <= state0;
      q <= "00";
    elsif clk'event and clk = '1' then
      case state is
        when state0 =>
          if (en = '1') then
             q <= "00";
          end if;
          state <= state1;
        when state1=>
          if (en = '1') then
             q <= "01";
          end if;
          state <= state2;
        when state2 =>
          if (en = '1') then
             q <= "10";
          end if;
          state <= state3;
        when state3 =>
          if (en = '1') then
             q <= "11";
          end if;
          state <= state0;
        when others => null;
      end case;
    end if;
  end process;
end behave
```

## syn_force_pads

This attribute prevents unused ports from being optimized away to allow I/O pad insertion on the unused port. This attribute is not supported at the global level. Instead, set the use_io_insertion option to control I/O insertion globally.

This attribute is supported in the rtl, and it will override the global use_io_insertion option setting on the given input, output, or bidir port.

For example, in the following Verilog file, the syn_force_pads attribute can be set to 1 on an unused input port (dataz), and it will not be optimized away, regardless of the use_io_insertion global setting.

**Verilog Syntax** object /* synthesis syn_force_pads = {1 | 0} */;

where *object* is port declaration.

**Verilog Example**

```
`define DSIZE 9
`define OSIZE 18

module multp9x9(dataout, dataax, dataay, dataz, clk, rst, ce);
   output [`OSIZE-1:0] dataout;
   input [`DSIZE:0] dataz /* synthesis syn_force_pads = 1*/;
   input [`DSIZE-1:0] dataax, dataay;
   input clk, rst, ce;
   reg  [`DSIZE-1:0] dataax_reg, dataay_reg;

   reg [`OSIZE-1:0] dataout;
   wire [`OSIZE-1:0] dataout_tmp ;
   assign dataout_tmp = dataax_reg * dataay_reg;

   always @(posedge clk or posedge rst)
   begin
     if (rst)
     begin
         dataax_reg <= 0;
         dataay_reg <= 0;
         dataout <= 0;
     end
     else if (ce == 1'b1)
     begin
         dataax_reg <= dataax;
         dataay_reg <= dataay;
         dataout <= dataout_tmp;
     end
   end
endmodule
```

To force I/O pads to be inserted for input ports that do not drive logic, follow the guidelines below.

- To force I/O pad insertion on an individual port, set the syn_force_pads attribute on the port with a value to 1. To disable I/O insertion for a port, set the attribute on the port with a value of 0.

Enable this attribute to preserve user-instantiated pads, insert pads on unconnected ports, insert bi-directional pads on bi-directional ports instead of converting them to input ports, or insert output pads on unconnected outputs.

If you do not set the syn_force_pads attribute, the synthesis design optimizes any unconnected I/O buffers away.

**VHDL Syntax** Attribute syn_force_pads of object: objectType is "true | false" ;

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity multp9x9 is
  port ( dataout : out std_logic_vector(17 downto 0);
      dataax, dataay: in std_logic_vector( 8 downto 0);
           dataz : in std_logic_vector(8 downto 0);
           clk,rst,ce: in std_logic
    );
  attribute syn_force_pads : string;
  attribute syn_force_pads of dataz : signal is "true";

end multp9x9;

architecture rtl of multp9x9 is

signal dataax_reg,dataay_reg: std_logic_vector(8 downto 0);
signal dataout_tmp: std_logic_vector(17 downto 0);

begin

  dataout_tmp <= dataax_reg * dataay_reg;
  process (clk, rst)
  begin
      if rst = '1' then
           dataax_reg <= (others => '0');
           dataay_reg <= (others => '0');
           dataout <= (others => '0');
      elsif clk'event and clk = '1' then
        if ce = '1' then
           dataax_reg <= dataax;
           dataay_reg <= dataay;
           dataout <= dataout_tmp;
      end if;
      end if;
  end process;

end rtl;
```

## syn_hier

This attribute allows you to control the amount of hierarchical transformation that occurs across boundaries on module or component instances during optimization. This attribute cannot be applied globally. The user must set this attribute on the selective modules to stop cross-boundary optimizations.

**syn_hier Values** The following value can be used for syn_hier:

- Hard – Preserves the interface of the design unit with no exceptions. This attribute affects only the specified design units.

**Verilog Syntax** object /* synthesis syn_hier = "value" */ ;

where *object* can be a module declaration and *value* can be any of the values described in syn_hier Values. Check the attribute values to determine where to attach the attribute.

**Verilog Example**

```
module top1 (Q, CLK, RST, LD, CE, D)
   /* synthesis syn_hier = "hard" */;
```

**VHDL Syntax** attribute syn_hier of object : architecture is "value" ;

where *object* is an architecture name and *value* can be any of the values described in syn_hier Values. Check the attribute values to determine the level at which to attach the attribute.

**VHDL Example**

```
architecture struct of cpu is
attribute syn_hier : string;
attribute syn_hier of struct: architecture is "hard";
```

## syn_insert_pad

This attribute removes an existing I/O buffer from a port or net when I/O buffer insertion is enabled.

The syn_insert_pad attribute is used when the use_io_insertion global option is enabled (when I/O buffers are automatically inserted) to allow users to selectively remove an individual buffer from a port or net.

It can also be used to force an I/O buffer to be inserted on a specific port or net, if the use_io_insertion global option is disabled.

- Setting the attribute to 0 on a port or net removes the I/O buffer (or prevents an I/O buffer from being automatically inserted, if the use_io_insertion global option is enabled).
- Setting the attribute to 1 on a port or net forces an I/O buffer to be inserted if the use_io_insertion global option is disabled.

**Verilog Syntax** object /* synthesis syn_insert_pad = {1 | 0} */;

where *object* is a port or net declaration.

**Verilog Example**

```
`define OSIZE 16
`define DSIZE 8

module mac8x8 (dataout, x, y, clk, rst);
 output [`OSIZE:0] dataout;
 input [`DSIZE-1:0] x, y;
 input clk;
 input rst /* synthesis syn_insert_pad = 0 */;
 reg  [`OSIZE:0] dataout;
 reg [`DSIZE-1:0] x_reg, y_reg;
 wire [`OSIZE-1:0] multout ;
 wire [`OSIZE:0] sum_out;

 assign multout = x_reg * y_reg;
 assign sum_out = multout + dataout;

 always @(posedge clk or posedge rst)
 begin
   if (rst)
   begin
      x_reg = 0;
      y_reg = 0;
      dataout = 0;
   end
   else
   begin
      x_reg = x;
      y_reg = y;
      dataout = sum_out;
   end
 end
endmodule
```

In the previous example, the input port labeled “rst” will not have an input buffer connected to it in the technology-mapped netlist after LSE is completed.

**VHDL Syntax** attribute syn_insert_pad of object : objectType is "true | false" ;

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;

entity register_en_reset is
          generic (
                    width : integer := 8
                     );
  port (
    datain  : in std_logic_vector(width-1 downto 0);
    clk     : in std_logic;
    enable  : in std_logic;
    reset   : in std_logic;
    dataout : out std_logic_vector(width-1 downto 0)
    );
  attribute syn_insert_pad : string;
  attribute syn_insert_pad of reset : signal is "false";


end register_en_reset;

architecture lattice_behav of register_en_reset is

begin
    process (clk,reset)
    begin
      if (reset = '1') then
                      dataout <= (others => '0');
      elsif (rising_edge(clk) and enable = '1') then
        dataout <= datain;
      end if;
    end process;
end lattice_behav;
```

## syn_keep

This attribute keeps the specified net intact during optimization and synthesis.

**Verilog Syntax** object /* synthesis syn_keep = 1 */ ;

where *object* is a wire or reg declaration. Make sure that there is a space between the object name and the beginning of the comment slash (/).

**Verilog Example**

```
module syn_keep1(
        input a,
        input b,
        input clk,
        output q1,
        output q2);
  reg temp1;
  reg temp2;
  reg q1;
  reg q2;
  wire or_result;
  wire keep1/* synthesis syn_keep=1 */;
  wire keep2/* synthesis syn_keep=1 */;

  always @(posedge clk)
    begin
      temp1 = a;
      temp2 = b;
    end
  assign or_result = (temp1 | temp2);
  assign keep1 = or_result;
  assign keep2 = or_result;
  always@(posedge clk)
    begin
      q1 = keep1;
      q2 = keep2;

    end
endmodule
```

**VHDL Syntax** attribute syn_keep of object : objectType is true ;

where *object* is a single or multiple-bit signal.

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;

entity syn_keep1 is
  port(
    a : in std_logic;
    b : in std_logic;
    clk: in std_logic;
    q1: out std_logic;
    q2: out std_logic
    );
end entity;

architecture behave of syn_keep1 is
  signal temp1 : std_logic;
  signal temp2 : std_logic;
  signal keep1 : std_logic;
  signal keep2 : std_logic;
  signal or_result : std_logic;
  attribute syn_keep: boolean;
  attribute syn_keep of keep1,keep2: signal is true;
begin
  process(clk)
  begin
    if clk'event and clk = '1' then
      temp1 <= a;
      temp2 <= b;
    end if;
  end process;

  or_result <= (temp1 or temp2);
  keep1 <= or_result;
  keep2 <= or_result;

  process(clk)
  begin
    if clk'event and clk = '1' then
      q1 <= keep1;
      q2 <= keep2;
    end if;
  end process;

end behave;
```

## syn_maxfan

This attribute overrides the default (global) fan-out guide for an individual input port, net, or register output.

**Verilog Syntax** object /* synthesis syn_maxfan = "value" */ ;

**Note**

LSE will take integer values for non-integral values to syn_maxfan attribute.

For example, syn_maxfan value of 5.1 will be truncated to 5.

**Verilog Example**

```
module test (registered_data_out, clock, data_in);
output [31:0] registered_data_out;
input clock;
input [31:0] data_in /* synthesis syn_maxfan=1000 */;
reg [31:0] registered_data_out /* synthesis syn_maxfan=1000 */;
```

**VHDL Syntax** attribute syn_maxfan of object : objectType is "value" ;

**VHDL Example**

```
entity test is
   port (clock : in bit;
         data_in : in bit_vector(31 downto 0);
         registered_data_out: out bit_vector(31 downto 0) );
attribute syn_maxfan : integer;
attribute syn_maxfan of data_in : signal is 1000;
```

## syn_multstyle

This attribute specifies whether the multipliers are implemented as dedicated hardware blocks or as logic.

**syn_multstyle Values** block_mult | logic

**Value Description Default** block_mult Implements the multipliers as dedicated hardware blocks.

This attribute only applies to families that use DSP blocks on the device. To override this behavior, specify a value of logic.

**Verilog Syntax** input net /* synthesis syn_multstyle = "block_mult | logic" */;

**Verilog Example**

```
module syn_multstyle1(
            input [7:0] in1,
            input [7:0] in2,
            input rst,
            input clk,
            output [15:0] result);

  reg [7:0] temp1,temp2;
  reg [15:0] result;
  wire [15:0] product /*synthesis syn_multstyle = "logic"*/;

  always@(posedge clk ,negedge rst)
    begin
      if(!rst)
        begin
          temp1 = 'b0;
          temp2 = 'b0;
        end
      else
        begin
          temp1  = in1;
          temp2  = in2;
        end
    end

  assign product = temp1*temp2;

  always@(posedge clk, negedge rst)
    begin
      if(!rst)
        begin
          result  = 'b0;
        end
      else
        begin
          result = product;
        end
      end
endmodule
```

**VHDL Syntax** attribute syn_multstyle of instance : signal is "block_mult | logic";

**VHDL Example**

```
library ieee ;
use ieee.std_logic_1164.all ;
USE ieee.numeric_std.all;

entity mult is
port (clk : in std_logic ;
 a : in std_logic_vector(7 downto 0) ;
 b : in std_logic_vector(7 downto 0) ;
 c : out std_logic_vector(15 downto 0))
end mult ;

architecture rtl of mult is
signal mult_i : std_logic_vector(15 downto 0) ;
attribute syn_multstyle : string ;
attribute syn_multstyle of mult_i : signal is "logic" ;

begin
mult_i <= std_logic_vector(unsigned(a)*unsigned(b)) ;
iCEcube2 User Guide www.latticesemi.com 88
process(clk)
begin
 if (clk'event and clk = '1') then
 c <= mult_i ;
end if ;
end process
```

## syn_noprune

This attribute prevents instance optimization for black-box modules (including technology-specific primitives) with unused output ports. This attribute is not a global attribute and works on the component basis. The user must set the attribute on the instance.

**Verilog Syntax** object /* synthesis syn_noprune = 1 */ ;

where *object* is a module instance. The data type is Boolean.

**Verilog Example**

```
module top(a1,b1,c1,d1,y1,clk);
output y1;
input a1,b1,c1,d1;
input clk;
wire x2,y2;
reg y1;
syn_noprune u1(a1,b1,c1,d1,x2,y2) /* synthesis syn_noprune=1 */
;

always @(posedge clk)
   y1<= a1;

endmodule
```

**VHDL Syntax** attribute syn_noprune of object : objectType is true ;

where the *data type* is boolean, and *object* is a component.

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;
entity top is
   port (a1, b1 : in std_logic;
         c1,d1,clk : in std_logic;
         y1 :out std_logic );
end ;
architecture behave of top is
component noprune
port (a, b, c, d : in std_logic;
      x,y : out std_logic );
end component;
signal x2,y2 : std_logic;
attribute syn_noprune : boolean;
attribute syn_noprune of noprune : component is true;
begin
   u1: noprune port map(a1, b1, c1, d1, x2, y2);
   process begin
      wait until (clk = '1') and clk'event;
      y1 <= a1;
   end process;
end;
```

## syn_pipeline

This attribute permits registers to be moved to improve timing, and specifies that registers that are outputs of Multipliers/Adders can be moved to improve timing. Depending on the criticality of the path, the tool moves the output register to the input side.

**Verilog Syntax** object /* synthesis syn_pipeline = {1 | 0} */ ;

where *object* is a register declaration.

- The value of 0 (or false) indicates pipelining for the specified register is disabled, which means the register position in the design is fixed.
- The value of 1 (or true) indicates pipelining for the specified register is allowed, which means the register may be moved if it helps improve timing.

LSE identifies registers that are candidates for possible pipelining based on running RTL timing analysis. It may identify some candidate registers, or it may determine there are none that are suitable.

If LSE decides no candidate registers for pipelining exist, and if the user sets the syn_pipeline attribute to “1” on a specific register in the RTL to force pipelining for that register, that attribute will not be honored.

If global pipelining is enabled for a design, and LSE has identified one or more registers as possible candidates for pipelining, the user may prevent these registers from being pipelined by setting synthesis attribute syn_pipeline=0 for each of those registers in the RTL.

**Verilog Example**

```
module pipeline (a,b,c,d,clk,out);

input [3:0] a,b,c,d;
input clk;
output[7:0]out;

reg[7:0]out,out1 /* synthesis syn_pipeline = 0 */;
reg[3:0] a_temp,b_temp,c_temp,d_temp;

always @(posedge clk)
begin
   a_temp <= a;
   b_temp <= b;
   c_temp <= c;
   d_temp <= d;
   out1 <= (a_temp * b_temp) +(c_temp * d_temp);
   out <= out1;
end
endmodule
```

In the previous example, the registers labeled “out1” will not be moved to the input side of the adder to improve timing.

**VHDL Syntax** attribute syn_pipeline of object : objectType is {true|false} ;

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;

entity syn_pipeline_exp is
port (CLK_0 : in std_logic;
    A_IN : in std_logic_vector(3 downto 0);
    B_IN : in std_logic_vector(3 downto 0);
    RST : out std_logic_vector(7 downto 0)
    );
end syn_pipeline_exp;

architecture rtl of syn_pipeline_exp is
signal A_REGSTR : std_logic_vector(3 downto 0);
signal B_REGSTR : std_logic_vector(3 downto 0);
signal TMP : std_logic_vector(7 downto 0);
signal TMP1 : std_logic_vector(7 downto 0);
signal TMP2 : std_logic_vector(7 downto 0);
attribute syn_pipeline : string;
attribute syn_pipeline of TMP1 : signal is "true";

begin
  process(CLK_0)
  begin
    if (CLK_0'event and CLK_0 = '1') then
      TMP <= A_REGSTR * B_REGSTR;
      A_REGSTR <= A_IN;
      B_REGSTR <= B_IN;
      TMP1 <= TMP;
      TMP2 <= TMP1;
      RST <= TMP2;
    end if;
  end process;

end rtl;
```

## syn_preserve

This attribute prevents sequential optimizations such as constant propagation and inverter push-through from removing the specified register. The syn_encoding attribute is not honored if there is a syn_preserve attribute on any of the state machine registers.

**Verilog Syntax** object /* synthesis syn_preserve = 1 */ ;

where *object* is a register definition signal or a module.

**Verilog Example**

```
module syn_preserve1(
          input[3:0]in1,
          input[3:0]in2,
          input[3:0]in3,
          input clk,
          output [7:0] result,
          output [3:0] sum
          )/* synthesis syn_preserve= 1*/;

reg [7:0] result/*synthesis syn_multstyle = "EBR"*/;
reg [3:0] temp1,temp2,temp3;
reg [3:0] sum;

always@(posedge clk)
  begin
    temp1 = in1 & in2;
    temp2 = !temp1;
    temp3 = temp1 & temp2;
    result = temp3*in3;
    sum = temp3 + in3;
  end
endmodule
```

**VHDL Syntax** attribute syn_preserve of object : objectType is true ;

where *object* is an output port or an internal signal that holds the value of a state register or architecture.

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity syn_preserve2 is
  port(
     in1: in std_logic_vector(3 downto 0);
     in2: in std_logic_vector(3 downto 0);
     in3: in std_logic_vector(3 downto 0);
     clk: in std_logic;
     result: out std_logic_vector(7 downto 0);
     sum   : out std_logic_vector(3 downto 0)
    );
end entity;

architecture behave of syn_preserve2 is
  signal temp1,temp2,temp3 : std_logic_vector(3 downto 0);
  attribute syn_preserve : boolean;
  attribute syn_preserve of behave: architecture is true;
  attribute syn_multstyle : string;
  attribute syn_multstyle of result: signal is "EBR";
begin
  process(clk)
  begin
    if clk'event and clk = '1' then
      temp1 <= in1 and in2;
      temp2 <= not temp1;
      temp3 <= temp1 and temp2;
      result <= temp3*in3;
      sum <= temp3 + in3;
    end if;
  end process;
end behave;
```

## syn_ramstyle

The syn_ramstyle attribute specifies the implementation to use for an inferred RAM. You apply syn_ramstyle globally to a module or RAM instance. To turn off RAM inference, set its value to registers.

The following values can be specified globally or on a module or RAM instance:

- Registers – Causes an inferred RAM to be mapped to registers (flip-flops and logic) rather than the technology-specific RAM resources.
- Distributed – Causes the RAM to be implemented using the distributed RAM or PFU resources.
- Block_ram – Causes the RAM to be implemented using the dedicated RAM resources. If your RAM resources are limited, you can use this attribute to map additional RAMs to registers instead of the dedicated or distributed RAM resources.
- no_rw_check (some modes, but all technologies). – You cannot specify this value alone. Without no_rw_check, the synthesis tool inserts bypass logic around the RAM to prevent the mismatch. If you know your design does not read and write to the same address simultaneously, use no_rw_check to eliminate bypass logic. Use this value only when you cannot simultaneously read and write to the same RAM location and you want to minimize overhead logic.

**Verilog Syntax** object /* synthesis syn_ramstyle = "string" */ ;

where *object* is a register definition (reg) signal. The data type is string.

**Verilog Example**

```
module ram4 (datain,dataout,clk);
output [31:0] dataout;
input clk;
input [31:0] datain;
reg [7:0] dataout[31:0] /* synthesis syn_ramstyle="block_ram" */;
```

**VHDL Syntax** attribute syn_ramstyle of object : objectType is "string" ;

where *object* is a signal that defines a RAM or a label of a component instance. *Data type* is string.

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;
entity ram4 is
   port (d : in std_logic_vector(7 downto 0);
         addr : in std_logic_vector(2 downto 0);
         we : in std_logic;
         clk : in std_logic;
         ram_out : out std_logic_vector(7 downto 0) );
end ram4;
library synplify;
architecture rtl of ram4 is
type mem_type is array (127 downto 0) of std_logic_vector (7
downto 0);
signal mem : mem_type; -- mem is the signal that defines the
RAM
attribute syn_ramstyle : string;
attribute syn_ramstyle of mem : signal is "block_ram";
```

## syn_replicate

This attribute controls replication. While the synthesis tool can automatically replicate registers during optimization, this attribute disables replication either globally or on a per register basis.

**Verilog Syntax** object /* synthesis syn_replicate = 1 | 0 */;

**Verilog Example** For example:

```
module syn_replicate1 (en1,en2,clk,in1,in2,q);
  input en1,en2;
  input clk;
  input [6:0]in1,in2;
  output [6:0]q;
  reg [6:0]q;
  reg enc /* synthesis syn_maxfan = 1 syn_replicate = 1*/;

  always @(posedge clk)
    begin
      enc = en1 & en2;
    end

  always @(posedge clk)
    begin
      if (enc)
        q = in1;

      else
        q = in2;
    end
endmodule
```

**VHDL Syntax** attribute syn_replicate of object : objectType is true | false ;

**VHDL Example** For example:

```
library ieee;
use ieee.std_logic_1164.all;

entity syn_replicate2 is
  port(
     en1: in std_logic;
     en2: in std_logic;
     clk: in std_logic;
     in1: in std_logic_vector(6 downto 0);
     in2: in std_logic_vector(6 downto 0);
     q:  out std_logic_vector(6 downto 0)
    );
end entity;

architecture behave of syn_replicate2 is
  signal enc : std_logic;
  attribute syn_maxfan: integer;
  attribute syn_maxfan of behave : architecture is 1;
  attribute syn_replicate: boolean;
  attribute syn_replicate of enc : signal is false;
begin
  process(clk)
  begin
    if clk'event and clk = '1' then
      enc <= (en1 and en2);
    end if;
  end process;

  process(clk)
  begin
    if enc = '1' then
      q <= in1;
    else
      q <= in2;
    end if;
  end process;
end behave;
```

## syn_romstyle

This attribute allows you to implement ROM architectures using dedicated or distributed ROM. Infer ROM architectures uses a CASE statement in your code.

For the synthesis tool to implement a ROM, at least half of the available addresses in the CASE statement must be assigned a value. For example, consider a ROM with six address bits (64 unique addresses). The case statement for this ROM must specify values for at least 32 of the available addresses. You can apply the syn_romstyle attribute globally to the design by adding the attribute to the module or entity.

The following values can be specified globally on a module or ROM instance:

- Auto – Allows the synthesis tool to chose the best implementation to meet the design requirements for speed, size, etc.
- Logic – Causes the ROM to be implemented using the distributed ROM or PFU resources.
- EBR – Causes the ROM to be implemented using the dedicated ROM resources. If your ROM resources are limited, you can use this attribute to map additional ROM to registers instead of the dedicated or distributed RAM resources.

**Verilog Syntax** object /* syn_romstyle = "auto(default) | EBR | logic" */ ;

**Verilog Example**

```
reg [8:0] z /* synthesis syn_romstyle = "EBR" */;
```

**VHDL Syntax** attribute syn_romstyle of object : object_type is "auto(default) | EBR | logic" ;

**VHDL Example**

```
signal z : std_logic_vector(8 downto 0);
attribute syn_romstyle : string;
attribute syn_romstyle of z : signal is "logic";
```

## syn_srlstyle

This attribute determines how to implement the sequential shift components.

**Verilog Syntax** object /* synthesis syn_srlstyle = "string",

where *string* can take one of the following values:

- Registers: seqShift register components are implemented as registers.
- Distributed: seqShift register components are implemented as distributed RAM.
- Block_ram: seqShift register components are implemented as block RAM

If the attribute value set by the user cannot be honored (for example, the user sets the attribute value to "block_ram", but the selected device does not contain enough available EBR blocks to implement the shift register), LSE will display a message to indicate this.

" | registers | distributed | |block_ram" */ ;

In the above syntax, *object* is a register declaration.

**Verilog Example** The following example implements seqShift components as distributed memory with any required fabric logic.

```
module test_srl(clk, enable, dataIn, result, addr);
input clk, enable;
input [3:0] dataIn;
input [3:0] addr;
output [3:0] result;
reg [3:0] regBank[15:0]
  /* synthesis syn_srlstyle="distributed" */;
integer i;
always @(posedge clk) begin
  if (enable == 1) begin
    for (i=15; i>0; i=i-1) begin
      regBank[i] <= regBank[i-1];
    end
  regBank[0] <= dataIn;
  end
end
assign result = regBank[addr];
endmodule
```

The following example implements a seqShift for 16x256 bits wide and serial in and serial out register using syn_srlstyle set to block_ram.

**VHDL Syntax** attribute syn_srlstyle of object : signal is

" registers | distributed |block_ram " ;

In the above syntax, *object* is a register.

```
// shift left register with 16X256 bits width and serial in and
serial out
module test(clock, arst, sr_en, shiftin, shiftout);
parameter sh_len=16;
parameter sh_width=256;
parameter ARESET_VALUE = {(sh_width){1'b0}};
input clock,arst,sr_en;
input [sh_width-1:0] shiftin;
output [sh_width-1:0] shiftout;
integer i;
reg [sh_width-1:0] sreg [sh_len-1:0] /* synthesis
syn_srlstyle="block_ram" */;
  always @(posedge clock or posedge arst)
  begin
    if(arst)
      begin
        for(i = 0;i <= sh_len-1;i = i+1)
        sreg[i] <= ARESET_VALUE ;
      end
    else
  begin
    if(sr_en)
    begin
      sreg[0] <= shiftin;
      for(i=sh_len-1;i>0;i=i-1)
        sreg[i] <= sreg[i-1];
      end
    end
  end
assign shiftout = sreg[sh_len-1];
endmodule
```

**Verilog Example** The example below implements seqShift components as distributed memory primitives:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity d_p is
  port (clk : in std_logic;
    data_out : out std_logic_vector(127 downto 0));
  end d_p;

architecture rtl of d_p is
type dataAryType is array(3 downto 0) of
  std_logic_vector(127 downto 0);
signal h_data_pip_i : dataAryType;
attribute syn_srlstyle : string;
attribute syn_srlstyle of h_data_pip_i : signal
is "distributed";
begin
  process (Clk)
  begin
    if (Clk'Event And Clk = '1') then
      h_data_pip_i <= (h_data_pip_i(2 DOWNTO 0)) &
      h_data_pip_i(3);
    end if;
  end process;
data_out <= h_data_pip_i(0);
end rtl;
```

## syn_sharing

Directive. Enables or disables the sharing of operator resources during the compilation stage of synthesis.

The syn_sharing directive controls resource sharing during the compilation stage of synthesis. This is a compiler-specific optimization that does not affect the mapper, meaning that the mapper might still perform resource sharing optimizations to improve timing, even if syn_sharing is disabled.

If you disable resource sharing globally, you can use the syn_sharing directive to turn on resource sharing for specific modules or architectures.

**Verilog Syntax** object /* synthesis syn_sharing=”on | off” */ ;

**Verilog Example**

```
module syn_sharing1 (
          input [7:0] inA1,
          input [7:0] inA2,
          input [7:0] inB1,
          input [7:0] inB2,
          input clk,
          input sel1,
          input sel2,
          input rst,
          output [15:0] product1,
          output [15:0] product2
          )/*synthesis syn_sharing = 1*/;

  reg [15:0] product1,product2;
  wire [15:0] temp1,temp2;
  assign temp1 = inA1*inB1;
  assign temp2 = inA2*inB2;
  always@(posedge clk)
    begin
      if(sel1)
        begin
          if (sel2)
            product1 = temp1;
          else
            product1 = temp2;
        end
      else
        begin
          if (sel2)
            product2 = temp1;
          else
            product2 = temp2;
        end
    end
endmodule
```

**VHDL Syntax** attribute syn_sharing of object : objectType is “true | false” ;

### VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity syn_sharing2 is
  port(
    inA1 : in std_logic_vector(7 downto 0);
    inA2 : in std_logic_vector(7 downto 0);
    inB1 : in std_logic_vector(7 downto 0);
    inB2 : in std_logic_vector(7 downto 0);
    clk  : in std_logic;
    sel1 : in std_logic;
    sel2 : in std_logic;
    rst  : in std_logic;
    product1 : out std_logic_vector(15 downto 0);
    product2 : out std_logic_vector(15 downto 0)
    );
end entity;

architecture behave of syn_sharing2 is
 signal temp1,temp2: std_logic_vector(15 downto 0);
 attribute syn_sharing : boolean;
 attribute syn_sharing of behave : architecture is false;
begin
  temp1 <= inA1*inB1;
  temp2 <= inA2*inB2;
  process(clk)
  begin
    if clk'event and clk ='1' then
      if sel1 = '1' then
        if sel2 = '1' then
          product1 <= temp1;
        else
          product1 <= temp2;
        end if;
      else
        if sel2 = '1' then
          product2 <= temp1;
        else
          product2 <= temp2;
        end if;

      end if;
    end if;
  end process;
end behave;
```

## syn_state_machine

This attribute enables/disables state-machine optimization on individual state registers in the design. To extract some state machines, use this attribute with a value of 1 on just those individual state-registers to be extracted. If there are state machines in your design that you do not want extracted, use syn_state_machine with a value of 0 to override extraction on those individual state registers.

All state machines are usually detected during synthesis. However, on occasion there are cases in which certain state machines are not detected. You can use this attribute to declare those undetected registers as state machines.

The syn_sharing attribute only can be used in architecture. The syn_sharing attribute cannot be used in entity.

**Verilog Syntax** object /* synthesis syn_state_machine = 0 | 1 */ ;

where *object* is a state register. Data type is Boolean: 0 does not extract an FSM, 1 extracts an FSM.

**Verilog Example**

```
module syn_state_machine1 (clk, reset, en, q);
  input clk, reset, en;
  output[1:0]q;
  reg q;
  reg [3:0] state,next_state /* synthesis syn_state_machine = 0
*/;
  parameter state0 = 4'b1000;
  parameter state1 = 4'b0100;
  parameter state2 = 4'b0010;
  parameter state3 = 4'b0001;
  always @(posedge clk or posedge reset)
    begin
      if (reset)
          state <= state0;
      else
        state <= next_state;
    end

  always @(state)
    begin
      case (state)
        state0:
          begin
            if (en == 1)
              q <= 2'b00;
            next_state <= state1;
          end

        state1:
          begin
            if (en == 1)
              q <= 2'b01;
            next_state <= state2;
          end
        state2:
          begin
            if (en == 1)
              q <= 2'b10;
            next_state <= state3;
          end
        state3:
          begin
            if (en == 1)
              q <= 2'b11;
            next_state <= state0;
          end
      endcase
    end
endmodule
```

**VHDL Syntax** attribute syn_state_machine of object : objectType is true | false ;

where *object* is a signal that holds the value of the state machine.

**VHDL Example**

```
attribute syn_state_machine of current_state: signal is true;
```

The following is the source code used for the previous example.

```
library ieee;
use ieee.std_logic_1164.all;
entity syn_statemachine_exp is
port (CLK_0, RESET, IN1 : in std_logic;
                  OUT1 : out std_logic_vector (2 downto 0)
      );
end syn_statemachine_exp;

architecture behave of syn_statemachine_exp is
type ST_VALS is (STATE0, STATE1, STATE2, STATE3);
signal STATE, NXT_ST: ST_VALS;
attribute syn_state_machine : boolean;
attribute syn_state_machine of STATE : signal is true;

begin
  process (CLK_0, RESET)
  begin
    if RESET = '1' then
      STATE <= STATE0;
    elsif rising_edge(CLK_0) then
      STATE <= NXT_ST;
    end if;
  end process;

  process (STATE, IN1)
  begin
    case STATE is
      when STATE0 =>
        OUT1 <= "000";
        if IN1 = '1' then NXT_ST <= STATE1;
        else NXT_ST <= STATE0;
        end if;
      when STATE1 =>
        OUT1 <= "001";
        if IN1 = '1' then NXT_ST <= STATE2;
        else NXT_ST <= STATE1;
        end if;
      when STATE2 =>
        OUT1 <= "010";
        if IN1 = '1' then NXT_ST <= STATE3;
        else NXT_ST <= STATE2;
        end if;
      when others =>
        OUT1 <= "XXX"; NXT_ST <= STATE0;
    end case;
  end process;

end behave;
```

## syn_use_carry_chain

This attribute is used to turn on or off the carry chain implementation for adders.

**Verilog Syntax** object synthesis syn_use_carry_chain = {1| 0} */ ;

**Verilog Example** To use this attribute globally, apply it to the module.

```
module test (a, b, clk, rst, d) /* synthesis
syn_use_carry_chain = 1 */;
```

**VHDL Syntax** attribute syn_use_carry_chain of object : objectType is true | false ;

**VHDL Example**

```
architecture archtest of test is
signal temp : std_logic;
signal temp1 : std_logic;
signal temp2 : std_logic;
signal temp3 : std_logic;
attribute syn_use_carry_chain : boolean;
attribute syn_use_carry_chain of archtest : architecture is
true;
```

## syn_useenables

This attribute controls the use of clock enables on registers in the design. Exploiting clock enables on registers is usually beneficial. However, there are timing closure situations where clock enable routing causes timing violations. This is one reason why the user may want to stop the use of clock enable on the register.

**Verilog Syntax:** object /* synthesis syn_useenables = "0 | 1" */;

**Verilog Example**

```
module syn_useenable1 (Din1,Din2,en,clk,Dout);
  input [7:0] Din1, Din2;
  input clk,en;
  output [7:0] Dout;
  reg [7:0] temp1;
  reg [7:0] Dout /* synthesis syn_useenables = 0*/;
  always@(posedge clk)
    begin
      temp1 <= Din1 & Din2;
    end
  always @(posedge clk)
    begin
      if(en)
        Dout <= temp1;
    end
endmodule
```

**VHDL Syntax:** attribute syn_useenables of object : objectType is "true | false";

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;

entity syn_useenable2 is
  port(
     Din1 : in std_logic_vector(7 downto 0);
     Din2 : in std_logic_vector(7 downto 0);
     clk  : in std_logic;
     en   : in std_logic;
     Dout : out std_logic_vector(7 downto 0)
    );
end entity;

architecture behave of syn_useenable2 is
  signal temp1 : std_logic_vector(7 downto 0);
  attribute syn_useenables: boolean;
  attribute syn_useenables of Dout: signal is false;
begin
  process(clk)
  begin
    if clk'event and clk = '1' then
      temp1 <= Din1 and Din2;
    end if;
  end process;

  process(clk)
  begin
    if clk'event and clk = '1' then
      if en = '1' then
        Dout <= temp1;
      end if;
    end if;
  end process;
end behave;
```

## syn_useioff

This attribute overrides the default behavior to pack registers into I/O pad cells based on timing requirements for the target Lattice families. Attribute syn_useioff is Boolean-valued: 1 enables (default) and 0 disables register packing. You can place this attribute on an individual register or port or apply it globally. When applied globally, the synthesis tool packs all input, output, and I/O registers into I/O pad cells. When applied to a register, the synthesis tool packs the register into the pad cell; and when applied to a port, it packs all registers attached to the port into the pad cell.

The syn_useioff attribute can be set on the following ports:

- Top-level port.
- Register driving the top-level port.
- Lower-level port, if the register is specified as part of the port declaration.

**Verilog Syntax** object /*synthesis syn_useioff = {1 | 0} */ ;

**Verilog Example** To use this attribute globally, apply it to the module.

```
module test (a, b, clk, rst, d) /* synthesis syn_useioff = 1 */;
```

To use this attribute on individual ports, apply it to individual port declarations.

```
module test (a, b, clk, rst, d);
input a;
input b /* synthesis syn_useioff = 1 */;
```

**VHDL Syntax** attribute syn_useioff of object : objectType is true | false ;

**VHDL Example**

```
architecture archtest of test is
signal temp : std_logic;
signal temp1 : std_logic;
signal temp2 : std_logic;
signal temp3 : std_logic;
attribute syn_useioff : boolean;
attribute syn_useioff of archtest : architecture is true;
```

## translate_off/translate_on

This attribute allows you to synthesize designs originally written for use with other synthesis tools without needing to modify source code. All source code that is between these two attributes is ignored during synthesis.

**Verilog Syntax** /* pragma translate_off */
/* pragma translate_on */

**Verilog Example**

```
module real_time (ina, inb, out);
input ina, inb;
output out;
/* synthesis translate_off */
realtime cur_time;
/* synthesis translate_on */
assign out = ina & inb;
endmodule
```

**VHDL Syntax** pragma translate_off
pragma translate_on

**VHDL Example**

```
library ieee;
use ieee.std_logic_1164.all;
entity adder is
   port (a, b, cin:in std_logic;
         sum, cout:out std_logic );
end adder;
architecture behave of adder is
signal a1:std_logic;
--synthesis translate_off
constant a1:std_logic:='0';
--synthesis translate_on
begin
   sum <= (a xor b xor cin);
   cout <= (a and b) or (a and cin) or (b and cin); end behave;
```

# Synopsys Design Constraints Timing/ Physical Constraints

This section describes the new physical constraints in the Radiant software that are formatted similar to SDC formats to set physical constraints.

## ldc_create_group

**Description** Defines a single identifier that refers to a group of objects. Only slice and IO can be created currently.

**Usage** create_group -name group_name [-bbox {height width}] <objects>

**Options**

- -name - group name
- [-bbox]: is used to optionally define the maximum number of rows (R) and columns (C), of the group's bounding box, -bbox is not applicable to port
- <objects>: objects named in the group, either port, instance, pin or net.

**Example**

```
create_group -name group1[get_ports{a*}]
```

## ldc_create_region

**Description** Define a rectangular area.

**Usage** create_region -name region_name [-site site] <-width width> <-height height>

**Options**

- -name: a user-defined name of the region
- -site: a row/column Slice D location of the target device
- -width: the width of the region in columns
- -height: the height of the region in rows

**Example**

```
create_region -name region0 -site R16C2D -width 11 -height 30
```

## ldc_create_vref

**Description** Define a voltage reference. The PIO site serves as the input pin for an on-chip voltage reference.

**Usage** ldc_create_vref -name vref_name -site site_name

**Options**

- -name: voltage reference name
- -site: PIO site of the target device

**Example**

```
ldc_create_vref -name VREF1_BANK_3 -site N21
```

## ldc_set_location

**Description** When applied to a specified component, it places the component at a specified site or bank and locks the component to the site or bank. In this case, -site or -bank and <object> are valid combinations.

When applied to a specified group, it places the group at a specified site or within a region. In this case, -site or -region and -group are valid combinations.

**Usage** ldc_set_location [-site site_name]|[-bank bank_num]|[-region region_name] <object>

**Options**

- -site: a row/column Slice D location or PIO site of the target device
- -bank: bank number
- -region: user defined region name
- Object: object to be located

**Example**

```
ldc_set_location -site 11 [get_ports {A}]
ldc_set_location -region region0 [get_group {group1}]
```

## ldc_set_vcc

**Description** Sets the voltage and/or derate for the bank or core.

**Usage** ldc_set_vcc [-bank bank|-core] [-derate derate] [voltage]

**Options**

- -bank: bank number, set voltage to bank
- -core: set voltage to core
- -derate: voltage derating percent
- Voltage: can be any compatible voltage supply to that bank or the core; for example, 2.5, 3.3

**Example**

```
ldc_set_vcc -bank 1 3.3
ldc_set_vcc -bank 1 -derate -3
```

## ldc_set_port

**Description** Set constraint attributes to ports; -iobuf, is used exclusively; -vref must be combined with –iobuf.

**Usage** ldc_set_port [-iobuf [-vref <vref_name>]] <key-value list> <ports>

**Options**

- -iobuf: set IOBUF attributes. valid key-values are available in LCT file.
- -vref: voltage reference name, must be bound to -iobuf.
- <key-value list>: key-value list must be enclosed with curly braces.
- <ports>: ports to be set. If no port specified, set constraint to all ports.

**Example**

```
ldc_set_port -iobuf {IO_TYPE=HSTL15_II PULLMODE=UP} [get_ports
{A}]
```

## ldc_set_sysconfig

**Description** Set sysconfig attributes.

**Usage** ldc_set_sysconfig <key-value list>

**Options** <key-value list>: SYSCONFIG attributes, valid key-values are available in sysConfig file. key-value list must be enclosed with curly braces.

**Example**

```
ldc_set_sysconfig {CONFIG_MODE=SLAVE_SERIAL PERSISTENT=OFF
DONE_OD=ON}
```

## ldc_set_attribute

**Description** Set constraint attributes to the objects or the design if no object is specified.

**Usage** ldc_set_attribute <key-value list> <objects>

**Options** key-value list: constraint attributes to be set to object(s) or design if no object specified. Valid key-values are:

- KeyValue
- Global attributes
- FORMAT and CODE is a valid combination to implement USERCODE.
- FORMATSpecifies one of the available formats for codes: binary (BIN), hexadecimal (HEX), text (ASCII), AUTO.
- CODEFor binary or BIN format, specify a 32-bit user code string using only 1 or 0 digits. For hexadecimal or HEX format, specify an eight-character user code string using only 0 through F values. For text or ASCII format, specify a four character user code string using only alpha and numeric values.
- TEMPERATURE<number> (C|F|K).

- No <object> specified.
- Net attributes.
- PRIMARY is set to TRUE / FALSE.
- GSRTRUE: specify the net to be GSR net.
- FALSE: don't infer the net to be GSR net.
- <object> must be net.
- PRIMARYTRUE: set PRIMARY net.
- FALSE: prohibit PRIMARY net; <object> must be net.
- Clock attributes.

**Example**

```
ldc_set_attribute {FORMAT=HEX CODE=536*56D69}
```

## ldc_prohibit

**Description** Prohibits the use of a site.

**Usage** ldc_prohibit –site site

**Options** site: site name

**Example**

```
ldc_prohibit -site AB
```

# Lattice Module Reference Guide

IP Catalog provides a variety of modules to assist your design work. These modules cover a variety of common functions and can be customized. They are optimized for Lattice device architectures. Use these modules to speed up your design work and to get the most effective results.

Parameterized Module Instantiation (PM) is an alternate way to use some of the modules that come with IP Catalog. Instead of using IP Catalog, you directly instantiate a module into your HDL and customize it by setting parameters in the HDL using PMI. As a result, you may find this easier to use than IP Catalog in certain situations.

This guide describes the modules that come with IP Catalog and the related PMI modules. The descriptions mainly cover the options and controls of the configuration dialog boxes for the modules. When there is a related PMI module, the descriptions also include specifications for the PMI module's customizable parameters and ports.

**See Also**


- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86
- "Using PMI" on page 91


## Finding Modules in This Guide

The module descriptions are arranged alphabetically. To find modules by their functional type, see the following lists.

**Arithmetic_Modules**

- "Adder" on page 484
- "Complex_Multiplier (Complex_Mult)" on page 485
- "pmi_dsp" on page 486
- "Multiply_Accumulate (Mult_Accumulate)" on page 486
- "Multiply_Add_Subtract (Mult_Add_Sub)" on page 486
- "Multiplier" on page 487
- "Subtractor" on page 489

**Memory_Modules**

- "Psuedo Dual-Port RAM (RAM_DP)" on page 488
- "Single Port RAM (RAM_DQ)" on page 488

**Architecture_Modules**

- "PLL" on page 487
- "SPI_I2C" on page 489

**Table 38: PMI Module Lookup (In Alphabetical Order)**

| IP Catalog Module Name | PMI Module Name |
|---|---|
| "Adder" on page 484 | "pmi_add" on page 485 |
| "Complex_Multiplier (Complex_Mult)" on page 485 | "pmi_complex_mult" on page 485 |
| N/A | "pmi_dsp" on page 486 |
| "Multiply_Accumulate (Mult_Accumulate)" on page 486 | "pmi_mac" on page 486 |
| "Multiply_Add_Subtract (Mult_Add_Sub)" on page 486 | "pmi_multaddsub" on page 487 |
| "Multiplier" on page 487 | "pmi_mult" on page 487 |
| "PLL" on page 487 | N/A |
| "Psuedo Dual-Port RAM (RAM_DP)" on page 488 | "pmi_ram_dp" on page 488 |
| "Single Port RAM (RAM_DQ)" on page 488 | "pmi_ram_dq" on page 488 |
| "SPI_I2C" on page 489 | N/A |
| "Subtractor" on page 489 | "pmi_sub" on page 489 |

# Adder

For more information on Adder configuration and ports, refer to the *Arithmetic Modules* document.

**See Also**

- “PMI or IP Catalog?” on page 85
- “Creating IP Catalog Modules and IP” on page 86

## pmi_add

For more information on pmi_add configuration and ports, refer to the *Arithmetic Modules* document.

PMI templates can be found in the Radiant software Source Editor. Refer to “Using Templates” on page 64.

PMI source files can also be found in the <install_path>/ip/pmi/ directory.

**See Also**

- “PMI or IP Catalog?” on page 85
- “Creating IP Catalog Modules and IP” on page 86

# Complex_Multiplier (Complex_Mult)

For more information on Complex_Multiplier configuration and ports, refer to the *Arithmetic Modules* document.

**See Also**

- “PMI or IP Catalog?” on page 85
- “Creating IP Catalog Modules and IP” on page 86

## pmi_complex_mult

For more information on pmi_complex_mult configuration and ports, refer to the *Arithmetic Modules* document.

PMI templates can be found in the Radiant software Source Editor. Refer to “Using Templates” on page 64.

PMI source files can also be found in the <install_path>/ip/pmi/ directory.

**See Also**

- “PMI or IP Catalog?” on page 85
- “Creating IP Catalog Modules and IP” on page 86

# pmi_dsp

For more information on pmi_dsp configuration and ports, refer to the *Arithmetic Modules* document.

PMI templates can be found in the Radiant software Source Editor. Refer to "Using Templates" on page 64.

PMI source files can also be found in the <install_path>/ip/pmi/ directory.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

# Multiply_Accumulate (Mult_Accumulate)

For more information on Multiply_Accumulate configuration and ports, refer to the *Arithmetic Modules* document.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

# pmi_mac

For more information on pmi_mac configuration and ports, refer to the *Arithmetic Modules* document.

PMI templates can be found in the Radiant software Source Editor. Refer to "Using Templates" on page 64.

PMI source files can also be found in the <install_path>/ip/pmi/ directory.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

# Multiply_Add_Subtract (Mult_Add_Sub)

For more information on Multiply_Add_Subtract configuration and ports, rrefer to the *Arithmetic Modules* document.

**See Also**

- "PMI or IP Catalog?" on page 85

- “Creating IP Catalog Modules and IP” on page 86

## pmi_multaddsub

For more information on pmi_multaddsub configuration and ports, refer to the *Arithmetic Modules* document.

PMI templates can be found in the Radiant software Source Editor. Refer to “Using Templates” on page 64.

PMI source files can also be found in the <install_path>/ip/pmi/ directory.

**See Also**

- “PMI or IP Catalog?” on page 85
- “Creating IP Catalog Modules and IP” on page 86

# Multiplier

For more information on Multiplier configuration and ports, refer to the *Arithmetic Modules* document.

**See Also**

- “PMI or IP Catalog?” on page 85
- “Creating IP Catalog Modules and IP” on page 86

## pmi_mult

For more information on pmi_mult configuration and ports, refer to the *Arithmetic Modules* document.

PMI templates can be found in the Radiant software Source Editor. Refer to “Using Templates” on page 64.

PMI source files can also be found in the <install_path>/ip/pmi/ directory.

**See Also**

- “PMI or IP Catalog?” on page 85
- “Creating IP Catalog Modules and IP” on page 86

# PLL

For more information on PLL refer to the *iCE40 UltraPlus sysCLOCK PLL Design and Usage Guide - Radiant software* document.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

# Psuedo Dual-Port RAM (RAM_DP)

For more information on RAM_DP, refer to the *Memory Modules* document.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

## pmi_ram_dp

For more information on pmi_ram_dp configuration and ports, refer to the *Memory Modules* document.

PMI templates can be found in the Radiant software Source Editor. Refer to "Using Templates" on page 64.

PMI source files can also be found in the <install_path>/ip/pmi/ directory.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

# Single Port RAM (RAM_DQ)

For more information on Single Port RAM (RAM_DQ) refer to the *Memory Modules* document.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

## pmi_ram_dq

For more information on pmi_ram_dq configuration and ports, refer to the *Memory Modules* document.

PMI templates can be found in the Radiant software Source Editor. Refer to "Using Templates" on page 64.

PMI source files can also be found in the <install_path>/ip/pmi/ directory.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

## SPI_I2C

For more information on SPI_I2C refer to the *iCE40 UltraPlus sysCLOCK PLL Design and Usage Guide - Radiant software* document.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

## Subtractor

For more information on Subtractor configuration and ports, rrefer to the *Arithmetic Modules* document.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

## pmi_sub

For more information on pmi_sub, refer to the *Arithmetic Modules* document.

PMI templates can be found in the Radiant software Source Editor. Refer to "Using Templates" on page 64.

PMI source files can also be found in the <install_path>/ip/pmi/ directory.

**See Also**

- "PMI or IP Catalog?" on page 85
- "Creating IP Catalog Modules and IP" on page 86

# FPGA Libraries Reference Guide

Lattice Semiconductor supports some libraries used in designing FPGAs with different device architectures in a number of CAE synthesis, schematic capture, and simulation platforms. These libraries are the main front-end design libraries for Lattice FPGAs. Logic design primitives in these libraries offer flexibility and efficiency to facilitate building specific applications with Lattice devices.

A specific primitive can be found according to the device family and functional category. Primitives available to each of the following device families are listed according to appropriate functional categories.

## Primitive Library - iCE40 UltraPlus

This library includes compatible primitives supported by the iCE40 UltraPlus device family.

- Adders/Subtractors
- Flip-Flops
- Input/Output Buffer
- Memory
- Multiplier
- PIC Cells
- Special Cells
  - Clock/PLL
  - Combinatorial Primitives
  - Interfaces
  - Oscillators

- Miscellaneous

### References

For further information, a variety of technical notes for the iCE40 UltraPlus family are available on the Lattice Web site.

**Table 1: Adders/Subtractors**

| Primitive | Description | Notes |
|---|---|---|
| CCU2_B | Carry Chain. | |
| FA2 | Carry chain two bit full adder. | USER INSTANTIATION: Not Recommended; prefer RTL synthesis inference. |

**Table 2: Flip-Flops**

| Primitive | Description |
|---|---|
| FD1P3BZ | Positive edge triggered D flip-flop with positive level enable and positive level asynchronous preset. |
| FD1P3DZ | Positive edge triggered D flip-flop with positive level enable and positive level asynchronous clear. |
| FD1P3IZ | Positive edge triggered D flip-flop with positive level synchronous clear and positive level enable (clear overrides enable) |
| FD1P3JZ | Positive edge triggered D flip-flop with positive level synchronous preset and positive level enable (preset overrides enable). |
| FD1P3XZ | Positive edge triggered D flip-flop with synchronous or asynchronous set/reset. |

**Table 3: Input/Output Buffer**

| Primitive | Description | Notes |
|---|---|---|
| BB_B | Bidirectional I/O buffer with tri-state. | |
| IB | Input buffer. | |
| OB | Output buffer. | |
| OBZ_B | Output buffer with tristate. | |
| OB_RGB | Output Buffer. | USER INSTANTIATION: Illegal. Please refer to RGB and RGB1P8V. |

**Table 4: Memory**

| Primitive | Description | Notes |
|---|---|---|
| EBR_B | 4 Kb pseudo-dual port block ram with configurable write/ read width & optional bitstream initialization. | USER INSTANTIATION: Not recommended; prefer either RTL synthesis inference or IP Generation Tool. |
| PDP4K | 4Kb pseudo-dual port block RAM. | |
| SP256K | Single Port RAM that can be configured in 16K x 16 mode. This block can be cascaded using logic implemented in fabric to create larger memories. | |
| VFB_B | 256Kb single port RAM. | USER INSTANTIATION: Not recommended. Please refer to SP256K. |

**Table 5: Multiplier**

| Primitive | Description |
|---|---|
| MAC16 | DSP block capable of being configured as a multiplier, adder, subtractor, accumulator, multiply-adder or multiply-subtractor. |

**Table 6: PIC Cells**

| Primitive | Description |
|---|---|
| IFD1P3AZ | Positive edge triggered input D flip-flop with positive level enable. |
| IOL_B | Input/output registers, for both single data and double data rate. |

## Special Cells

**Table 7: Clock/PLL**

| Primitive | Description | Notes |
|---|---|---|
| PLL_B | Phase locked loop. **For internal use**. | USER INSTANTIATION: Not recommended; prefer IP Generation Tool. |

**Table 8: Combinatorial Primitives**

| Primitive | Description |
|---|---|
| LUT4 | 4-Input Look Up Table. |

**Table 9: Interfaces**

| Primitive | Description |
|---|---|
| I2C_B | The iCE40 UltraPlus device supports two I2C hard IP primitives. |
| BB_I3C | I/O with user controllable pull up resistors. |
| SPI_B | Hard SPI interface. |

**Table 10: Oscillators**

| Primitive | Description | Notes |
|---|---|---|
| HSOSC | High-frequency oscillator. Generates 48MHz nominal clock, +- 10 percent, with user programmable divider. Can drive global clock network or fabric routing. | |
| HSOSC1P8V | High-frequency oscillator. Generates 48MHz nominal clock, +- 10 percent, with user programmable divider. Can drive global clock network or fabric routing. For use when VPP_2V5 is connected to a voltage below 2.3V. | |
| HSOSC_CORE | High-frequency oscillator. Generates 48MHz nominal clock, +- 10 percent, with user programmable divider. Can drive global clock network or fabric routing. | |
| LSOSC | Low-frequency oscillator. Generates 10KHz nominal clock, +- 10 percent. Can drive global clock network or fabric routing. | |
| LSOSC1P8V | Low-frequency oscillator. Generates 10KHz nominal clock, +- 10 percent. Can drive global clock network or fabric routing. For use when VPP_2V5 is connected to a voltage below 2.3V. | |
| LSOSC_CORE | Low-frequency oscillator. Generates 10KHz nominal clock, +- 10 percent. Can drive global clock network or fabric routing. | |

**Table 11: Miscellaneous**

| Primitive | Description | Notes |
|---|---|---|
| BB_OD | Input/Output buffer. | |
| BUF | Non-inverting buffer. | USER INSTANTIATION: Yes; refer to Template Editor. |
| FILTER | Adds a 50ns delay to a signal. | |
| INV | Inverter. | |
| OFD1P3AZ | Positive edge triggered output D flip-flop with positive level enable. | |
| PUR | Power up set/reset. | |
| RGB | RGB LED drive module, containing three open drain I/O pins for RGB LED outputs. | |
| RGB1P8V | RGB LED drive module, containing three open drain I/O pins for RGB LED outputs. Trim bits are driven from Fabric. | |
| RGB_CORE | RGB LED drive module, containing three open drain I/O pins for RGB LED outputs. | |
| RGBPWM | Generates the PWM signals for the RGB LED drivers. | |
| VHI | Logic High Generator. | |
| VLO | Logic Low Generator. | |
| WARMBOOT | Allows for loading a different configuration during run time. | |

# iCECube2 Primitives vs. Radiant Software Primitives

For complete information on updating iCECube2 Primitives for iCE40 UltraPlus devices to Radiant Software Primitives for iCE40 UltraPlus devices, refer to the "Converting iCECube2 Primitives to Radiant Primitives" section of the *Migrating iCEcube2 iCE40 UltraPlus Designs to Lattice Radiant* document.

# Alphanumeric Primitives List

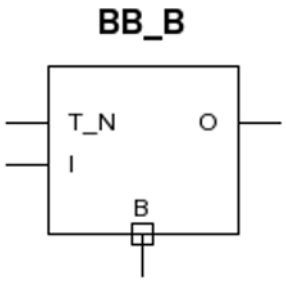
This section lists all the iCE40 UltraPlus library primitives in alphanumeric order.

## BB_B

Bidirectional I/O buffer with tri-state.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- T_N (Tri-state control, active low meaning T_N = 0 ' O = Z)
- I (Data to pad)

OUTPUT:

- O (Data from pad)
- BIDIS: B (Connection to pad)

ATTRIBUTES: syn_black_box black_box_pad_pin="B"

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Yes; refer to Template Editor.

**NOTES:**

**Table 1: BB_ Truth Table**

| INPUTS | | OUTPUTS | BIDIRECTIONAL |
|---|---|---|---|
| I | T_N | O | B |
| X | 0 | U | Z |
| X | 0 | 1 | 1 |
| X | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

X = Don’t care

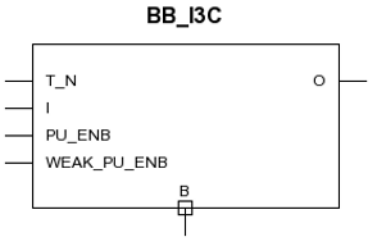
U = Unknown

## BB_I3C

I/O with user controllable pull-up resistors.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- PU_ENB (Pullup enable)
- WEAK_PU_ENB (Weak pullup enable)
- T_N (Tri-state control (active low))
- I (Data to pad)

OUTPUTS:

- O (Data from pad)
- BIDIS: B (Pad)

PARAMETERS: PULLMODE: "100K" (default), "3P3K", "6P8K", "10K", “NA”

IO_TYPE: "LVCMOS33" (default), "LVCMOS25", "LVCMOS18", "LVCMOS12", "LVDSE"

BANK_VCCIO: "3.3" (default), "2.5", "1.8", "1.2"

DRIVE: "6" (default), "NA", "2", "4", "6", "10", "12", "16"

ATTRIBUTES: syn_black_box black_box_pad_pin="B"

SYNTHESIS INFERENCE: No

USER INSTANTIATION: Yes, use Template Editor.

**Note:**

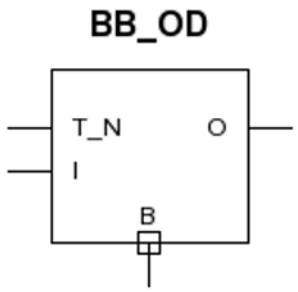
LVDSE for BIDI may not be supported, but for stability-sake, better to leave it as an option to make it 1:1 with PIO physical cell model. LCT table will correctly stop LDVSE from going through for I3C.

## BB_OD

Input/Output buffer.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- T_N (Tri-state control, active low)
- I (Data to pad)

OUTPUTS:

- O (Data from pad)
- BIDIS: B (Connection to pad)

ATTRIBUTES: syn_black_box black_box_pad_pin="B"

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Yes; refer to Template Editor.

**NOTES:**

**Table 2: BB_OD Truth Table**

| INPUTS | | OUTPUTS | BIDIRECTIONAL |
|---|---|---|---|
| I | T_N | O | B |
| X | 0 | U | Z |
| X | 0 | 1 | 1 |
| X | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 1 | Z | Z |

X = Don’t care

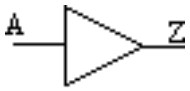
U = Unknown

## BUF

Non-inverting buffer.

Architectures Supported:

- iCE40 UltraPlus




INPUT: A

OUTPUT: Z

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: No

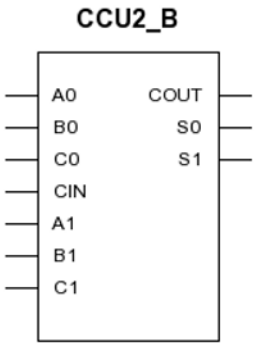
USER INSTANTIATION: Yes; refer to Template Editor.

## CCU2_B

Carry Chain.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- A0 (Input bit 0 for non-operand signal)
- B0 (Input bit 0 of the first operand)
- C0 (Input bit 0 of the second operand)
- CIN (Carry in)
- A1 (Input bit 1 for non-operand signal)
- B1 (Input bit 1 of the first operand)
- C1 (Input bit 1 of the second operand)

OUTPUTS:

- COUT (Carry out)
- S0 (Output bit 0 of the sum)
- S1 (Output bit 1 of the sum)

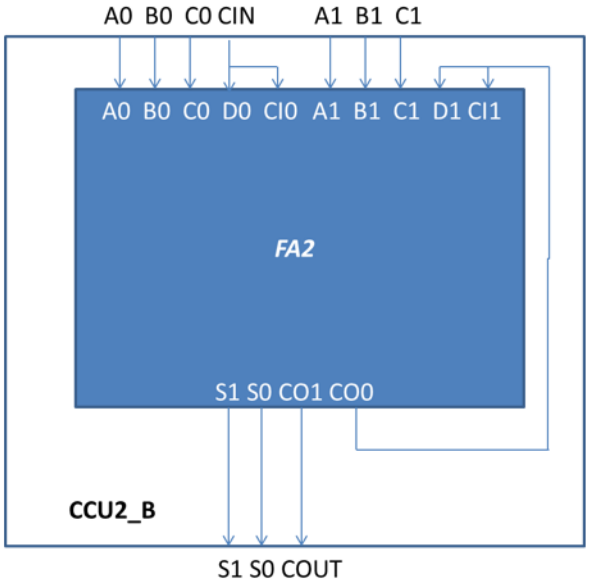
PARAMETERS: INIT0: hexadecimal value (default: "0xc33c"), INIT1: hexadecimal value (default: "0xc33c")

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Not recommended; prefer RTL synthesis inference.

IMPORTANT: Please read last paragraph that explicitly details how to use CCU2.



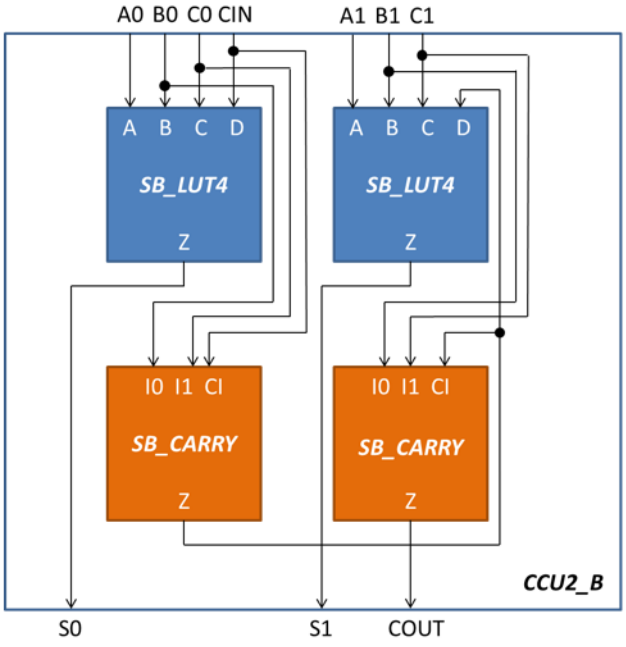




**Table 3: CCU2_B Truth Table**

| Inputs | | | | | Outputs | | |
|---|---|---|---|---|---|---|---|
| B1 | B0 | C1 | C0 | CIN | S1 | S0 | COUT |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

**Table 3: CCU2_B Truth Table (Continued)**

| Inputs | | | | | Outputs | | |
|---|---|---|---|---|---|---|---|
| **B1** | **B0** | **C1** | **C0** | **CIN** | **S1** | **S0** | **COUT** |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

NOTE: Table 3 is the truth table for an example full adder functionality using CCU2_B. B1, B0 and C1, C0 are used as operands, while A1, A0 are not used and are tied to ground.

**Rules & Restrictions:**

COUT can only go to D input of LUT above it or next CIN.

To drive VHI to the carry-in “half” a CCU2 must be used. This is done by driving B0 or C0 or both (if S0 can be a don’t care value) to VHI and leaving CIN floating. Your real CCU2 will then start with B1, C1 and the carry-in will be VHI.

To drive VLO to the carry-in “half” a CCU2 must be used. This is done by driving VLO for B0 and C0 and leaving CIN floating. Your real CCU2 starts with B1, C1 and CIN1 will be driven by VLO.

If you want to bring out last COUT to fabric, simply connect COUT to whatever is your fabric logic. MAP and Placer will work to insert a LUT and place it in the appropriate location automatically (thus ensuring rule 1 is followed).

See examples below that takes into account the above rules:

NOTE: The examples below do not make use of the A0 or A1 ports as they are not needed for carry-chain functionality.

**[Burning half a CCU2 for carry-chain]**

CCU2_B example:

```
   CCU2_B counter_12_add_4_0 (.B0(GND_net), .C0(GND_net),  CIN(),

       .B1(firstB), .C1(firstC),  .COUT(firstCarry), .S0(), .S1(firstSum));
```

[Using LUT to bring last Carry Out]

CCU2_B example:

```
   CCU2_B counter_12_add_4_7 (.B0(GND_net), .C0(out_c_5),
CIN(COUT_prev),

       .B1(GND_net), .C1(out_c_6),  .COUT(LAST_CARRY), .S0(n40),
.S1(n39));


LUT  lutInst( .D(LAST_CARRY), .F(CARRYOUT_TO_FABRIC)); //eqn : F=D
```

[Using CCU2 to bring last Carry Out]

**The logic/why it works:** Last Sum = GND XOR GND XOR CIN_PREV = lastCarryOut

CCU2_B example:

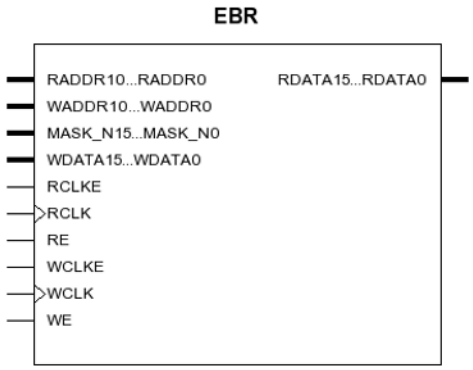
```
   CCU2_B counter_12_add_4_9 (.B0(GND_net), .C0(out_c_7),
CIN(COUT_prev),

       .B1(GND net), .C1(GND_net),  .COUT(), .S0(n40), .S1(lastCarryOut));
```

## EBR_B

4 Kb pseudo-dual port block ram with configurable write/read width and optional bitstream initialization.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- RADDR10...RADDR0 (Read address)
- WADDR10...WADDR0 (Write address)
- MASK_N15...MASK_N0 (Write mask)
- WDATA15...WDATA0 (Write data)
- RCLKE (Read clock enable)
- RCLK (Read clock)
- RE (Read enable)
- WCLKE (Write clock enable)
- WCLK (Write clock)
- WE (Write enable)

OUTPUTS:

- RDATA15...RDATA0 (Read data)

PARAMETERS: INIT_0:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_1:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_2:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_3:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_4:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_5:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_6:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_7:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_8:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_9:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_A:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_B:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_C:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_D:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_E:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

INIT_F:
"0x0000000000000000000000000000000000000000000000000000000000000000000000" (default)

DATA_WIDTH_W: "2" (default), "4", "8", "16" (Write data width in bits)

DATA_WIDTH_R: "2" (default), "4", "8", "16" (Read data width in bits)

ATTRIBUTES: syn_black_box

SAME AS: SB_RAM4K

SYNTHESIS INFERENCE: No

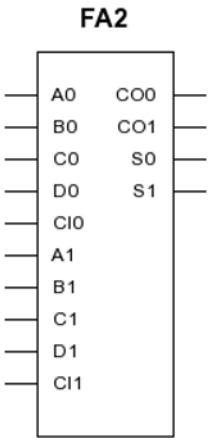
USER INSTANTIATION: Not recommended; prefer either RTL synthesis inference or IP Generation Tool.

## FA2

Carry chain two bit full adder.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- A0 (non-operand A bit 0)

- B0 (operand B bit 0)
- C0 (operand C bit 0)
- D0 (operand D bit 0)
- CI0 (carry in 0)
- A1 (non-operand A bit 1)
- B1 (operand B bit 1)
- C1 (operand C bit 1)
- D1 (operand D bit 1)
- CI1 (carry in 1)

OUTPUTS:

- CO0 (Carry out 0)
- CO1 (Carry out 1)
- S0 (Output bit 0 of the sum)
- S1 (Output bit 1 of the sum)

PARAMETERS: INIT0: hexadecimal value (default: "0xc33c")

INIT1: hexadecimal value (default: "0xc33c")

ATTRIBUTES: syn_black_box

**Table 4: FA2 Truth Table**

| Inputs | | | | | | | | Outputs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **B1** | **C1** | **D1** | **BO** | **CO** | **DO** | **CI1** | **CI0** | **S1** | **SO** | **CO1** | **CO0** |
| 0 | 0 | =CI1 | 0 | 0 | =CI0 | =CO0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | =CI1 | 0 | 0 | =CI0 | =CO0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | =CI1 | 0 | 1 | =CI0 | =CO0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | =CI1 | 0 | 1 | =CI0 | =CO0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | =CI1 | 0 | 0 | =CI0 | =CO0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | =CI1 | 0 | 0 | =CI0 | =CO0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | =CI1 | 0 | 1 | =CI0 | =CO0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | =CI1 | 0 | 1 | =CI0 | =CO0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | =CI1 | 1 | 0 | =CI0 | =CO0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | =CI1 | 1 | 0 | =CI0 | =CO0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | =CI1 | 1 | 1 | =CI0 | =CO0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | =CI1 | 1 | 1 | =CI0 | =CO0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | =CI1 | 1 | 0 | =CI0 | =CO0 | 0 | 1 | 1 | 0 | 0 |

**Table 4: FA2 Truth Table (Continued)**

| Inputs | | | | | | | | Outputs | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **B1** | **C1** | **D1** | **BO** | **CO** | **DO** | **CI1** | **CI0** | **S1** | **SO** | **CO1** | **CO0** |
| 0 | 1 | =CI1 | 1 | 0 | =CI0 | =CO0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | =CI1 | 1 | 1 | =CI0 | =CO0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | =CI1 | 1 | 1 | =CI0 | =CO0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | =CI1 | 0 | 0 | =CI0 | =CO0 | 0 | 1 | 0 | 0 | |
| 1 | 0 | =CI1 | 0 | 0 | =CI0 | =CO0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | =CI1 | 0 | 1 | =CI0 | =CO0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | =CI1 | 0 | 1 | =CI0 | =CO0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | =CI1 | 0 | 0 | =CI0 | =CO0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | =CI1 | 0 | 0 | =CI0 | =CO0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | =CI1 | 0 | 1 | =CI0 | =CO0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | =CI1 | 0 | 1 | =CI0 | =CO0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | =CI1 | 1 | 0 | =CI0 | =CO0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | =CI1 | 1 | 0 | =CI0 | =CO0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | =CI1 | 1 | 1 | =CI0 | =CO0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | =CI1 | 1 | 1 | =CI0 | =CO0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | =CI1 | 1 | 0 | =CI0 | =CO0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | =CI1 | 1 | 0 | =CI0 | =CO0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | =CI1 | 1 | 1 | =CI0 | =CO0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | =CI1 | 1 | 1 | =CI0 | =CO0 | 1 | 1 | 1 | 1 | 1 |

NOTE: In Table 2 ports A0 and A1 are tied low or are unconnected.

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Not Recommended; prefer RTL synthesis inference.

**Rules & Restrictions:**

1.Carry-out (CO0, CO1) can only go to D input of LUT above it or next (CI0, CI1).

2.To drive VHI to the first carry-in, CI0, do not tie CI0 to 1 or VHI. Instead, leave it dangling; bitgen will ensure that CI0 will correctly be VHI as expected.

3.To drive VLO to the first carry-in, CI0 should be left dangling and half a FA2 must be used. This is done by grounding B0 and C0 and leaving D0 floating. This will ensure that CI1 will correctly be VLO as expected and the second half of FA2 can be used normally.

4.If you want to bring out last carry-out (CO0, CO1) to fabric, simply connect the carry-out to whatever is your fabric logic. MAP and Placer will work to insert a LUT and place it in the appropriate location automatically (thus ensuring rule 1 is followed).

See examples below that takes into account the above rules:

NOTE: The examples below do not make use of the A0 or A1 ports as they are not needed for 2-bit full-adder functionality.

**[Burning half an FA2 for carry-chain]**

CIO & DO Dangling

```
FA2 counter_12_add_4_1 (.B0(GND_net), .C0(GND_net), .B1(VCC_net),
    .C1(out_c_0), .D1(n192), .CI1(n192), .CO0(n192), .CO1(n78),
    .S1(n45));
FA2 counter_12_add_4_9 (.B0(GND_net), .C0(out_c_7), .D0(n84), .CI0(n84),
    .B1(GND_net), .C1(GND_net), .D1(n204), .CI1(n204), .CO0(n204),
    .S0(n38));
FA2 counter_12_add_4_7 (.B0(GND_net), .C0(out_c_5), .D0(n82), .CI0(n82),
    .B1(GND_net), .C1(out_c_6), .D1(n201), .CI1(n201), .CO0(n201),
    .CO1(n84), .S0(n40), .S1(n39));
FA2 counter_12_add_4_5 (.B0(GND_net), .C0(out_c_3), .D0(n80), .CI0(n80),
    .B1(GND_net), .C1(out_c_4), .D1(n198), .CI1(n198), .CO0(n198),
    .CO1(n82), .S0(n42), .S1(n41));
FA2 counter_12_add_4_3 (.B0(GND_net), .C0(out_c_1), .D0(n78), .CI0(n78),
    .B1(GND_net), .C1(out_c_2), .D1(n195), .CI1(n195), .CO0(n195),
    .CO1(n80), .S0(n44), .S1(n43));
```

**[Using LUT to bring last Carry Out]**

FA2 example:

FA2 counter_12_add_4_7 (.B0(GND_net), .C0(out_c_5), .D0(n82), .CI0(n82),

.B1(GND_net), .C1(out_c_6), .D1(n201), .CI1(n201), .CO0(n201),

.CO1(LAST_CARRY), .S0(n40), .S1(n39));

LUT  lutInst( .D(LAST_CARRY), .F(CARRYOUT_TO_FABRIC)); //eqn : F=D

**[Using FA2 to bring last Carry Out]**

FA2 example:

FA2 counter_12_add_4_9 (.B0(GND_net), .C0(out_c_7), .D0(n84), .CI0(n84),

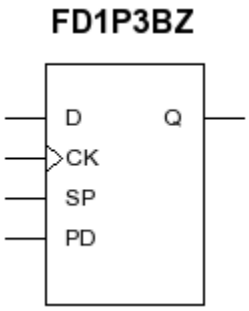
.B1(GND_net), .C1(GND_net), .S1(CARRYOUT_TO_FABRIC),
.D1(lastCarryOut), .CI1(lastCarryOut),  .CO0(lastCarryOut),.S0(n38));

## FD1P3BZ

Positive edge triggered D flip-flop with positive level enable and positive level asynchronous preset.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- D (data in)
- CK (clock)
- SP (clock enable, active high)
- PD (preset, active high)

OUTPUTS:

- Q (data out)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Not recommended; prefer RTL synthesis inference.

**Table 5: FD1P3BZ Truth Table**

| **Inputs** | | | | **Output** |
|---|---|---|---|---|
| D | SP | CK | PD | Q |
| X | 0 | X | 0 | Q |
| X | X | X | 1 | 1 |
| 0 | 1 | ↑ | 0 | 0 |
| 1 | 1 | ↑ | 0 | 1 |

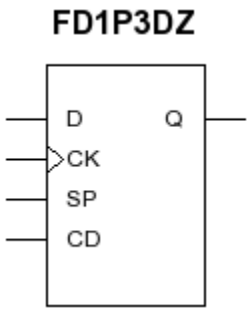
X = Don't care

## FD1P3DZ

Positive edge triggered D flip-flop with positive level enable and positive level asynchronous clear.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- D (data in)
- CK (clock)
- SP (clock enable, active high)
- CD (clear, active high)

OUTPUTS:

- Q (data out)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Not recommended; prefer RTL synthesis inference.

**Table 6: FD1P3DZ Truth Table**

| Inputs | | | | Output |
|---|---|---|---|---|
| D | SP | CK | CD | Q |
| X | 0 | X | 0 | Q |
| X | X | X | 1 | 0 |
| 0 | 1 | ↑ | 0 | 0 |
| 1 | 1 | ↑ | 0 | 1 |

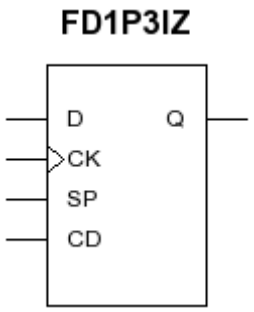
X = Don't care

## FD1P3IZ

Positive edge triggered D flip-flop with positive level synchronous clear and positive level enable (clear overrides enable).

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- D (data in)
- CK (clock)
- SP (clock enable, active high)
- CD (clear, active high. Clock enable must be active to clear).

OUTPUTS:

- Q (data out)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Not recommended; prefer RTL synthesis inference.

**Table 7: FD1P3IZ Truth Table**

| Inputs | | | | Output |
|---|---|---|---|---|
| D | SP | CK | CD | Q |
| X | 0 | X | 0 | Q |
| X | 1 | ↑ | 1 | 0 |

**Table 7: FD1P3IZ Truth Table**

| | | | | |
|---|---|---|---|---|
| 0 | 1 | ↑ | 0 | 0 |
| 1 | 1 | ↑ | 0 | 1 |

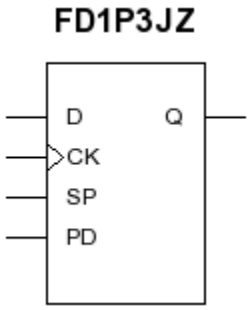
X = Don't care

## FD1P3JZ

Positive edge triggered D flip-flop with positive level synchronous preset and positive level enable (preset overrides enable).

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- D (data in)
- CK (clock)
- SP (clock enable, active high)
- PD (preset, active high. Clock enable must be active to preset).

OUTPUTS:

- Q (data out)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Not recommended; prefer RTL synthesis inference.

**Table 8: FD1P3JZ Truth Table**

| Inputs | | | | Output |
|---|---|---|---|---|
| D | SP | CK | PD | Q |
| X | 0 | X | 0 | Q |
| X | 1 | ↑ | 1 | 1 |
| 0 | 1 | ↑ | 0 | 0 |
| 1 | 1 | ↑ | 0 | 1 |

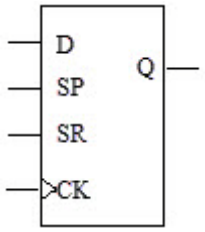
X = Don't care

## FD1P3XZ

Positive edge triggered D flip-flop with synchronous or asynchronous set/ reset.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- D (data in)
- SP (clock enable, active high)
- SR (set/reset, active high)
- CK (clock)

OUTPUTS:

- Q (data out)

PARAMETERS: REGSET: “RESET” (default), “SET”

SRMODE: “CE_OVER_LSR (default), “ASYNC”

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Not recommended; prefer RTL for synthesis inference.

## FILTER

Adds a 50ns delay to a signal

Architectures Supported:

- iCE40 UltraPlus


INPUTS:

- FILTERIN (Filter input)

OUTPUTS:

- FILTEROUT (Filter output)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SAME AS: SB_FILTER_50NS

SYNTHESIS INFERENCE: No

USER INSTANTIATION: Yes; refer to Template Editor.

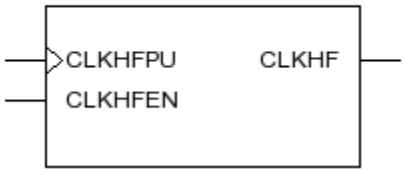
## HSOSC

High-frequency oscillator. Generates 48MHz nominal clock, +- 10 percent, with user programmable divider. Can drive global clock network or fabric routing.

Architectures Supported:

- iCE40 UltraPlus

HSOSC




INPUTS:

- CLKHFPU (Power up the oscillator. After power up, output will be stable after 100us. Active high).
- CLKHFEN (Enable the clock output. Enable should be low for the 100us power up period. Active high).

OUTPUTS:

- CLKHF (Oscillator output)

PARAMETERS: CLKHF_DIV: "0b00" (default), "0b01", "0b10", "0b11" (Clock divider selection. 0b00 = 48MHz, 0b01 = 24MHz, 0b10 = 12MHz, 0b11 = 6MHz)

ATTRIBUTES: syn_black_box

SAME AS: SB_HFOSC

SYNTHESIS INFERENCE: No

USER INSTANTIATION: Yes; refer to Template Editor.

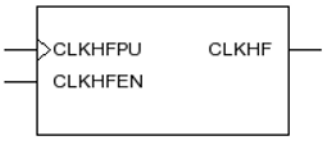
## HSOSC1P8V

High-frequency oscillator. Generates 48MHz nominal clock, +- 10 percent, with user programmable divider. Can drive global clock network or fabric routing. For use when VPP_2V5 is connected to a voltage below 2.3V.

Architectures Supported:

- iCE40 UltraPlus

HSOSC1P8V




INPUTS:

- CLKHFPU (Power up the oscillator. After power up, output will be stable after 100us. Active high).
- CLKHFEN (Enable the clock output. Enable should be low for the 100us power up period. Active high).

OUTPUTS:

- CLKHF (Oscillator output)

PARAMETERS: CLKHF_DIV: "0b00" (default), "0b01", "0b10", "0b11" (Clock divider selection. 0b00 = 48MHz, 0b01 = 24MHz, 0b10 = 12MHz, 0b11 = 6MHz)

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: No

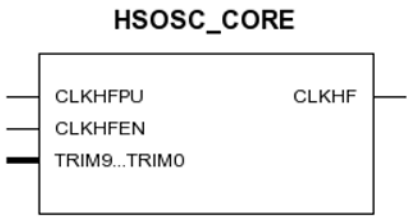
USER INSTANTIATION: Yes; refer to Template Editor.

## HSOSC_CORE

High-frequency oscillator. Generates 48MHz nominal clock, +- 10 percent, with user programmable divider. Can drive global clock network or fabric routing

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- CLKHFPU (Power up the oscillator. After power up, output will be stable after 100us. Active high).
- CLKHFEN (Enable the clock output. Enable should be low for the 100us power up period. Active high).
- TRIM9...TRIM0 ()

OUTPUTS:

- CLKHF (Oscillator output)

PARAMETERS: CLKHF_DIV: "0b00" (default), "0b01", "0b10", "0b11" (Clock divider selection. 0b00 = 48MHz, 0b01 = 24MHz, 0b10 = 12MHz, 0b11 = 6MHz); FABRIC_TRIME: "DISABLE" (default), "ENABLE" (Trim bits source selection. ENABLE - Trim bits sourced from Fabric, DISABLE - Trim bits sourced from NVCM.)

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: No

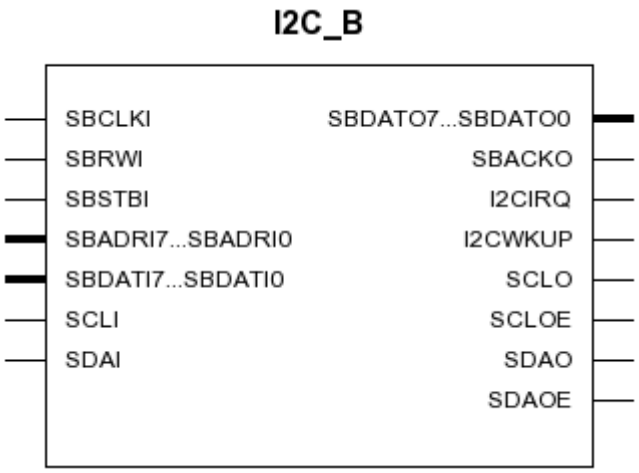
USER INSTANTIATION: Not Recommended; use HSOSC or HSOSC1P8V

## I2C_B

Hard I2C interface.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- SBCLKI (System clock input)
- SBRWI (System read/write input)
- SBSTBI (Strobe signal)
- SBADRI7...SBADRI0 (System bus control register address)
- SBDATI7...SBDATI0 (System data input)
- SCLI (Serial clock input)
- SDAI (Serial data input)

OUTPUTS:

- SBDATO7...SBDATO0 (System data output)

- SBACKO (System acknowledgment)
- I2CIRQ (I2C interrupt output)
- I2CWKUP (I2C wakeup from standby signal)
- SCLO (Serial clock output)
- SCLOE (Serial clock output enable, active high)
- SDAO (Serial data output)
- SDAOE (Serial data output enable, active high)

PARAMETERS: I2C_SLAVE_INIT_ADDR: "0b1111100001" (default), "0b1111100010" (Upper bits [9:2] can be changed through control registers. Lower bits [1:0] are fixed to 01 in the upper left and 10 in the upper right)

BUS_ADDR74: "0b0001" (default), "0b0011" (Fixed value. Upper left corner should be 0b0001, upper right corner should be 0b0011. SBADRI[7:4] should match this value to activate the IP)

I2C_CLK_DIVIDER: "0" (default), "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64", "65", "66", "67", "68", "69", "70", "71", "72", "73", "74", "75", "76", "77", "78", "79", "80", "81", "82", "83", "84", "85", "86", "87", "88", "89", "90", "91", "92", "93", "94", "95", "96", "97", "98", "99", "100", "101", "102", "103", "104", "105", "106", "107", "108", "109", "110", "111", "112", "113", "114", "115", "116", "117", "118", "119", "120", "121", "122", "123", "124", "125", "126", "127", "128", "129", "130", "131", "132", "133", "134", "135", "136", "137", "138", "139", "140", "141", "142", "143", "144", "145", "146", "147", "148", "149", "150", "151", "152", "153", "154", "155", "156", "157", "158", "159", "160", "161", "162", "163", "164", "165", "166", "167", "168", "169", "170", "171", "172", "173", "174", "175", "176", "177", "178", "179", "180", "181", "182", "183", "184", "185", "186", "187", "188", "189", "190", "191", "192", "193", "194", "195", "196", "197", "198", "199", "200", "201", "202", "203", "204", "205", "206", "207", "208", "209", "210", "211", "212", "213", "214", "215", "216", "217", "218", "219", "220", "221", "222", "223", "224", "225", "226", "227", "228", "229", "230", "231", "232", "233", "234", "235", "236", "237", "238", "239", "240", "241", "242", "243", "244", "245", "246", "247", "248", "249", "250", "251", "252", "253", "254", "255", "256", "257", "258", "259", "260", "261", "262", "263", "264", "265", "266", "267", "268", "269", "270", "271", "272", "273", "274", "275", "276", "277", "278", "279", "280", "281", "282", "283", "284", "285", "286", "287", "288", "289", "290", "291", "292", "293", "294", "295", "296", "297", "298", "299", "300", "301", "302", "303", "304", "305", "306", "307", "308", "309", "310", "311", "312", "313", "314", "315", "316", "317", "318", "319", "320", "321", "322", "323", "324", "325", "326", "327", "328", "329", "330", "331", "332", "333", "334", "335", "336", "337", "338", "339", "340", "341", "342", "343", "344", "345", "346", "347", "348", "349", "350", "351", "352", "353", "354", "355", "356", "357", "358", "359", "360", "361", "362", "363", "364", "365", "366", "367", "368", "369", "370", "371", "372", "373", "374", "375", "376", "377", "378", "379", "380", "381", "382", "383", "384", "385", "386", "387", "388", "389", "390", "391", "392", "393", "394", "395", "396", "397", "398", "399", "400", "401", "402", "403", "404",

"405", "406", "407", "408", "409", "410", "411", "412", "413", "414", "415",
"416", "417", "418", "419", "420", "421", "422", "423", "424", "425", "426",
"427", "428", "429", "430", "431", "432", "433", "434", "435", "436", "437",
"438", "439", "440", "441", "442", "443", "444", "445", "446", "447", "448",
"449", "450", "451", "452", "453", "454", "455", "456", "457", "458", "459",
"460", "461", "462", "463", "464", "465", "466", "467", "468", "469", "470",
"471", "472", "473", "474", "475", "476", "477", "478", "479", "480", "481",
"482", "483", "484", "485", "486", "487", "488", "489", "490", "491", "492",
"493", "494", "495", "496", "497", "498", "499", "500", "501", "502", "503",
"504", "505", "506", "507", "508", "509", "510", "511", "512", "513", "514",
"515", "516", "517", "518", "519", "520", "521", "522", "523", "524", "525",
"526", "527", "528", "529", "530", "531", "532", "533", "534", "535", "536",
"537", "538", "539", "540", "541", "542", "543", "544", "545", "546", "547",
"548", "549", "550", "551", "552", "553", "554", "555", "556", "557", "558",
"559", "560", "561", "562", "563", "564", "565", "566", "567", "568", "569",
"570", "571", "572", "573", "574", "575", "576", "577", "578", "579", "580",
"581", "582", "583", "584", "585", "586", "587", "588", "589", "590", "591",
"592", "593", "594", "595", "596", "597", "598", "599", "600", "601", "602",
"603", "604", "605", "606", "607", "608", "609", "610", "611", "612", "613",
"614", "615", "616", "617", "618", "619", "620", "621", "622", "623", "624",
"625", "626", "627", "628", "629", "630", "631", "632", "633", "634", "635",
"636", "637", "638", "639", "640", "641", "642", "643", "644", "645", "646",
"647", "648", "649", "650", "651", "652", "653", "654", "655", "656", "657",
"658", "659", "660", "661", "662", "663", "664", "665", "666", "667", "668",
"669", "670", "671", "672", "673", "674", "675", "676", "677", "678", "679",
"680", "681", "682", "683", "684", "685", "686", "687", "688", "689", "690",
"691", "692", "693", "694", "695", "696", "697", "698", "699", "700", "701",
"702", "703", "704", "705", "706", "707", "708", "709", "710", "711", "712",
"713", "714", "715", "716", "717", "718", "719", "720", "721", "722", "723",
"724", "725", "726", "727", "728", "729", "730", "731", "732", "733", "734",
"735", "736", "737", "738", "739", "740", "741", "742", "743", "744", "745",
"746", "747", "748", "749", "750", "751", "752", "753", "754", "755", "756",
"757", "758", "759", "760", "761", "762", "763", "764", "765", "766", "767",
"768", "769", "770", "771", "772", "773", "774", "775", "776", "777", "778",
"779", "780", "781", "782", "783", "784", "785", "786", "787", "788", "789",
"790", "791", "792", "793", "794", "795", "796", "797", "798", "799", "800",
"801", "802", "803", "804", "805", "806", "807", "808", "809", "810", "811",
"812", "813", "814", "815", "816", "817", "818", "819", "820", "821", "822",
"823", "824", "825", "826", "827", "828", "829", "830", "831", "832", "833",
"834", "835", "836", "837", "838", "839", "840", "841", "842", "843", "844",
"845", "846", "847", "848", "849", "850", "851", "852", "853", "854", "855",
"856", "857", "858", "859", "860", "861", "862", "863", "864", "865", "866",
"867", "868", "869", "870", "871", "872", "873", "874", "875", "876", "877",
"878", "879", "880", "881", "882", "883", "884", "885", "886", "887", "888",
"889", "890", "891", "892", "893", "894", "895", "896", "897", "898", "899",
"900", "901", "902", "903", "904", "905", "906", "907", "908", "909", "910",
"911", "912", "913", "914", "915", "916", "917", "918", "919", "920", "921",
"922", "923", "924", "925", "926", "927", "928", "929", "930", "931", "932",
"933", "934", "935", "936", "937", "938", "939", "940", "941", "942", "943",
"944", "945", "946", "947", "948", "949", "950", "951", "952", "953", "954",
"955", "956", "957", "958", "959", "960", "961", "962", "963", "964", "965",
"966", "967", "968", "969", "970", "971", "972", "973", "974", "975", "976",
"977", "978", "979", "980", "981", "982", "983", "984", "985", "986", "987",
"988", "989", "990", "991", "992", "993", "994", "995", "996", "997", "998",

"999", "1000", "1001", "1002", "1003", "1004", "1005", "1006", "1007", "1008", "1009", "1010", "1011", "1012", "1013", "1014", "1015", "1016", "1017", "1018", "1019", "1020", "1021", "1022", "1023" (Ratio of SBCLKI / SCLO)

SDA_INPUT_DELAYED: "0" (default), "1" (Add 50ns delay to the SDAI signal)

SDA_OUTPUT_DELAYED: "0" (default), "1" (Add 50ns delay to the SDAO signal)

FREQUENCY_PIN_SBCLKI: "NONE" (default) (SDC constraint entry on SBCLKI port of I2C.)

ATTRIBUTES: syn_black_box

SAME AS: SB_I2C

SYNTHESIS INFERENCE: No

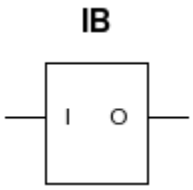
USER INSTANTIATION: Not recommended; prefer IP Generation Tool.

## IB

Input buffer.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- I (Data from pad)

OUTPUTS:

- O (Data to fabric)

PARAMETERS: None

ATTRIBUTES: syn_black_box black_box_pad_pin="I"

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Yes; refer to Template Editor.

NOTES:

**Table 9: IB Truth Table**

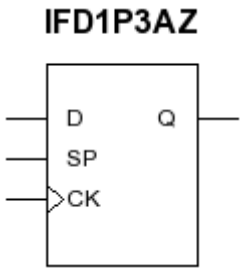
| Inputs | Output |
|---|---|
| 1 | 0 |
| 1 | 1 |
| 0 | 0 |

## IFD1P3AZ

Positive edge triggered input D flip-flop with positive level enable.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- D (data in)
- SP (clock enable, active high)
- CK (clock)

OUTPUTS:

- Q (data out)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Yes; refer to Template Editor.

**Table 10: IFD1P3AZ Truth Table**

| Inputs | | | Output |
|---|---|---|---|
| D | SP | CK | Q |
| X | 0 | X | Q |

**Table 10: IFD1P3AZ Truth Table**

| | | | |
|---|---|---|---|
| 0 | 1 | ↑ | 0 |
| 1 | 1 | ↑ | 1 |

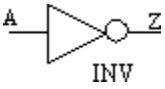
X = Don't care

## INV

Inverter.

Architectures Supported:

- iCE40 UltraPlus




INPUT:

- A

OUTPUT:

- Z

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Yes; refer to Template Editor.

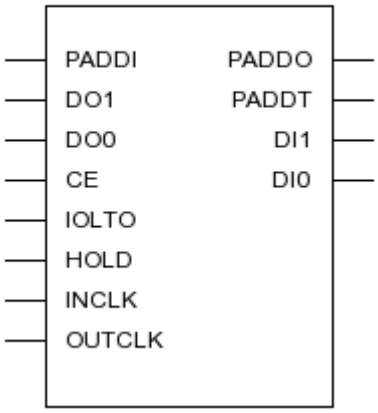
## IOL_B

Input/output registers, for both single data and double data rate.

Architectures Supported:

- iCE40 UltraPlus

**IOL_B**




INPUTS:

- PADDI (Data from pad)
- DO1 (Data to pad)
- DO0 (Data to pad)
- CE (Clock enable)
- IOLTO (Tri-state enable (active low))
- HOLD (Hold DI0 state)
- INCLK (Input clock)
- OUTCLK (Output clock)

OUTPUTS:

- PADDO (Data to pad)
- PADDT (Tri-state control to pad)
- DI1 (Data from pad)
- DI0 (Data from pad)

PARAMETERS: LATCHIN: "NONE_REG" (default), "LATCH_REG", "LATCH_BYPASS", “NONE DDR” (NONE REG = no latch, just register, LATCH_REG = latch DI0 when HOLD is asserted, LATCH_BYPASS = latch non-registered DO0, and NONE DDR = No latch, DDR operation.)

DDROUT: "NO" (default), "YES" (Select double data rate output)

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: No

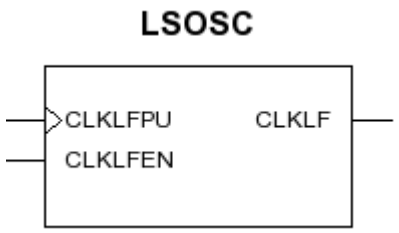
USER INSTANTIATION: Yes; refer to Template Editor.

## LSOSC

Low-frequency oscillator. Generates 10KHz nominal clock, +- 10 percent. Can drive global clock network or fabric routing.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- CLKLFPU (Power up the oscillator. After power up, output will be stable after 100us. Active high)
- CLKLFEN (Enable the clock output. Enable should be low for the 100us power up period. Active high)

OUTPUTS:

- CLKLF (Oscillator output)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SAME AS: SB_LFOSC

SYNTHESIS INFERENCE: No

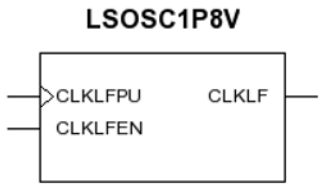
USER INSTANTIATION: Yes; refer to Template Editor.

## LSOSC1P8V

Low-frequency oscillator. Generates 10KHz nominal clock, +- 10 percent. Can drive global clock network or fabric routing. For use when VPP_2V5 is connected to a voltage below 2.3V.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- CLKLFPU (Power up the oscillator. After power up, output will be stable after 100us. Active high)
- CLKLFEN (Enable the clock output. Enable should be low for the 100us power up period. Active high)

OUTPUTS:

- CLKLF (Oscillator output)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: No

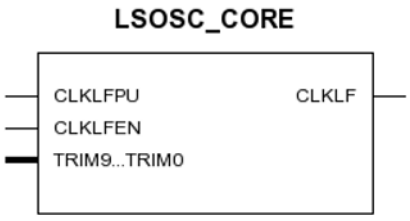
USER INSTANTIATION: Yes; refer to Template Editor.

## LSOSC_CORE

Low-frequency oscillator. Generates 10KHz nominal clock, +- 10 percent. Can drive global clock network or fabric routing.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- CLKLFPU (Power up the oscillator. After power up, output will be stable after 100us. Active high)

- CLKLFEN (Enable the clock output. Enable should be low for the 100us power up period. Active high)
- TRIM9...TRIM0 ()

OUTPUTS:

- CLKLF (Oscillator output)

PARAMETERS: FABRIC_TRIME: "DISABLE" (default), "ENABLE" (Trim bits source selection. ENABLE - Trim bits sourced from Fabric, DISABLE - Trim bits sourced from NVCM.)

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: No

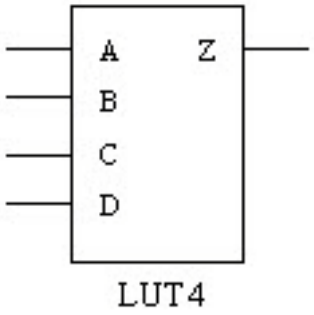
USER INSTANTIATION: Not Recommended; use LSOSC or LSOSC1P8V.

## LUT4

4-Input Look Up Table.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- A
- B
- C
- D

OUTPUT:

- Z

PARAMETERS: INIT: hexadecimal value (default: "0x0000")

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Not recommended; prefer RTL synthesis inference.

SAME AS: SB_LUT4

NOTES:

The programming of the LUT4 (that is, the 0 or 1 value of each memory location within the LUT4) is determined by the value assigned with INIT. The value is expressed in hexadecimal code. Highest memory locations are in the most significant hex digit, the lowest in the least significant digit.

**Table 11: LUT4 Truth Table**

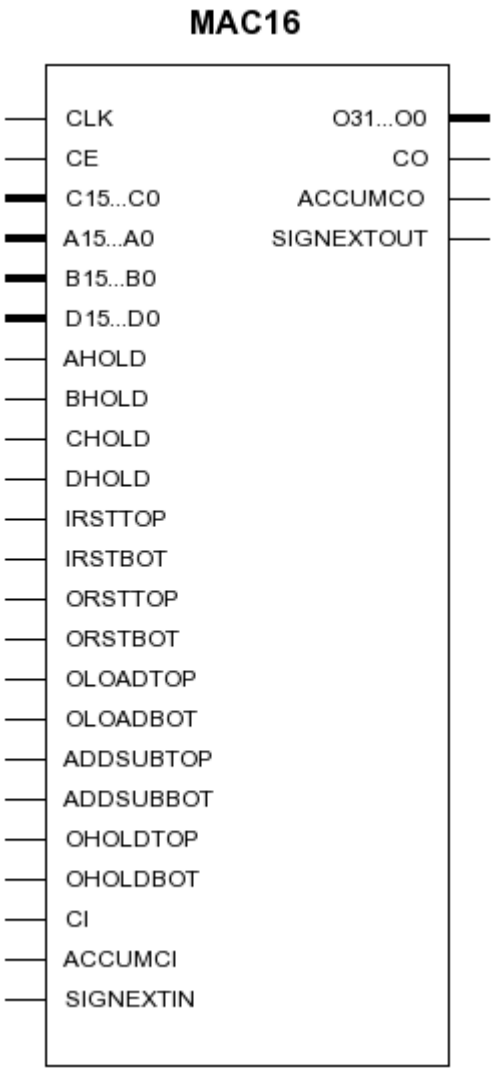
| **Inputs** | | | | **Output** |
|---|---|---|---|---|
| D | C | B | A | Z |
| 0 | 0 | 0 | 0 | INIT[0] |
| 0 | 0 | 0 | 1 | INIT[1] |
| 0 | 0 | 1 | 0 | INIT[2] |
| 0 | 0 | 1 | 1 | INIT[3] |
| 0 | 1 | 0 | 0 | INIT4] |
| 0 | 1 | 0 | 1 | INIT[5] |
| 0 | 1 | 1 | 0 | INIT[6] |
| 0 | 1 | 1 | 1 | INIT[7] |
| 1 | 0 | 0 | 0 | INIT[8] |
| 1 | 0 | 0 | 1 | INIT[9] |
| 1 | 0 | 1 | 0 | INIT[10] |
| 1 | 0 | 1 | 1 | INIT[11] |
| 1 | 1 | 0 | 0 | INIT[12] |
| 1 | 1 | 0 | 1 | INIT[13] |
| 1 | 1 | 1 | 0 | INIT[14] |
| 1 | 1 | 1 | 1 | INIT[15] |

## MAC16

DSP block capable of being configured as a multiplier, adder, subtractor, accumulator, multiply-adder or multiply-subtractor.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- CLK (Clock input. Applies to all clocked elements in the MAC16A block.)
- CE (Clock Enable input. Active High.)
- C15...C0 (Input to the C Register / Direct input to the adder accumulator.)
- A15...A0 (Input to the A Register / Direct input to the multiplier blocks / Direct input to the adder accumulator.)
- B15...B0 (Input to the B Register / Direct input to the multiplier blocks / Direct input to the adder accumulator.)
- D15...D0 (Input to the D Register / Direct input to the adder accumulator.)
- AHOLD (Hold A registers Data .Controls data flow into the input register A. Active High. 0 - Update (load) register at next active clock edge. 1 - Hold (retain) current register value, regardless of clock.)

- BHOLD (Hold B registers Data .Controls data flow into the B input register. Active High. 0 - Update (load) register at next active clock edge. 1 - Hold (retain) current register value, regardless of clock.)
- CHOLD (Hold C registers Data. Controls data flow into the C input register. Active High. 0 - Update (load) register at next active clock edge. 1 - Hold (retain) current register value, regardless of clock.),
- DHOLD (Hold D registers Data. Controls data flow into the D input register. Active High. 0 - Update (load) register at next active clock edge. 1 - Hold (retain) current register value, regardless of clock.)
- IRSTTOP (Reset input to the A and C input registers, and the pipeline registers in the upper half of the multiplier block. Active High.)
- IRSTBOT (Reset input to the B and C input registers, and the pipeline registers in the lower half of the multiplier block, and the 32-bit multiplier result pipeline register. Active High.)
- ORSTTOP (Reset the high-order bits of the accumulator register ([31:16]). Active High.)
- ORSTBOT (Reset the low-order accumulator register bits ([15:0]). Active High.)
- OLOADTOP (High-order Accumulator Register Accumulate/Load. Controls whether the accumulator register accepts the output of the adder/subtractor or whether the register is loaded with the value from Input C (or Register C, if configured). 0 - Accumulator Register [31:16] loaded with output from adder/subtractor. 1 - Accumulator Register [31:16] loaded with Input C or Register C, depending on primitive parameter value.)
- OLOADBOT (Low-order Accumulator Register Accumulate/Load. Controls whether the low-order accumulator register bits (15:0] accepts the output of the adder/subtractor or whether the register is loaded with the value from Input D (or Register D, if configured). 0 - Accumulator Register [15:0] loaded with output from adder/subtractor. 1 - Accumulator Register [15:0] loaded with Input D or Register D, depending on primitive parameter value.)
- ADDSUBTOP (High-order Add/Subtract. Controls whether the adder/ subtractor adds or subtracts. 0 - Add: W+X+HCI 1 - Subtract: W-X-HCI.)
- ADDSUBBOT (Low-order Add/Subtract. Controls whether the adder/ subtractor adds or subtracts. 0 - Add: Y+Z+LCI 1 - Subtract: Y-Z-LCI.)
- OHOLDTOP (High-order Accumulator Register Hold. Controls data flow into the high-order ([31:16]) bits of the accumulator. 0 - Update (load) register at next active clock edge. 1 - Hold (retain) current register value, regardless of clock.)
- OHOLDBOT (Low-order Accumulator Register Hold. Controls data flow into the high-order ([15:0]) bits of the accumulator. 0 - Update (load) register at next active clock edge. 1 - Hold (retain) current register value, regardless of clock.)
- CI (Carry/borrow input from lower logic tile.)
- ACCUMCI (Cascade Carry/borrow input from lower MAC16 block.)

- SIGNEXTIN (Sign extension input from lower MAC16 block.)

OUTPUTS:

- O31...O0 (32 bit MAC16 Output.
- O[31:0] - 32-bit result of a 16x16 multiply operation or a 32-bit adder/ accumulate function.
- O[31:16] - 16-bit result of an 8x8 multiply operation or a 32-bit adder/ accumulate function.
- O[15:0] -16-bit result of an 8x8 multiply operation or a 32-bit adder/ accumulate function.)
- CO (Carry/borrow output to higher logic tile.)
- ACCUMCO (Cascade Carry/borrow output to higher MAC16 block.)
- SIGNEXTOUT (Sign extension output to higher MAC16 block.)

PARAMETERS:

NEG_TRIGGER: "0b0" (default), "0b1" (Controls input clock polarity.)

A_REG: "0b0" (default), "0b1" (Input A register Control, 0b1 = registered (C1))

B_REG: "0b0" (default), "0b1" (Input B register Control, 0b1 = registered (C2))

C_REG: "0b0" (default), "0b1" (Input C register Control, 0b1 = registered (C0))

D_REG: "0b0" (default), "0b1" (Input D register Control, 0b1 = registered (C3))

TOP_8x8_MULT_REG: "0b0" (default), "0b1" (Top 8x8 multiplier output register control, 0b1 = registered (C4))

BOT_8x8_MULT_REG: "0b0" (default), "0b1" (Bottom 8x8 multiplier output register control, 0b1 = registered (C5))

PIPELINE_16x16_MULT_REG1: "0b0" (default), "0b1" (16x16 intermediate register control, 0b1 = registered (C6))

PIPELINE_16x16_MULT_REG2: "0b0" (default), "0b1" (16x16 multiplier pipeline register control, 0b1 = registered (C7))

TOPOUTPUT_SELECT: "0b00" (default), "0b01", "0b10", "0b11" (Selects top output O[31:16], 0b00 = top accumulator, 0b01 = top accumulator (registered), 0b10 = top mult8x8, 0b11 = mult16x16 (C8, C9))

TOPADDSUB_LOWERINPUT: "0b00" (default), "0b01", "0b10", "0b11" (Selects lower input for the upper adder/subtractor, 0b00 = input A, 0b01 = top mult8x8, 0b10 = mult16x16, 0b11 = sign extend from lower adder/subtractor (C10, C11))

TOPADDSUB_UPPERINPUT: "0b0" (default), "0b1" (Selects upper input for the upper adder/subtractor, 0b0 = accumulate, 0b1 = input C (C12))

TOPADDSUB_CARRYSELECT: "0b00" (default), "0b01", "0b10", "0b11" (Carry/borrow input select to upper adder/subtractor, 0b00 = constant 0, 0b01 = constant 1, 0b10 = carry from lower adder/subtractor, 0b11 = carry from lower adder/subtractor (C13, C14))

BOTOUTPUT_SELECT: "0b00" (default), "0b01", "0b10", "0b11" (Selects lower output O[15:0], 0b00 = bottom adder/subtractor, 0b01 = bottom adder/subtractor (registered), 0b10 = bottom mult8x8, 0b11 = mult16x16 (C15, C16))

BOTADDSUB_LOWERINPUT: "0b00" (default), "0b01", "0b10", "0b11" (Selects lower input for the lower adder/subtractor, 0b00 = input B, 0b01 = bottom mult8x8, 0b10 = mult16x16, 0b11 = sign extend from previous MAC16 (C17, C18))

BOTADDSUB_UPPERINPUT: "0b0" (default), "0b1" (Selects upper input for the lower adder/subtractor, 0b0 = accumulate, 0b1 = input C (C19))

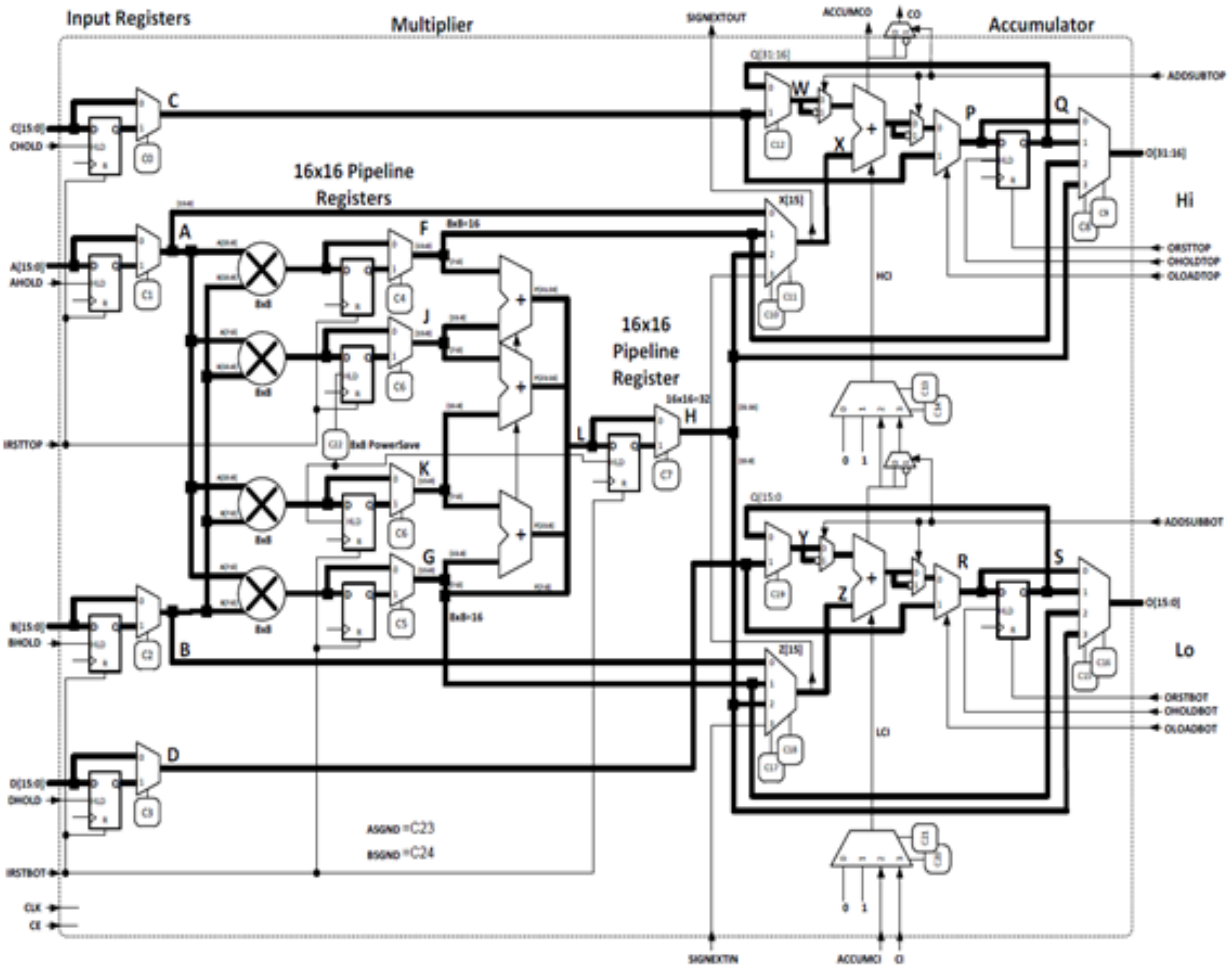
BOTADDSUB_CARRYSELECT: "0b00" (default), "0b01", "0b10", "0b11" (Carry/borrow input select to lower adder/subtractor, 0b00 = constant 0, 0b01 = constant 1, 0b10 = ACCUMCI, 0b11 = CI (C20, C21))

MODE_8x8: "0b0" (default), "0b1" (Selects 8x8 Multiplier mode and 8x8 Low-Power Multiplier Blocking Option, 0b1 = 8x8 mode (C22))

A_SIGNED: "0b0" (default), "0b1" (Indicates whether multiplier input A is signed or unsigned, 0b1 = signed)

B_SIGNED: "0b0" (default), "0b1" (Indicates whether multiplier input B is signed or unsigned. 0b1 = signed)

NOTES




ATTRIBUTES: syn_black_box

SAME AS: SB_MAC16

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Prefer RTL synthesis inference or IP Generation Tool. For direct instantiation, use PMI module instead.

**Rules & Restrictions:**

As noted by pin description, ACCUMCI and SIGNEXTIN can only come from previous MAC16's ACCUMCO and SIGNEXTOUT.

Failure to do so will cause connectivity DRC error.

If MODE_8x8 is 1, then PIPELINE_16x16_MULT_REG1 and PIPELINE_16x16_MULT_REG2 must be 0b0.

**NOTE:**

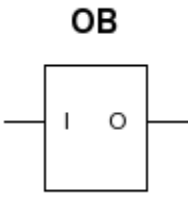
If the operation is 8x8, then these PIPELINE attributes are irrelevant. However, we have to be strict because we don’t know the intended use of the DSP. If operation is truly 16x16, then this illegal combination setting would cause an incorrect operation.

## OB

Output buffer.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- I (Data from fabric)

OUTPUTS:

- O (Data to pad)

PARAMETERS: None

ATTRIBUTES: syn_black_box black_box_pad_pin="O"

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Yes; refer to Template Editor.

NOTES:

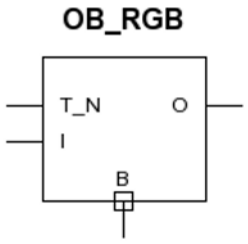
**Table 12: OB Truth Table**

| Inputs | Output |
|---|---|
| I | O |
| 1 | 1 |
| 0 | 0 |

## OB_RGB

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- T_N (Tri-state control (active low)
- I (Data to pad)

OUTPUTS:

- O (Data from pad)

BIDIS: B (Pad)

PARAMETERS: None

ATTRIBUTES: syn_black_box black_box_pad_pin="B"

SYNTHESIS INFERENCE: No

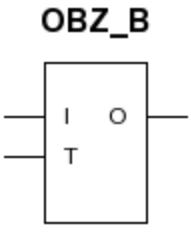
USER INSTANTIATION: Not Recommended; use RGB and RGB1P8V.

## OBZ_B

Output buffer with tristate.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- I (Data from fabric)
- T_N (Tri-state control, active low meaning T_N=0 ⟶ O = Z)

OUTPUTS:

- O (Data to pad)

PARAMETERS: None

ATTRIBUTES: syn_black_box black_box_pad_pin="O"

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Yes; refer to Template Editor.

NOTES:

**Table 13: OBZ_B Truth Table**

| Inputs | | Output |
|---|---|---|
| I | T_N | O |
| X | 0 | Z |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

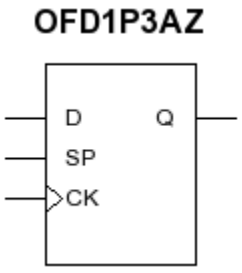
X = Don't care

## OFD1P3AZ

Positive edge triggered output D flip-flop with positive level enable.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- D (data in)
- SP (clock enable, active high)
- CK (clock)

OUTPUTS:

- Q (data out)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Yes; refer to Template Editor

NOTES:.

**Table 14: OFD1P3AZ Truth Table**

| Inputs | | | Output |
|---|---|---|---|
| D | SP | CK | Q |
| X | 0 | X | Q |
| 0 | 1 | ↑ | 0 |
| 1 | 1 | ↑ | 1 |

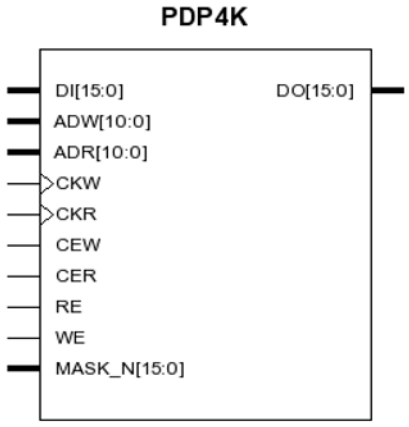
X = Don't care

## PDP4K

4Kb pseudo-dual port block RAM.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- DI[15:0] (data in)
- ADW[10:0] (write address)
- ADR[10:0] (read address)

- CKW (write clock)
- CKR (read clock)
- CEW (write clock enable, active high)
- CER (read clock enable, active high)
- RE (read enable, active high), WE (write enable, active high), MASK_N[15:0] (per-bit write enable mask, active low)

OUTPUTS:

- DO[15:0] (data output)

PARAMETERS: INITVAL_0:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_1:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_2:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_3:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_4:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_5:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_6:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_7:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_8:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_9:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_A:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_B:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_C:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_D:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_E:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

INITVAL_F:
"0x00000000000000000000000000000000000000000000000000000000000000000000" (default)

DATA_WIDTH_W: "2" (default), "4", "8", "16" (Write data width)

DATA_WIDTH_R: "2" (default), "4", "8", "16" (Read data width)

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

USER INSTANTIATION: Not recommended; prefer either RTL synthesis inference or IP Generation Tool.

**Rules & Restrictions:**

See EBR_B section. The same rules regard port mapping and initial value parameter conversion for EBR_B apply for PDP4K (only difference is that for EBR_B ⟶ INIT_#, while for PDP4K ⟶ INITVAL_#).

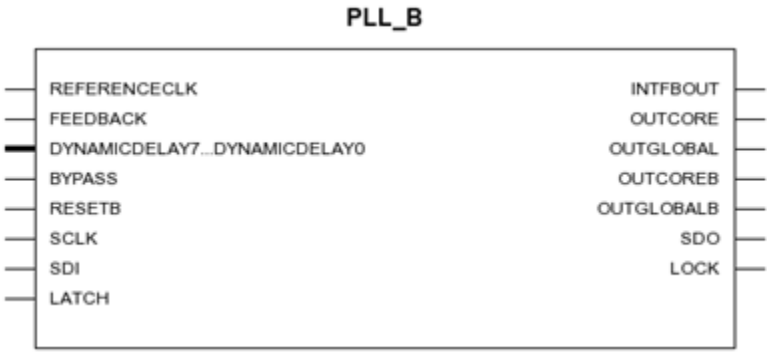
## PLL_B

Phase locked loop.

Architectures Supported:

- iCE40 UltraPlus

USER INSTANTIATION: Illegal. Please refer to PUR.




INPUTS:

- REFERENCECLK (Reference clock)
- FEEDBACK (External feedback input)
- DYNAMICDELAY7...DYNAMICDELAY0 (Dynamic control of delay for adjusting the phase alignment. Only used when DELAY_ADJUSTMENT_MODE is set to DYNAMIC. RESETB must be active before this bus can change value in order to ensure proper PLL functionality. Once value is changed user must set RESETB high and wait for PLL to lock)
- BYPASS (Bypass PLL core (directly connect REFERENCECLK to OUTCORE/OUTGLOBAL pins))
- RESET_N (Asynchronously reset the PLL, active low)
- SCLK (Clock for internal testing)
- SDI (Data for internal testing)
- LATCH (Enable low power mode, active high. OUTGLOBAL*/OUTCORE* ports are held static at their last value when ENABLE_ICEGATE_PORT* is set to 1)

OUTPUTS:

- INTFBOUT (For internal feedback PLL operation, must route this output to FEEDBACK)
- OUTCORE (Output clock A, drives regular fabric routing)
- OUTGLOBAL (Output clock A, drives global clock network)
- OUTCOREB (Output clock B, drives regular fabric routing)
- OUTGLOBALB (Output clock B, drives global clock network)
- SDO (Data for internal testing)
- LOCK (Indicates that the output signal on OUTGLOBALB/OUTCOREB is locked to the REFERENCECLK port, active high)

PARAMETERS: FEEDBACK_PATH: "SIMPLE" (default), "DELAY", "PHASE_AND_DELAY" (Selects the feedback path. SIMPLE = internal feedback, directly from VCO. DELAY = internal feedback, through the delay

adjustment block. PHASE_AND_DELAY = internal feedback, through the phase shifter and delay adjustment block)

DELAY_ADJUSTMENT_MODE_FEEDBACK: "FIXED" (default), "DYNAMIC" (Selects the mode for the delay adjustment block in the feedback path. FIXED = fixed delay, selected with the FDA_FEEDBACK parameter. DYNAMIC = dynamic delay, selected by the DYNAMICDELAY port)

FDA_FEEDBACK: "0" (default), "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15" (Sets a constant value for the delay adjustment block. OUTGLOBALA and OUTCOREA are delayed by (n+1)*150ps)

DELAY_ADJUSTMENT_MODE_RELATIVE: "FIXED" (default), "DYNAMIC" (Selects the mode for the delay adjustment block)

FDA_RELATIVE: "0" (default), "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15" (Selects a constant value for the delay adjustment block. OUTGLOBALA and OUTCOREA are delayed with regards to the B output signals, by (n+1)*150ps)

SHIFTREG_DIV_MODE: "0" (default), "1" (Selects shift register configuration. 0 = divide by 4, 1 = divide by 7. Used when FEEDBACK_PATH is set to PHASE_AND_DELAY)

PLLOUT_SELECT_PORTA: "SHIFTREG_0deg" (default), "SHIFTREG_90deg", "GENCLK", "GENCLK_HALF" (Configures the OUTCOREA and OUTGLOBALA ports. SHIFTREG_0deg = 0 degree phase shift (FEEDBACK_PATH must also be set to PHASE_AND_DELAY), SHIFTREG_90deg = 90 degree phase shift (FEEDBACK_PATH must be set to PHASE_AND_DELAY, and SHIFTREG_DIV_MODE to 0), GENCLK = the internally generated frequency and no phase shift, GENCLK_HALF = half of the internally generated frequency and no phase shift)

PLLOUT_SELECT_PORTB: "SHIFTREG_0deg" (default), "SHIFTREG_90deg", "GENCLK", "GENCLK_HALF" (Configures the OUTCOREB and OUTGLOBALB ports. SHIFTREG_0deg = 0 degree phase shift (FEEDBACK_PATH must also be set to PHASE_AND_DELAY), SHIFTREG_90deg = 90 degree phase shift (FEEDBACK_PATH must be set to PHASE_AND_DELAY, and SHIFTREG_DIV_MODE to 0), GENCLK = the internally generated frequency and no phase shift, GENCLK_HALF = half of the internally generated frequency and no phase shift)

DIVR: "0" (default), "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15" (REFERENCECLK divider)

DIVF: "0" (default), "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64" , "65", "66", "67", "68", "69", "70", "71", "72", "73", "74" , "75", "76", "77", "78", "79", "80", "81", "82", "83", "84" , "85", "86", "87", "88", "89", "90", "91", "92", "93", "94" , "95", "96", "97", "98", "99", "100", "101", "102", "103", "104" , "105", "106", "107", "108", "109",

"110", "111", "112", "113", "114" , "115", "116", "117", "118", "119", "120", "121", "122", "123", "124" , "125", "126", "127" (Feedback divider)

DIVQ: "1" (default), "2", "3", "4", "5", "6" (VCO divider)

FILTER_RANGE: "0" (default), "1", "2", "3", "4", "5", "6", "7" (PLL filter range)

EXTERNAL_DIVIDE_FACTOR: "NONE" (default), (Divide-by factor of a divider in the external feedback path. Can be set to any integer value.)

ENABLE_ICEGATE_PORTA: "0" (default), "1" (Enables the PLL power-down control for port A, 0 = disabled, 1 = controlled by LATCH input)

ENABLE_ICEGATE_PORTB: "0" (default), "1" (Enables the PLL power-down control for port B, 0 = disabled, 1 = controlled by LATCH input)

TEST_MODE: "0" (default), "1" (Enables the use of test ports, SCLK, SDI, SDO. Disabled = 0, Enabled = 1)

FREQUENCY_PIN_REFERENCECLK: "NONE" (default), (SDC Constraint entry on REFERENCECLK port of PLL)

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: No

USER INSTANTIATION: Not recommended; prefer IP Generation Tool.

**Rules & Restrictions:**

FEEDBACK port (external feedback input pin) CANNOT be VHI, VLO, or floating. Otherwise, PLL in HW will not operate correctly.

When EXTERNAL_DIVIDE_FACTOR != NONE, DELAY_ADJUSTMENT_MODE should be set to either FIXED or DYNAMIC.

When EXTERNAL_DIVIDE_FACTOR != NONE, PLLOUT_SELECT_PORTA != SHIFTREG_0deg, PLLOUT_SELECT_PORTA != SHIFTREG_90deg, PLLOUT_SELECT_PORTB != SHIFTREG_0deg, PLLOUT_SELECT_PORTB != SHIFTREG_90deg.

When FEEDBACK_PATH = DELAY, DELAY_ADJUSTMENT_MODE_FEEDBACK should be set to either FIXED or DYNAMIC.

When FEEDBACK_PATH = DELAY, PLLOUT_SELECT_PORTA != SHIFTREG_0deg, PLLOUT_SELECT_PORTA != SHIFTREG_90deg, PLLOUT_SELECT_PORTB != SHIFTREG_0deg , PLLOUT_SELECT_PORTB != SHIFTREG_90deg.

When FEEDBACK_PATH =PHASE_AND_ DELAY, DELAY_ADJUSTMENT_MODE_FEEDBACK should be set to either FIXED or DYNAMIC.

When FEEDBACK_PATH = PHASE_AND_DELAY both PLLOUT_SELECT_PORTA and PLLOUT_SELECT_PORTB must be set to either SHIFTREG_0deg or SHIFTREG_90deg.

When EXTERNAL_DIVIDE_FACTOR = NONE, make sure INTFBOUT is connected to FEEDBACK.

When EXTERNAL_DIVIDE_FACTOR != NONE, make sure INTFBOUT is NOT connected to FEEDBACK.

REFERNCECLK input must be between 10 and 133 MHz.

**How to calculate output frequency:**

When FEEDBACK_PATH = “SIMPLE,” the calculation of the period of the output clock of PLL_B is:

Output_period = reference_clk_period * __EQ__[EXTERNAL_DIVIDE_FACTOR,NONE,(2^DIVQ),1/EXTERNAL_DIVIDE_FACTOR] * (DIVR + 1) * (1/(DIVF + 1)) * __EQ__[PLLOUT_SELECT_PORTB,GENCLK_HALF,2,1]

When FEEDBACK_PATH=”DELAY,” the calculation of the period of the output clock of PLL_B is:

Output_period = reference_clk_period * __EQ__[EXTERNAL_DIVIDE_FACTOR,NONE,1,1/EXTERNAL_DIVIDE_FACTOR] * (DIVR + 1) * (1/(DIVF + 1)) * __EQ__[PLLOUT_SELECT_PORTB,GENCLK_HALF,2,1]

When FEEDBACK_PATH=”PHASE_AND_DELAY,” the calculation of the period of the output clock of PLL_B is:

Output_period = reference_clk_period * __EQ__[EXTERNAL_DIVIDE_FACTOR,NONE,1,1/EXTERNAL_DIVIDE_FACTOR] * (DIVR + 1) * (1/(DIVF + 1)) * __EQ__[PLLOUT_SELECT_PORTA,GENCLK_HALF,2,1] * __EQ__[PLLOUT_SELECT_PORTA,SHIFTREG 0deg, __EQ__[SHIFTREG_DIV_MODE,1, 7, 4], 1] * __EQ__[PLLOUT_SELECT_PORTA,SHIFTREG 0deg, __EQ__[SHIFTREG_DIV_MODE,1, 7, 4],1]

In the above equations, the sections beginning with “__EQ__” and enclosed in “[ ]” are treated as if/else statements during calculation. For example, “__EQ__[EXTERNAL_DIVIDE_FACTOR,NONE,1,1/EXTERNAL_DIVIDE_FACTOR]” means that if EXTERNAL_DIVIDE_FACTOR is set to NONE then use the value “1,” otherwise use the value “1/EXTERNAL_DIVIDE_FACTOR” in the calculation.

**NOTE:**

EXTERNAL_DIVIDE_FACTOR is embedded in each of the equations, so even if you are using external feedback in your design, you will still need to choose one of the above equations based on the FEEDBACK_PATH setting.

Example PLL_B Instantiation (External Feedback):

```
PLL_B PLL_B_inst (
    .REFERENCECLK       (REFERENCECLK   ),
    .FEEDBACK           (FEEDBACK       ),
    .DYNAMICDELAY7      (DYNAMICDELAY7  ),
    .DYNAMICDELAY6      (DYNAMICDELAY6  ),
    .DYNAMICDELAY5      (DYNAMICDELAY5  ),
    .DYNAMICDELAY4      (DYNAMICDELAY4  ),
    .DYNAMICDELAY3      (DYNAMICDELAY3  ),
    .DYNAMICDELAY2      (DYNAMICDELAY2  ),
    .DYNAMICDELAY1      (DYNAMICDELAY1  ),
    .DYNAMICDELAY0      (DYNAMICDELAY0  ),
    .BYPASS             (BYPASS         ),
    .RESET_N            (RESET_N        ),
    .SCLK               (SCLK           ),
    .SDI                (SDI            ),
    .LATCH              (LATCH          ),
    .INTFBOUT           (/* NOT CONNECTED /),
    .OUTCORE            (OUTCORE        ),
    .OUTGLOBAL          ( ),
    .OUTCOREB           (OUTCOREB       ),
    .OUTGLOBALB         ( ),
    .SDO                (SDO            ),
    .LOCK               (LOCK           )
  );
defparam PLL_B_inst.EXTERNAL_DIVIDE_FACTOR = "15";
defparam PLL_B_inst.DELAY_ADJUSTMENT_MODE_FEEDBACK =
"DYNAMIC";
```

Example PLL_B Instantiation (Internal Feedback):

```
PLL_B PLL_B_inst (
    .REFERENCECLK       (REFERENCECLK   ),
    .FEEDBACK           (INTFBOUT       ),
    .DYNAMICDELAY7      (DYNAMICDELAY7  ),
    .DYNAMICDELAY6      (DYNAMICDELAY6  ),
    .DYNAMICDELAY5      (DYNAMICDELAY5  ),
    .DYNAMICDELAY4      (DYNAMICDELAY4  ),
    .DYNAMICDELAY3      (DYNAMICDELAY3  ),
    .DYNAMICDELAY2      (DYNAMICDELAY2  ),
    .DYNAMICDELAY1      (DYNAMICDELAY1  ),
    .DYNAMICDELAY0      (DYNAMICDELAY0  ),
    .BYPASS             (BYPASS         ),
    .RESET_N            (RESET_N        ),
    .SCLK               (SCLK           ),
    .SDI                (SDI            ),
    .LATCH              (LATCH          ),
    .INTFBOUT           (INTFBOUT),
    .OUTCORE            (OUTCORE        ),
    .OUTGLOBAL          ( ),
```

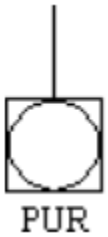
```
        .OUTCOREB               (OUTCOREB       ),
        .OUTGLOBALB             ( ),
        .SDO                    (SDO            ),
        .LOCK                   (LOCK           )
    );
defparam PLL_B_inst.EXTERNAL_DIVIDE_FACTOR = "NONE";

defparam PLL_B_inst.FEEDBACK_PATH = "SIMPLE";
```

## PUR

Power up set/reset.

Architectures Supported:

- iCE40 UltraPlus




INPUT:

- PUR_N

PARAMETERS: RST_PULSE: "1" (default), 0 or 1

SYNTHESIS INFERENCE: No

USER INSTANTIATION: Not Recommended

Description

PUR is used to reset or set all register elements in your design upon device configuration/startup. The PUR component can be connected to a net from an input buffer or an internally generated net. It is active LOW and when pulsed will set or reset all register bits to the same state as the local set or reset functionality. The pulse will be of the length specified by RST_PULSE.

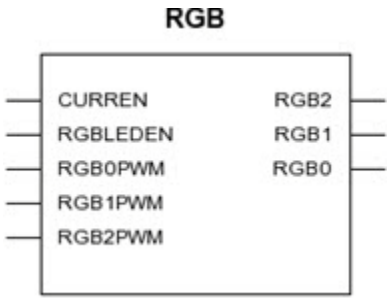
It is not necessary to connect signals for PUR to any register elements explicitly. The function will be implicitly connected globally.

# RGB

RGB LED drive module, containing three open drain I/O pins for RGB LED outputs.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- CURREN (Enable reference current to IR drivers. Enabling the drivers takes 100us to reach a stable reference current value. Active high)
- RGBLEDEN (Enable the RGB driver. Active high)
- RGB0PWM (Input data to drive RGB0 LED pin)
- RGB1PWM (Input data to drive RGB1 LED pin)
- RGB2PWM (Input data to drive RGB2 LED pin)

OUTPUTS:

- RGB2 (RGB2 LED output)
- RGB1 (RGB1 LED output)
- RGB0 (RGB0 LED output)

PARAMETERS: CURRENT_MODE: "0" (default), "1" (0 = full current, 1 = half current)

RGB0_CURRENT: "0b000000" (default), "0b000001", "0b000011", "0b000111", "0b001111", "0b011111", "0b111111" (0b000000 = 0mA, 0b000001 = 4mA, 0b000011 = 8mA, 0b000111 = 12mA, 0b001111 = 16mA, 0b011111 = 20mA, 0b111111 = 24mA. Divide by 2 for half current mode)

RGB1_CURRENT: "0b000000" (default), "0b000001", "0b000011", "0b000111", "0b001111", "0b011111", "0b111111" (0b000000 = 0mA, 0b000001 = 4mA, 0b000011 = 8mA, 0b000111 = 12mA, 0b001111 = 16mA, 0b011111 = 20mA, 0b111111 = 24mA. Divide by 2 for half current mode)

RGB2_CURRENT: "0b000000" (default), "0b000001", "0b000011", "0b000111", "0b001111", "0b011111", "0b111111" (0b000000 = 0mA,

0b000001 = 4mA, 0b000011 = 8mA, 0b000111 = 12mA, 0b001111 = 16mA, 0b011111 = 20mA, 0b111111 = 24mA. Divide by 2 for half current mode)

ATTRIBUTES: syn_black_box black_box_pad_pin="RGB2,RGB1,RBG0"

SYNTHESIS INFERENCE: No

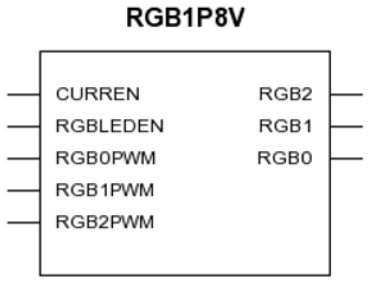
USER INSTANTIATION: Yes; refer to Template Editor.

## RGB1P8V

RGB LED drive module, containing three open drain I/O pins for RGB LED outputs. Trim bits are driven from Fabric.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- CURREN (Enable reference current to IR drivers. Enabling the drivers takes 100us to reach a stable reference current value. Active high)
- RGBLEDEN (Enable the RGB driver. Active high)
- RGB0PWM (Input data to drive RGB0 LED pin)
- RGB1PWM (Input data to drive RGB1 LED pin)
- RGB2PWM (Input data to drive RGB2 LED pin)

OUTPUTS:

- RGB2 (RGB LED output)
- RGB1 (RGB LED output)
- RGB0 (RGB LED output)

PARAMETERS: CURRENT_MODE: "0" (default), "1" (0 = full current, 1 = half current)

RGB0_CURRENT: "0b000000" (default), "0b000001", "0b000011", "0b000111", "0b001111", "0b011111", "0b111111" (0b000000 = 0mA, 0b000001 = 4mA, 0b000011 = 8mA, 0b000111 = 12mA, 0b001111 = 16mA, 0b011111 = 20mA, 0b111111 = 24mA. Divide by 2 for half current mode)

RGB1_CURRENT: "0b000000" (default), "0b000001", "0b000011", "0b000111", "0b001111", "0b011111", "0b111111" (0b000000 = 0mA, 0b000001 = 4mA, 0b000011 = 8mA, 0b000111 = 12mA, 0b001111 = 16mA, 0b011111 = 20mA, 0b111111 = 24mA. Divide by 2 for half current mode)

RGB2_CURRENT: "0b000000" (default), "0b000001", "0b000011", "0b000111", "0b001111", "0b011111", "0b111111" (0b000000 = 0mA, 0b000001 = 4mA, 0b000011 = 8mA, 0b000111 = 12mA, 0b001111 = 16mA, 0b011111 = 20mA, 0b111111 = 24mA. Divide by 2 for half current mode)

ATTRIBUTES: syn_black_box black_box_pad_pin="RGB2,RGB1,RBG0"

SYNTHESIS INFERENCE: No

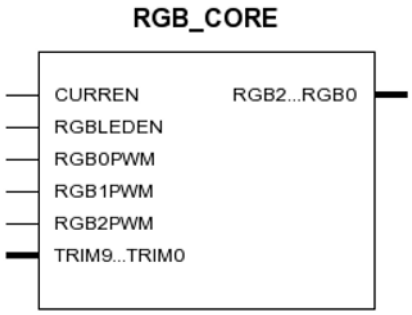
USER INSTANTIATION: Yes; refer to Template Editor.

## RGB_CORE

RGB LED drive module containing three open drain I/O pins for RGB LED outputs.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- CURREN (Enable reference current to IR drivers. Enabling the drivers takes 100us to reach a stable reference current value. Active high).
- RGBLEDEN (Enable the RGB driver. Active high)
- RGB0PWM (Input data to drive RGB0 LED pin)
- RGB1PWM (Input data to drive RGB1 LED pin)

- RGB2PWM (Input data to drive RGB2 LED pin)
- TRIM9…TRIM0 (Trim bit ports to trim output PWM signals. Pre-set value.)

OUTPUTS:

- RGB2…RGB0 (RGB LED outputs)

PARAMETERS: CURRENT_MODE: "0" (default), "1"

RGB0_CURRENT: "0b000000" (default), "0b000001", "0b000011", "0b000111", "0b001111", "0b011111", "0b111111"

RGB1_CURRENT: "0b000000" (default), "0b000001", "0b000011", "0b000111", "0b001111", "0b011111", "0b111111"

RGB2_CURRENT: "0b000000" (default), "0b000001", "0b000011", "0b000111", "0b001111", "0b011111", "0b111111"

FABRIC_TRIME: "DISABLE" (default), "ENABLE"

ATTRIBUTES: syn_black_box

SAME AS: SB_RGBA_DRV

SYNTHESIS INFERENCE: No

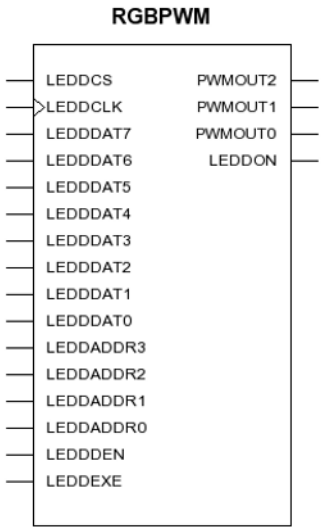
USER INSTANTIATION: Not Recommended; use RGB or RGB1P8V.

## RGBPWM

Generates the PWM signals for the RGB LED drivers.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- LEDDCS (Chip select)
- LEDDCLK (Write clock)
- LEDDDAT7...LEDDDAT0 (Data input)
- LEDDADDR3...LEDDADDR0 (Data address)
- LEDDDEN (Data enable input to indicate data and address are stable)
- LEDDEXE (Enable the IP to run the blinking sequence. When it is low, the sequence stops at the nearest OFF state)

OUTPUTS:

- PWMOUT2...PWMOUT0 (PWM outputs)

LEDDON (Indicates the LED is on)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SAME AS: SB_LEDDA_IP

SYNTHESIS INFERENCE: No

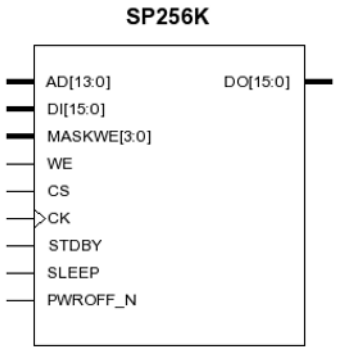
USER INSTANTIATION: Yes; refer to Template Editor.

## SP256K

Single Port RAM that can be configured in 16K x 16 mode. This block can be cascaded using logic implemented in fabric to create larger memories.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- AD[13:0] (This Address input port is used to address the location to be written during the write cycle and read during the read cycle.)
- DI[15:0] (The data input bus used to write the data into the memory location specified by address input port during the write cycle.)
- MASKWE[3:0] (The Bit Write feature where selective write to individual I/O's can be done using the Maskable Write Enable signals. Each bit masks a nibble of DI.)
- WE (When the Write Enable input is Logic High, the memory is in the write cycle. When the Write enable is Logic Low, the memory is in the Read Cycle.)
- CS (When the memory enable input is Logic High, the memory is enabled and read/write operations can be performed. When memory Enable input is logic Low, the memory is deactivated.)
- CK (This is the external clock for the memory.)
- STDBY (When this pin is active then memory goes into low leakage mode, there is no change in the output state.)
- SLEEP (This pin shut down power to periphery and maintain memory contents. The outputs of the memory are pulled low.)
- PWROFF_N (This pin turns off the power to the memory core when it is driven low. There is no memory data retention when it is driven low. When
- PWROFF_N is driven high, the SPRAM is powered on.)

OUTPUTS:

- DO[15:0] (It outputs the contents of the memory location addressed by the Address Input signals.)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: No

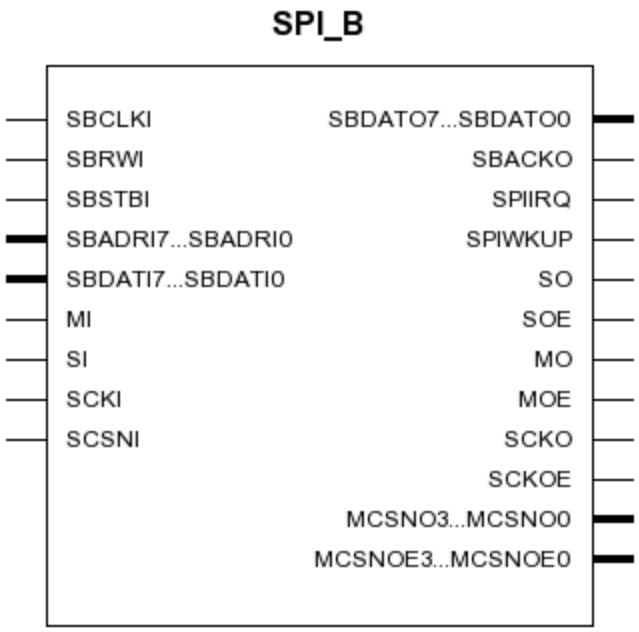
USER INSTANTIATION: Yes; refer to Template Editor.

## SPI_B

Hard SPI interface.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- SBCLKI (System clock input)
- SBRWI (System read/write input)
- SBSTBI (Strobe signal)
- SBADRI7...SBADRI0 (System bus control registers address)
- SBDATI7...SBDATI0 (System data input)
- MI (Master input from PAD)
- SI (Slave input from PAD)
- SCKI (Slave clock input from PAD)
- SCSNI (Slave chip select input from PAD)

OUTPUTS:

- SBDATO7...SBDATO0 (System data output)

- SBACKO (System acknowledgment)
- SPIIRQ (SPI interrupt output)
- SPIWKUP (SPI wake up from standby signal)
- SO (Slave output to PAD)
- SOE (Slave output enable to PAD, active high)
- MO (Master output to PAD)
- MOE (Master output enable to PAD, active high)
- SCKO (Slave clock output to PAD)
- SCKOE (Slave clock output enable to PAD)
- MCSNO3...MCSNO0 (Master chip select output to PAD)
- MCSNOE3...MCSNOE0 (Master chip select output enable to PAD)

PARAMETERS: BUS_ADDR74: "0b0000" (default), "0b0010" (Fixed value. Upper left corner should be 0b0000, upper right corner should be 0b0010. SBARDI[7:4] should match this value to activate the IP)

SPI_CLK_DIVIDER: "0" (default), "1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47", "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63" (Ratio of SCBLKI/SCKO)

FREQUENCY_PIN_SBCLKI: "NONE" (default) (SDC constraint entry on SBCLKI port of SPI.)

ATTRIBUTES: syn_black_box

SAME AS: SB_SPI

SYNTHESIS INFERENCE: No

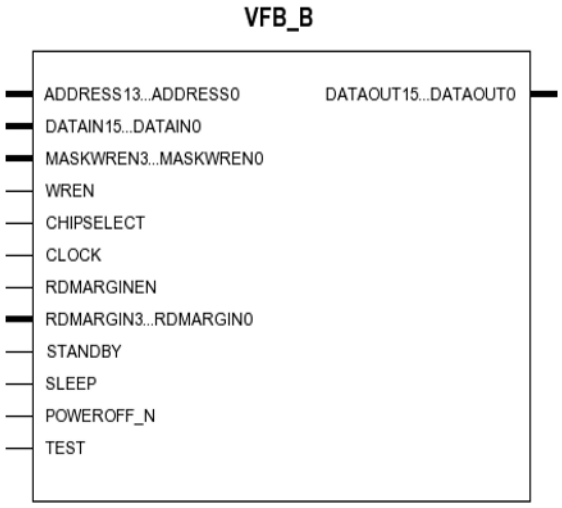
USER INSTANTIATION: Not recommended; prefer IP Generation Tool.

## VFB_B

256Kb single port RAM.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- ADDRESS13...ADDRESS0 (Read/write address)
- DATAIN15...DATAIN0 (Data input)
- MASKWREN3...MASKWREN0 (Write mask, each bit masks a nibble of DATAIN)
- WREN (Write enable)
- CHIPSELECT (Chip select)
- CLOCK (Clock)
- RDMARGINEN (Read margin enable)
- RDMARGIN3...RDMARGIN0 (Read margin)
- STANDBY (Standby mode enable, active high)
- SLEEP (Sleep enable, active high)
- POWEROFF_N (Power off enable, active low)
- TEST (Test enable)

OUTPUTS:

- DATAOUT15...DATAOUT0 (Data output)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SAME AS: SB_SPRAM256KA

SYNTHESIS INFERENCE: No

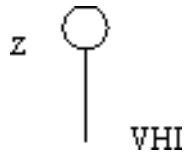
USER INSTANTIATION: Not recommended; use SP256K.

## VHI

Logic High Generator.

Architectures Supported:

- iCE40 UltraPlus




OUTPUT:

- Z

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

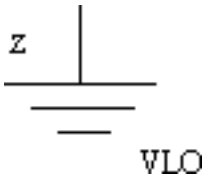
USER INSTANTIATION: Yes; refer to Template Editor.

## VLO

Logic Low Generator.

Architectures Supported:

- iCE40 UltraPlus




OUTPUT:

- Z

ATTRIBUTES: syn_black_box

SYNTHESIS INFERENCE: Yes

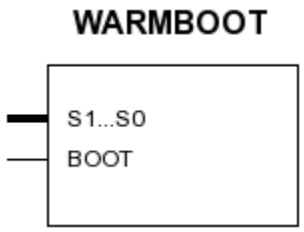
USER INSTANTIATION: Yes; refer to Template Editor.

## WARMBOOT

Allows for loading a different configuration during run time.

Architectures Supported:

- iCE40 UltraPlus




INPUTS:

- S1...S0 (Select desired configuration image)
- BOOT (Trigger the re-configuration process)

PARAMETERS: None

ATTRIBUTES: syn_black_box

SAME AS: SB_WARMBOOT

SYNTHESIS INFERENCE: No

USER INSTANTIATION: Yes; refer to Template Editor.

# DSP Inference Guide

The following are examples if DSP Inference.

```
// Multiply without register example, corresponding to the
following parameter settings of MAC16:
//      A_REG = "0b0"
//      B_REG = "0b0"
//      TOP_8x8_MULT_REG = "0b0"
//      BOT_8x8_MULT_REG = "0b0"
//      PIPELINE_16x16_MULT_REG1 = "0b0"
//      PIPELINE_16x16_MULT_REG0 = "0b0"
//      TOPOUTPUT_SELECT = "0b10"
//      BOTOUTPUT_SELECT = "0b10"

module mult_unsign_7_6(a,b,c);
    parameter A_WIDTH = 7;
    parameter B_WIDTH = 6;

    input unsigned [(A_WIDTH - 1):0] a;
    input unsigned [(B_WIDTH - 1):0] b;
    output unsigned [(A_WIDTH + B_WIDTH - 1):0] c;

    assign c = a * b;
endmodule


// Multiply/add without register example, corresponding to the
following parameter settings of
// MAC16:
//      A_REG = "0b0"
//      B_REG = "0b0"
//      TOP_8x8_MULT_REG = "0b0"
//      BOT_8x8_MULT_REG = "0b0"
//      PIPELINE_16x16_MULT_REG1 = "0b0"
//      PIPELINE_16x16_MULT_REG0 = "0b0"
//      TOPOUTPUT_SELECT = "0b00"
//      BOTOUTPUT_SELECT = "0b00"
//      TOPADDSUB_LOWERINPUT = "0b01"
//      BOTADDSUB_LOWERINPUT = "0b01"
module multaddsub_add_unsign_7_6(a,b,c,din);
    parameter A_WIDTH = 7;
    parameter B_WIDTH = 6;

    input unsigned [(A_WIDTH - 1):0] a;
    input unsigned [(B_WIDTH - 1):0] b;
    input unsigned [(A_WIDTH + B_WIDTH - 1):0] din;
    output unsigned [(A_WIDTH + B_WIDTH - 1):0] c;

    assign c = a * b + din;
endmodule

// Mult/sub with input registers, corresponding to the
following parameter settings of MAC16:
//      A_REG = "0b1"
//      B_REG = "0b1"
//      TOP_8x8_MULT_REG = "0b0"
//      BOT_8x8_MULT_REG = "0b0"
//      PIPELINE_16x16_MULT_REG1 = "0b0"
```

```
//      PIPELINE_16x16_MULT_REG0 = "0b0"
//      TOPOUTPUT_SELECT = "0b00"
//      BOTOUTPUT_SELECT = "0b00"
//      TOPADDSUB_LOWERINPUT = "0b01"
//      BOTADDSUB_LOWERINPUT = "0b01"

module multaddsub_sub_sign_ir_7_6(clk,a,b,din,c,rst,set);
    parameter A_WIDTH = 7;
    parameter B_WIDTH = 6;

    input rst;
    input set;
    input clk;
    input signed [(A_WIDTH - 1):0] a;
    input signed [(B_WIDTH - 1):0] b;
    input signed [(A_WIDTH + B_WIDTH - 1):0] din;
    output signed [(A_WIDTH + B_WIDTH - 1):0] c;

    reg signed [(A_WIDTH - 1):0] reg_a;
    reg signed [(B_WIDTH - 1):0] reg_b;
    reg signed [(A_WIDTH + B_WIDTH - 1):0] reg_din;
    assign c = reg_a * reg_b - reg_din;
    always @(posedge clk)
    begin
        if(rst)
        begin
            reg_a <= 0;
            reg_b <= 0;
            reg_din <= 0;
        end
        else if(set)
        begin
            reg_a <= -1;
            reg_b <= -1;
            reg_din <= -1;
        end
        else
        begin
            reg_a <= a;
            reg_b <= b;
           reg_din <= din;
        end
    end
endmodule
```

# Synthesis Attributes

This table lists synthesis attributes, and compares Radiant software and Lattice Diamond.

| Function | Synplify Pro Diamond | | | Synplify Pro Radiant | | |
|---|---|---|---|---|---|---|
| | **Attribute** | **Value** | **Description** | **Attribute** | **Value** | **Description** |
| ROM Synthesis | syn_romstyle | logic | Uses discrete logic primitives. | syn_romstyle | auto | Allows the synthesis tool to chose the best implementation to meet the design requirements for speed |
| | | block_rom | Specifies that the inferred ROM be mapped to the appropriate vendor-specific memory. | | logic | Uses discrete logic primitives. |
| | | distributed | Implements the ROM structure as distributed ROM. | | EBR | Specifies that the inferred ROM be mapped to the appropriate vendor-specific memory. |
| | | | | | distributed | Implements the ROM structure as distributed ROM. |
| DSP Synthesis | syn_dspstyle | DSP | Implements the multipliers as dedicated hardware blocks. | syn_multstyle | DSP | Implements the multipliers as dedicated hardware blocks. |
| | | logic | Implements the multipliers as logic. | | logic | Implements the multipliers as logic. |

# Command Line Reference Guide

This help guide contains information necessary for running the core design flow development from the command line. For tools that appear in the Radiant software graphical user interface, use Tcl commands to perform commands that are described in the “Tcl Command Reference Guide” on page 598.

## Command Line Program Overview

Lattice FPGA command line programs can be referred to as the FPGA flow core tools. These are tools necessary to run a complete design flow and are used for tasks such as module generation, design implementation, design verification, and design configuration. This topic provides an overview of those tools, their functions, and provides links to detailed usage information on each tool.

Each command line program provides multiple options for specifying device information, applying special functions using switches, designating desired output file names, and using command files. The programs also have particular default behavior often precludes the need for some syntax, making commands less complex. See “‘Command Line General Guidelines” on page 564 and “Command Line Syntax Conventions” on page 565“

To learn more about the applications, usage, and syntax for each command line program, click on the hyperlink of the command line name in the section below.

**Note**

Many of the command line programs described in this topic are run in the background when using the tools you run in the Radiant software environment. Please also note that in some cases, command line tools described here are used for earlier FPGA architectures only, are not always recommended for command line use, or are only available from the command line.

**Design Implementation Using Command Line Tools** The table below shows all of the command line tools used for various design functions, their graphical user interface counterparts, and provides functional descriptions of each.

**Table 1: The Radiant Software Core Design and Tool Chart**

| Design Function | Command Line Tool | Radiant Process | Description |
|---|---|---|---|
| **Core Implementation and Auxiliary Tools** | | | |
| Synthesis | SYNTHESIS | Synthesis Design | Runs input source files though synthesis based on Lattice Synthesis Engine options set in Strategies. |
| Synthesis | Synpwrap | Synthesis Design | Used to manage Synplicity Synplify and Synplify Pro synthesis programs. |
| Mapping | MAP | Map Design | Converts a design represented in logical terms into a network of physical components or configurable logic blocks. |
| Placement and Routing | PAR | Place & Route Design | Assigns physical locations to mapped components and creates physical connections to join components in an electrical network. |
| Static Timing Analysis | Timing | MAP Timing Report, Place & Route Timing Report | Generates reports that can be used for static timing analysis. |
| Back Annotation | Backanno | Tool does not exist in the Radiant software interface as process but employed in file export. | Distributes the physical design information back to the logical design to generate a timing simulation file. |
| Bitstream Generation | BITGEN | Bitstream | Converts a fully routed physical design into configuration bitstream data. |

**See Also**

- "Command Line Data Flow" on page 563
- "Command Line General Guidelines" on page 564
- "Command Line Syntax Conventions" on page 565
- "Invoking Core Tool Command Line Programs" on page 566

# Command Line Basics

This section provides basic instructions for running any of the core design flow development and tools from the command line.

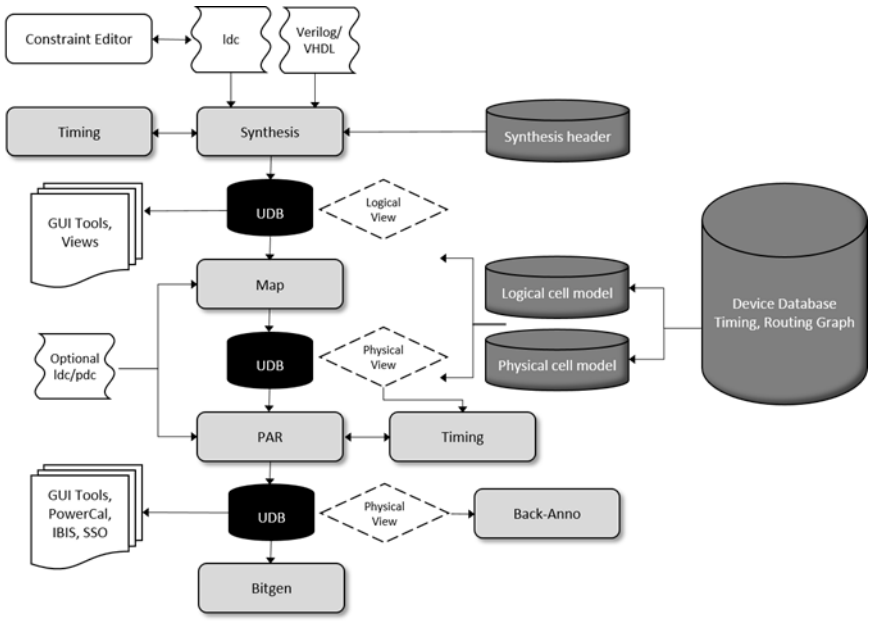
Topics include:



## Command Line Data Flow

The following chart illustrates the FPGA command line tool data flow through a typical design cycle.

**Command Line Tool Data Flow**




**See Also** "Command Line Reference Guide" on page 561

- "Command Line General Guidelines" on page 564

# Command Line General Guidelines

You can run the FPGA family Radiant software design tools from the command line. The following are general guidelines that apply.

- Files are position-dependent. Generally, they follow the convention [options] <infile> <outfile> (although order of <outfile> and <infile> are sometimes reversed). Use the **-h** command line option to check the exact syntax; for example, **par -h**.
- For any Radiant software FPGA command line program, you can invoke help on available options with the **-h** or **-help** command. See "Invoking Core Tool Command Line Programs" on page 566 for more information
- Command line options are entered on the command line in any order, preceded by a hyphen (-), and separated by spaces.
- Most command line options are case-sensitive and must be entered in lowercase letters. When an option requires an additional parameter, the option and the parameter must be separated by spaces or tabs (i.e., **-l 5** is correct, **-l5** is not).
- Options can appear anywhere on the command line. Arguments that are bound to a particular option must appear after the option (i.e., **-f** <command_file> is legal; <command_file> **-f** is not).
- For options that may be specified multiple times, in most cases the option letter must precede each parameter. For example, **-b romeo juliet** is not acceptable, while **-b romeo -b juliet** is acceptable.
- If you enter the FPGA family Radiant software application name on the command line with no arguments and the application requires one or more arguments (**par,** for example), you get a brief usage message consisting of the command line format string.
- For any Radiant software FPGA command line program, you can store program commands in a command file. Execute an entire batch of arguments by entering the program name, followed by the **–f** option, and the command file name. This is useful if you frequently execute the same arguments each time you execute a program or to avoid typing lengthy command line arguments. See "Using Command Files" on page 593.

**See Also** "Command Line Reference Guide" on page 561

- "Invoking Core Tool Command Line Tool Help" on page 567
- "Command Line Syntax Conventions" on page 565
- "Using Command Files" on page 593
- "Command Line Data Flow" on page 563

# Command Line Syntax Conventions

The following conventions are used when commands are described:

**Table 2: Command Line Syntax Conventions**

| Convention | Meaning |
|---|---|
| ( ) | Encloses a logical grouping for a choice between sub-formats. |
| [ ] | Encloses items that are optional. (Do not type the brackets.) Note that <infile[**.udb**]> indicates that the .udb extension is optional but that the extension must be UDB. |
| { } | Encloses items that may be repeated zero or more times. |
| \| | Logical OR function. You must choose one or a number of options. For example, if the command syntax says **pan up \| down \| right \| left** you enter **pan up** or **pan down** or **pan right** or **pan left**. |
| < > | Encloses a variable name or number for which you must substitute information. |
| , (comma) | Indicates a range for an integer variable. |
| - (dash) | Indicates the start of an option name. |
| : | The bind operator. Binds a variable name to a range. |
| **bold** text | Indicates text to be taken literally. You type this text exactly as shown (for example, “Type **autoroute -all -i 5** in the command area.”) Bold text is also used to indicate the name of an EPIC command, a UNIX command, or a DOS command (for example, “The **playback** command is used to execute the macro you created.”). |
| Italic text or text enclosed in angle brackets <> | Indicates text that is not to be taken literally. You must substitute information for the enclosed text. Italic text is also used to show a file directory path, for example, “the file is in the /cd/data/Radiant directory”). |
| Monospace | Indicates text that appears on the screen (for example, “File already exists.”) and text from UNIX or DOS text files. Monospace text is also used for the names of documents, files, and file extensions (for example, “Edit the autoexec.bat file” |

**See Also**


- “Command Line Reference Guide” on page 561
- “Command Line General Guidelines” on page 564
- “Invoking Core Tool Command Line Tool Help” on page 567
- “Using Command Files” on page 593

# Setting Up the Environment to Run Command Line

**For Windows** The environments for both the Radiant Tcl Console window or Radiant Standalone Tcl Console window (pnmainc.exe) are already set. You can start entering Tcl tool command or core tool commands in the console and the software will perform them.

When running the Radiant software from the Windows command line (via cmd.exe), you will need to add the following values to the following environment variables:

- PATH includes, for 64-bit

  `<Install_directory>\bin\nt64;<Install_directory>\ispfpga\bin\nt64`

  Example `<Install_directory>`:

  `c:\lscc\radiant\1.0\bin\nt64;c:\lscc\radiant\1.0\ispfpga\bin\nt64`

- FOUNDRY includes

  `set FOUNDRY= <Install_directory>\ispfpga`

**For Linux** On Linux, the Radiant software provides a similar standalone Tcl Console window (pnmainc) that sets the environment. The user can enter Tcl commands and core tool commands in it.

If you do not use the Tcl Console window, before running any of the command line programs, you need to run the following command:

- For BASH (64-bit):

  ```
  export bindir=<Install_directory>/bin/lin64
  source $bindir/radiant_env
  ```

After setting up for either Windows or PC, you can run the Radiant software executable files directly. For example, you can invoke the Place and Route program by:

```
par test_map.udb test_par.udb
```

**See Also** ▶"Invoking Core Tool Command Line Programs" on page 566

- "Invoking Core Tool Command Line Tool Help" on page 567

# Invoking Core Tool Command Line Programs

This topic provides general guidance for running the the Radiant software FPGA flow core tools. Refer to "Command Line Program Overview" on page 91 to see what these tools include and for further information.

For any the Radiant software FPGA command line programs, you begin by entering the name of the command line program followed by valid options for the program separated by spaces. Options include switches (**-f**, **-p**, **-o**, etc.), values for those switches, and file names, which are either input or output files. You start command line programs by entering a command in the UNIX™ or DOS™ command line. You can also run command line scripts or command files.

SeeTable 2 on page 565 for details and links to specific information on usage and syntax. You will find all of the usage information on the command line in the **Running FPGA Tools from the Command Line > Command Line Tool Usage** book topics.

**See Also** ▶"Command Line Reference Guide" on page 561

- "Command Line Syntax Conventions" on page 565
- "Invoking Core Tool Command Line Tool Help" on page 567
- "Setting Up the Environment to Run Command Line" on page 566
- "Using Command Files" on page 593

# Invoking Core Tool Command Line Tool Help

To get a brief usage message plus a verbose message that explains each of the options and arguments, enter the FPGA family Radiant software application name on the command line followed by **-help** or **-h**. For example, enter **bitgen -h** for option descriptions for the **bitgen** program.

To redirect this message to a file (to read later or to print out), enter this command:

command_name -help | -h > filename

The usage message is redirected to the filename that you specify.

For those FPGA family Radiant software applications that have architecture-specific command lines (e.g., iCE UltraPlus), you must enter the application name, **-help** (or **-h**), and the architecture to get the verbose usage message specific to that architecture. If you fail to specify the architecture, you get a message similar to the following:

Use '<appname> -help <architecture>' to get detailed usage for a particular architecture.

**See Also** ▶"Command Line Reference Guide" on page 561

- "Command Line Data Flow" on page 563
- "Command Line General Guidelines" on page 564
- "Command Line Syntax Conventions" on page 565

- "Setting Up the Environment to Run Command Line" on page 566
- "Using Command Files" on page 593


# Command Line Tool Usage

This section contains usage information of all of the command line tools and valid syntax descriptions for each.

Topics include:



## Running cmpl_lib.tcl from the Command Line

The cmpl_lib.tcl command allows you to perform simulation library compliation from the command line.

The following information is for running cmpl_libs.tcl from the command line using the tclsh application. The supported TCL version is 8.5 or higher.

If you don't have TCL installed, or you have an older version, perform the following:

- Add <*Radiant_install_path*>/tcltk/bin to the front of your PATH, and
- For Linux users only, add <*Radiant_install_path*>/tcltk/lib to the front of your LD_LIBRARY_PATH

**Note**

The default version of TCL on Linux could be older and may cause the script to fail. Ensure that you have TCL version 8.5 or higher.

To check TCL version, type:

```
tclsh
```

```
% info tclversion
% exit
```

For script usage, type:

```
tclsh cmpl_libs.tcl [-h|-help|]
```

**Notes**

- If Modelsim/Questa is already in your PATH and preceding any Aldec tools, you can use:

  '-sim_path .' for simplification; '.' will be added to the front of your PATH.
- Ensure the FOUNDRY environment variable is set. If the FOUNDRY environment variable is missing, then you need to set it before running the script. For details, refer to “Setting Up the Environment to Run Command Line” on page 96.
- To execute this script error free, Questasim 10.4e or a later 10.4 version, or Questasim 10.5b or a later version should be used for compilation.

Check log files under <*target_path*> (defualt = .) for any errors, as follows:

- For Linux, type:

  ```
  grep -i error *.log
  ```

- For Windows, type:

  ```
  find /i "error" *.log
  ```

**Subjects included in this topic:**

- Running compl_lib.tcl
- Command Line Syntax
- cmpl_lib.tcl Options
- Examples

**Running compl_lib.tcl** cmpl_libs.tcl allows you to compile simulation libraries from the command line.

**Command Line Syntax** tclsh <*Radiant_install_path*>/cae_library/simulation/scripts/cmpl_libs.tcl -sim_path <sim_path> [-sim_vendor {mentor<default>}] [-device {ice40up|all<default>}] [-target_path <target_path>]

**cmpl_lib.tcl Options** The table below contains all valid options for cmpl_libs.tcl

**Table 3: cmpl_lib.tcl Command Line Options**

| Option | Description |
|---|---|
| -sim_path <sim_path> | The -sim_path argument specifies the path to the simulation tool executable (binary) folder. This option is mandatory. Currently only Modelsim and Questa simulators are supported. NOTE: If <*sim_path*> has spaces, then it must be surrounded by " ". Do not use { }. |
| [-sim_vendor {mentor<default>}] | The -sim_vendor argument is optional, and intended for future use. It currently supports only Mentor Graphics simulators (Modelsim / Questa). |
| [-device {ice40up\|all<default>}] | The -device argument specifies the Lattice FPGA device to compile simulation libraries for. This argument is optional, and the default is to compile libraries for all the Lattice FPGA devices. |
| [-target_path <target_path>] | The -target_path argument specifies the target path, where you want the compiled libraries and modelsim.ini file to be located. This argument is optional, and the default target path is the current folder. NOTES: (1) This argument is recommended if the current folder is tthe Radiant software's startup (binary) folder, or if the current folder is write-protected. (2) If <*target_path*> has spaces, then it must be surrounded by " ". Do not use { }. |

**Examples** This section illustrates and describes a few examples of Simulation Libraries Compilation Tcl command.

**Example 1** The following command will compile all the Lattice FPGA libraries for Verilog simulation, and place them under the folder specified by -target_path. The path to Modelsim is specified by -sim_path.

```
tclsh <c:/lscc/radiant/1.0/>/cae_library/simulation/script/
cmpl_libs.tcl -sim_path C:/modeltech64_10.0c/win64 -target_path
c:/mti_libs
```

# Running SYNTHESIS from the Command Line

The Lattice synthesis tool SYNTHESIS allows you to synthesize Verilog and VHDL HDL source files into netlists for design entry into the Radiant software environment. Based on your strategy settings you specify in the Radiant software, a synthesis project (.synproj) file is created and then used by SYNTHESIS using the -f option. The Radiant software translates strategy options into command line options described in this topic.

Verilog source files are passed to the program using the **-ver** option and VHDL source files are passed using the **-vhd** option. For mixed language

designs the language type is automatically determined by SYNTHESIS based on the top module of the design. For IP design, you must also specify IP location (**-ip_dir)**, IP core name (**-corename**), and encrypted RTL file name (**-ertl_file**).

**Subjects included in this topic:**

- Running SYNTHESIS
- Command Line Syntax
- SYNTHESIS Options
- Examples

**Running SYNTHESIS** SYNTHESIS will convert your input netlist (.v) file into a structural verilog file that is used for the remaining mapping process.

- To run SYNTHESIS, type **synthesis** on the command line with valid options. A sample of a typical SYNTHESIS command would be as follows:

There are many command line options that give you control over the way SYNTHESIS processes the output file. Please refer to the rest of the subjects in this topic for more details. See examples.

**Command Line Syntax** **synthesis** [**-a** <arch>] [**-p** <device>] [**-sp** <performance_grade>] [**-t** <package_name>] [{**-path** <searchpath>}] [**-top** <top_module_name>] [**-ver** {<verilog_file.v>}] [**-lib** <libname>] [**-vhd** {<vhdl_file.vhd/vhdl>}] [**-udb** <udb_file.udb>] [**-hdl_param** < param_name param_value >] [**-vh2008**] [**-optimization_goal** <area | timing | balanced (default)>] [**-force_gsr** <auto(default) | yes | no>] [-**ramstyle** <auto(default) | distributed | block_ram(EBR) | registers>] [**-romstyle** <auto(default) | logic | EBR>] [**-output_edif** <filename.edf>] [**-output_hdl** <filename.v>] [**-sdc** <sdc_file.ldc>] [**-logfile** <synthesis_logfile>] [**-frequency** <target_frequency (default 200.0MHz (ICE40))>] [**-max_fanout** <max_fanout (default 1000)>] [**-bram_utilization** <bram_utilization (default 100%)>] [**-use_dsp** <0|1(default)>] [**-dsp_utilization** <dsp_utilization (default 100%)>] [**-mux_style** <auto(default) | pfu_mux | L6Mux_single | L6Mux_multiple>] [**-fsm_encoding_style** <auto(default) | one-hot | gray | binary>] [**-resolve_mixed_drivers** <0(default)|1>] [**-fix_gated_clocks** <0|1(default)] [**-use_carry_chain** <0|1(default)>] [**-carry_chain_length** <chain_length>] [**-use_io_insertion** <0|1(default)>] [**-use_io_reg** <0|1|auto(default)>] [**-resource_sharing** <0|1(default)>] [**-propagate_constants** <0|1(default)>] [**-remove_duplicate_regs** <0|1(default)>] [**-loop_limit** <max_loop_iter_cnt (default 1950)>] [**-twr_paths** <timing_path_cnt>] [**-dt**] [**-comp**] [**-syn**] [**-ifd**] [**-f** <project_file_name>]

**SYNTHESIS Options** The table below contains descriptions of all valid options for SYNTHESIS.

**Table 4: SYNTHESIS Command Line Options**

| Option | Description |
|---|---|
| **-a** <arch> | Sets the FPGA architecture. This synthesis option must be specified and if the value is set to any unsupported FPGA device architecture the command will fail. |
| **-p** <device> | Specifies the device type for the architecture (optional). |
| **-f** <proj_file_name> | Specifies the synthesis project file name (.synproj). The project file can be edited by the user to contain all desired command line options. |
| **-t** <package_name> | Specifies the package type of the device. |
| **-path** <searchpath> | Add searchpath for NGO files, Verilog “include” files (optional). |
| **-top** <top_module_name> | Name of top module (optional, but better to have to avoid ambiguity). |
| **-lib** <lib name> | Name of VHDL library (optional). |
| **-vhd** <vhdl_file.vhd/vhdl> | Names of VHDL design files (must have, if language is VHDL or mixed language). |
| **-ver** <verilog_file.v> | Names of Verilog design files (must have, if language is Verilog, or mixed language). |
| **-hdl_param** <name, value> | Allows you to override HDL parameter pairs in the design file. |
| **-optimization_goal** <balanced (default)\| area \| timing> | The synthesis tool allows you to choose among the following optimization options:<br>▶ **balanced** balances the levels of logic.<br>▶ **area** optimizes the design for area by reducing the total amount of logic used for design implementation.<br>▶ **timing** optimizes the design for timing.<br>The default setting depends on the device type. Smaller devices, such as ice40tp default to balanced. |
| **-force_gsr** <auto \| yes \| no> | Enables (**yes**) or disables (**no**) forced use of the global set/reset routing resources. When the value is auto, the synthesis tool decides whether to use the global set/reset resources. |

**Table 4: SYNTHESIS Command Line Options**

| Option | Description |
|---|---|
| **-ramstyle** <auto (default) \| distributed \| block_ram \| registers> | Sets the type of random access memory globally to *distributed*, *embedded block RAM*, or *registers*.The default is auto which attempts to determine the best implementation, that is, synthesis tool will map to technology RAM resources (EBR/Distributed) based on the resource availability.<br><br>This option will apply a syn_ramstyle attribute globally in the source to a module or to a RAM instance. To turn off RAM inference, set its value to registers.<br><br>▶ **registers** causes an inferred RAM to be mapped to registers (flip-flops and logic) rather than the technology-specific RAM resources.<br><br>▶ **distributed** causes the RAM to be implemented using the distributed RAM or PFU resources.<br><br>▶ **block_ram** causes the RAM to be implemented using the dedicated RAM resources. If your RAM resources are limited, for whatever reason, you can map additional RAMs to registers instead of the dedicated or distributed RAM resources using this attribute.<br><br>▶ **no_rw_check** (Certain technologies only). You cannot specify this value alone. Without no_rw_check, the synthesis tool inserts bypass logic around the RAM to prevent the mismatch. If you know your design does not read and write to the same address simultaneously, use no_rw_check to eliminate bypass logic. Use this value only when you cannot simultaneously read and write to the same RAM location and you want to minimize overhead logic. |

**Table 4: SYNTHESIS Command Line Options**

| Option | Description |
|---|---|
| **-romstyle** <auto (default) \| logic \| EBR> | Allows you to globally implement ROM architectures using *dedicated* , *distributed ROM*, or a *combination of the two* (auto). This applies the syn_romstyle attribute globally to the design by adding the attribute to the module or entity. You can also specify this attribute on a single module or ROM instance.<br>Specifying a syn_romstyle attribute globally or on a module or ROM instance with a value of:<br>▶ **auto** allows the synthesis tool to choose the best implementation to meet the design requirements for peformance, size, etc.<br>▶ **logic** causes the ROM to be implemented using the distributed ROM or PFU resources. Specifically, the logic value will implement ROM to logic (LUT4) or ROM technology primitives (e.g., ROM16X1, ROM32X1, ROM64X1 and so on).<br>▶ **EBR** causes the ROM to be mapped to dedicated EBR block resources. ROM address or data should be registered to map it to an EBR block. If your ROM resources are limited, for whatever reason, you can map additional ROM to registers instead of the dedicated or distributed RAM resources using this attribute.<br>Infer ROM architectures using a CASE statement in your code. For the synthesis tool to implement a ROM, at least half of the available addresses in the CASE statement must be assigned a value. For example, consider a ROM with six address bits (64 unique addresses). The case statement for this ROM must specify values for at least 32 of the available addresses. |
| **-output_hdl** <filename.v> | Specifies the name of the output Verilog netlist file. |
| **-sdc** <sdc_file.ldc> | Specifies a Synopsys design constraint (.ldc) file input. |
| **-loop_limit** <max_loop_iter_cnt (default 1950)> | Specifies the iteration limits for “for” and “while” loops in the user RTL for loops that have the loop index as a variable and not a constant.<br>The higher the loop_limit, the longer the run time. Also, for some designs, a higher loop limit may cause stack overflow during some of the optimizations during compile/synthesis.<br>The default value is 1950. Setting a higher value may cause stack overflow during some of the optimizations during synthesis. |
| **-logfile** <synthesis_logfile> | Specifies the name of the synthesis log file in ASCII format. If you do not specify a name, SYNTHESIS will output a file named synthesis.log by default. |
| **-frequency** <target_frequency (default 200.0MHz (ICE40))> | Specifies the target frequency setting. Default frequency value is 200.0 MHz. |

**Table 4: SYNTHESIS Command Line Options**

| Option | Description |
|---|---|
| **-max_fanout** <value> | Specifies maximum global fanout limit to the entire design at the top level. Default value is 1000 fanouts. |
| **-bram_utilization** <value> | Specifies block RAM utilization target setting in percent of total vacant sites. Default is 100 percent. |
| **-mux_style** <auto \| pfu_mux \| L6Mux_single \| L6Mux_multiple> | Specifies the MUX style setting. The -mux_style option controls the way the macrogenerator implements the multiplexer macros.<br>Valid options are *auto*, *pfu_mux*, *L6Mux_single*, and *L6Mux_multiple*. The default value is auto, meaning that the tool looks for the best implementation for each considered macro. |
| **-fsm_encoding_style** <auto \| one-hot \| gray \| binary> | Specifies One-Hot, Gray, or Binary style. The -fsm_encoding_style. Allows the user to determine which style is faster based on specific design implementation.<br>Valid options are *auto*, *one-hot*, *gray*, and *binary*. The default value is auto, meaning that the tool looks for the best implementation. |
| **-use_carry_chain** <0\|1> | Turns on (1) or off (0) carry chain implementation for adders. The 1 or true setting is the default. |
| **-carry_chain_length** <chain_length> | Specifies the maximum length of the carry chain. |
| **-use_io_insertion** <0\|1> | Specifies the use of I/O insertion. The 1 or true setting is the default. |
| **-use_io_reg** <0\|1> | Packs registers into I/O pad cells based on timing requirements for the target Lattice families. The value 1 enables and 0 disables (default) register packing. This applies it globally forcing the synthesis tool to pack all input, output, and I/O registers into I/O pad cells.<br>NOTE: You can place the syn_useioff attribute on an individual register or port. When applied to a register, the synthesis tool packs the register into the pad cell, and when applied to a port, packs all registers attached to the port into the pad cell.<br>The syn_useioff attribute can be set on a:<br>▶ top-level port<br>▶ register driving the top-level port<br>▶ lower-level port, only if the register is specified as part of the port declaration |
| **-resource_sharing** <0\|1> | Specifies the resource sharing option. The 1 or true setting is the default. |
| **-propagate_constants** <0\|1> | Prevents sequential optimization such as constant propagation, inverter push-through, and FSM extraction. The 1 or true setting is the default. |

**Table 4: SYNTHESIS Command Line Options**

| Option | Description |
|---|---|
| **-remove_duplicate_regs** <0\|1> | Specifies the removal of duplicate registers. The 1 or true setting is the default. |
| **-twr_paths** <timing_path_cnt> | Specifies the number of critical paths. |
| **-dt** | Disables the hardware evaluation capability. |
| **-udb** <udb_file.udb> | |
| **-ifd** | Sets option to dump intermediate files. If you run the tool with this option, it will dump about 20 intermediate encrypted Verilog files. If you supply Lattice with these files, they can be decrypted and analyzed for problems. This option is good to for analyzing simulation issues. |
| **-fix_gated_clocks** <0\|1(default)> | Allows you to enable/disable gated clock optimization. By default, the option is enabled. |
| **-vh2008** | Enables VHDL 2008 support. |

**Examples** Following are a few examples of SYNTHESIS command lines and a description of what each does.

synthesis -a "ice40tp" -p itpa08 -t SG48 -sp "6" -mux_style Auto

-use_io_insertion 1

-sdc "C:/my_radiant_tutorial/impl1/impl1.ldc"

-path "C:/lscc/radiant/1.0/ispfpga/ice40tp/data" "C:/my_radiant_tutorial/impl1" "C:/my_radiant_tutorial"

-ver "C:/my_radiant_tutorial/impl1/source/LED_control.v"

"C:/my_radiant_tutorial/impl1/source/spi_gpio.v"

"C:/my_radiant_tutorial/impl1/source/spi_gui_led_top.v"

-path "C:/my_radiant_tutorial"

-top spi_gui_led_top

-output_hdl "LEDtest_impl1.vm"

# Running Postsyn from the Command Line

The Postsyn process converts synthesized VM and integrates IPs into a completed design in UDB format for the remaining mapping process.

Usage: postsyn [-w] [-a <architecture>] [-p <device>] [-t <package>] [-sp <performance>] [-ldc <ldc_file>] [-iplist <iplist_file>] [-o <output.udb>] [-keeprtl] [-top ] <input.vm>

**Table 5:**

| Option | Description |
|---|---|
| -h(elp) | Print command help message. |
| -w | Overwrite ouptut file. |
| -a | Target architecture name. |
| -p | Target device name. |
| -t | Target package name. |
| -sp | Target performance grade. |
| -ldc | Load LDC file. |
| -iplist | Load IP list file. |
| -o | Output UDB file. |
| -keeprtl | Keep RTL view if it exists in UDB file. |
| -top | Indicate that the input is for the top design. |
| <input.vm> | Input structural Verilog file. |

# Running MAP from the Command Line

The **Map Design** process in the Radiant software environment can also be run through the command line using the **map** program. The **map** program takes an input database (.udb) file and converts this design represented as a network of device-independent components (e.g., gates and flip-flops) into a network of device-specific components (e.g., PFUs, PFFs, and EBRs) or configurable logic blocks in the form of a Unified Database (.udb) file.

Subjects included in this topic:

- Running MAP
- Command Line Syntax
- MAP Options
- Examples

**Running MAP** MAP uses the database (.udb) file that was the output of the **Synthesis** process and outputs a mapped Unified Database (.udb) file with constraints embedded.

- To run MAP, type **map** on the command line with, at minimum, the required options to describe your target technology (i.e., architecture,

device, package, and peformance grade), the input .udb along with the input .ldc file. The output .udb file specified by the **-o** option. That additional physical constraint file (*.pdc) can be applied optionally. A sample of a typical MAP command would be as follows:

**map** counter_impl1_syn.udb impl1.pdc -o counter_impl1.udb

**Note**

The **-a** (architecture) option is not necessary when you supply the part number with the **-p** option. There is also no need to specify the constraint file here, but if you do, it must be specified after the input .udb file name. The constraint file automatically takes the name "**output**" in this case, which is the name given to the output .udb file. If the output file was not specified with the **-o** option as shown in the above case, **map** would place a file named input.udb into the current working directory, taking the name of the input file. If you specify the input.ldc file and it is not there, map will error out.

There are many command line options that give you control over the way MAP processes the output file. Please refer to the rest of the subjects in this topic for more details.

**Command Line Syntax** **map** [ -h <arch> ] <infile[.udb]> [<options>]

**MAP Options** The table below contains descriptions of all valid options for MAP.

**Table 6: MAP Command Line Options**

| Option | Description |
|---|---|
| **-h** <arch> | Displays all of the available MAP command options for mapping to the specified architecture. |
| <infile[.udb]> | Specifies the output design file name in .udb format. The .udb extension is optional. |
| **-o** <name[.udb]> | Optional output design file .udb. |
| **-mp** <name[.mrp]> | Optional report file (.mrp). |
| **-xref_sig** | Report signal cross reference for renamed signals. |
| **-xref_sym** | Report symbol cross reference for renamed symbols. |
| **-u** | Unclip unused instances. |

**Examples** Following are some examples of MAP command lines and a description of what each does.

**Example 1** The following command maps an input database file named mapped.udb and outputs a mapped Unified Database file named mapped.udb.

map counter_impl1_syn.udb impl1.pdc -o counter_impl1.udb

**See Also** ►"Command Line Data Flow" on page 563

- "Command Line Program Overview" on page 561

# Running PAR from the Command Line

The **Place & Route Design** process in the Radiant software environment can also be run through the command line using the **par** program. The **par** program takes an input mapped Unified Database (.udb) file and further places and routes the design, assigning locations of physical components on the device and adding the inter-connectivity, outputting a placed and routed .udb file.

The Implementation Engine multi-tasking option available in UNIX is explained in detail here because the option is not available for PCs.

Subjects included in this topic:

- Running PAR
- Command Line Syntax
- General Options
- Placement Options
- Routing Options
- PAR Explorer (-exp) Options
- Examples
- PAR Multi-Tasking Options

**Running PAR** PAR uses your mapped Unified Database (.udb) file that were the outputs of the **Map Design** process or the **map** program. With these inputs, **par** outputs a new placed-and-routed .udb file, a PAR report (.par) file, and a PAD (specification (.pad) file that contains I/O placement information.

- To run PAR, type **par** on the command line with at minimum, the name of the input .udb file and the desired name of the output .udb file. "Design constraints from previous stages are automatically embedded in the input .udb file, however the par program can accept additional constraints with either a .pdc or .sdc file" A sample of a basic PAR command would be as follows:

**par** input.udb output.udb

There are many command line options that give you control over PAR. Please refer to the rest of the subjects in this topic for more details.

**Command Line Syntax** **par** [-w] [-n <iterations:0,100>] [-t <iteration:0,100>] [-stopzero] [-s <savecount:0,100>] [-m <nodelistfile>] [-r ] [-k] [-p ] [-x] [-pack <density:0,100>] [-sp <setupspeedgrade>] [-hsp <holdspeedgrade>] [-dh] [-hos] [-sort <method>] <infile> <outfile> [<pdcfile>]

### Note

All filenames without special switches must be in the order <infile> <outfile> <pdcfile>. Options may exist in any order.

## General Options

**Table 7: General PAR Command Line Options**

| Option | Description |
|---|---|
| **-f** | Read par command line arguments and switches from file. |
| **-w** | Overwrite. Allows overwrite of an existing file (including input file). |
| **-n** | Number of iterations (seeds). Use "-n 0" to run until fully routed and a timing score of zero is achieved. Default: 1. |
| **-t** | Start at this placer cost table entry. Default is 1. |
| **-stopzero** | Stop running iterations once a timing score of zero is achieved. |
| **-s** | Save "n" best results for this run. Default: Save All. |
| **-m** | Multi task par run. File "<node list file>", contains a list of node names to run the jobs on. |
| **-p** | Don't run placement. |
| **-r** | Don't run router. |
| **-k** | Keep existing routing in input UDB file.<br>Note: only meaningful when used with -p. |
| **-x** | Ignore timing constraints. |
| **-pack** | Set the packing density parameter.<br>Default: auto. |
| **-sp** | Change performance grade for setup optimization. Default: Keep current performance grade. |
| **-hsp** | Change performance grade for hold optimization.<br>Default: M. |
| **-dh** | Disable hold timing correction. |
| **-hos** | Prioritize hold timing correction over setup performance. |

**Table 7: General PAR Command Line Options**

| Option | Description |
|---|---|
| **-sort** | Set the sorting method for ranking multiple iterations. <method> "c" sorts by cumulative slack, "w" sorts by worst slack.<br>Default: c. |
| <infile> | Name of input UDB file. |
| <outfile> | Name of output UDB file. |

**Table 8: PAR Placement Command Line Options**

| Option | Description |
|---|---|
| <pdcfile> | Name of optional constraint file.<br>Note: the contents of <pdcfile> will overwrite all constraints saved in the input UDB file <infile>. |

**Examples** Following are a few examples of PAR command lines and a description of what each does.

**Example 1** The following command places and routes the design in the file input.udb and writes the placed and routed design to output.udb.

```
par input.udb output.udb
```

**Example 2** The following command runs 20 place and route iterations. The iterations begin at cost table entry 5. Only the best 3 output design files are saved.

```
par -n 20 -t 5 -s 3 input.udb output.udb
```

**Example 3** (Lattice FPGAs only) This is an example of **par** using the **-io** switch to generate .udb files that contain only I/O for viewing in the PAD Specification file for adjustment of ldc_set_location constraints for optimal I/O placement. You can display I/O placement assignments in the Radiant Spreadsheet View and choosing **View > Display IO Placement**.

```
par -io -w lev1bist.udb lev1bist_io.udb
```

**Using the PAR Multi-Tasking (-m) Option** This section provides information about environment setup, node list file creation, and step-by-step instructions for running the PAR Multi-tasking (**-m**) option from the command line. The PAR **-m** option allows you to use multiple machines (nodes) that are networked together for a multi-run PAR job, significantly reducing the total amount of time for completion. Before the multi-tasking option was developed, PAR could only run multiple jobs in a linear or serial fashion. The total time required to complete PAR was equal to the amount of time it took for each of the PAR jobs to run.

For example, the PAR command:

par -n 10 mydesign.udb output.udb

tells PAR to run 10 place and route passes (**-n 10**). It runs each of the 10 jobs consecutively, generating an output .udb file for each job, i.e., output_par.dir/5_1.udb, output_par.dir/5_2.udb, etc. If each job takes approximately one hour, then the run takes approximately 10 hours.

Suppose, however, that you have five nodes available. The PAR Multi-tasking option allows you to use all five nodes at the same time, dramatically reducing the time required for all ten jobs.

**To run the PAR multi-tasking option from the command line:**

1. First generate a file containing a list of the node names, one per line as in the following example:

```
# This file contains a profile node listing for a PAR multi
# tasking job.
[machine1]
SYSTEM = linux
CORENUM = 2
[machine2]
SYSTEM = linux
CORENUM = 2
Env = /home/user/setup_multipar.lin
Workdir = /home/user/myworkdir
```

   You must use the format above for the node list file and fill in all required parameters. Parameters are case insensitive.The node or machine names are given in square brackets on a single line.

   The **System** parameter can take linux or pc values depending upon your platform. However, the PC value cannot be used with Linux because it is not possible to create a multiple computer farm with PCs. **Corenum** refers to the number of CPU cores available. Setting it to zero will disable the node from being used. Setting it to a greater number than the actual number of CPUs will cause PAR to run jobs on the same CPU lengthening the runtime.

   The **Env** parameter refers to a remote environment setup file to be executed before PAR is started on the remote machine. This is optional. If the remote machine is already configured with the proper environment, this line can be omitted. To test to see if the remote environment is responsive to PAR commands, run the following:

```
ssh <remote_machine> par <par_option>
```

   See the **System Requirements** section below for details on this parameter.

   **Workdir** is the absolute path to the physical working directory location on the remote machine where PAR should be run. This item is also optional. If an account automatically changes to the proper directory after login, this line can be omitted. To test the remote directory, run the following,

```
ssh <remote_machine> ls <udb_file>
```

If the design can be found then the current directory is already available.

2. Now run the job from the command line as follows:

```
par -m nodefile_name -n 10 mydesign.udb output.udb
```

This runs the following jobs on the nodes specified.

```
Starting job 5_1 on node NODE1 at ...
Starting job 5_2 on node NODE2 at ...
Starting job 5_3 on node NODE3 at ...
Starting job 5_4 on node NODE4 at ...
 Starting job 5_5 on node NODE5 at ...
```

As the jobs finish, the remaining jobs start on the nodes until all 10 jobs are complete. Since each job takes approximately one hour, all 10 jobs will complete in approximately two hours.

**Note**

If you attempt to use the multi-tasking option and you have specified only one placement iteration, PAR will disregard the **-m** option from the command and run the job in normal PAR mode. In this case you will see the following message:

```
WARNING - par: Multi task par not needed for this job. -m
switch will be ignored.
```

**System Requirements** **ssh** must be located through the PATH variable. On Linux, the utility program's secure shell (**ssh**) and secure shell daemon (**sshd**) are used to spawn and listen for the job requests.

The executables required on the machines defined in the node list file are as follows:

- /bin/sh
- par (must be located through the PATH variable)

Required environment variable on local and remote machines are as follows:

- FOUNDRY (points at FOUNDRY directory structure must be a path accessible to both the machine from which the Implementation Engine is run and the node)
- LM_LICENSE_FILE (points to the security license server nodes)
- LD_LIBRARY_PATH (supports par path for shared libraries must be a path accessible to both the machine from which the Implementation Engine is run and the node)

To determine if everything is set up correctly, you can run the **ssh** command to the nodes to be used.

Type the following:

```
ssh <machine_name> /bin/sh -c par
```

If you get the usage message back on your screen, everything is set correctly. Note that depending upon your setup, this check my not work even though your status is fine.

If you have to set up your remote environment with the proper environment variables, you must create a remote shell environment setup file. An example of an ASCII file used to setup the remote shell environment would be as follows for ksh users:

```
export FOUNDRY=<install_directory>/ispfpga/bin/lin64
export PATH=$FOUNDRY/bin/lin64:$PATH
export LD_LIBRARY_PATH=$FOUNDRY/bin/lin:$LD_LIBRARY_PATH
64
```

For csh users, you would use the setenv command.

**Screen Output** When PAR is running multiple jobs and is not in multi-tasking mode, output from PAR is displayed on the screen as the jobs run. When PAR is running multiple jobs in multi-tasking mode, you only see information regarding the current status of the feature.

For example, when the job above is executed, the following screen output would be generated:

```
Starting job 5_1 on node NODE1
Starting job 5_2 on node NODE2
Starting job 5_3 on node NODE3
Starting job 5_4 on node NODE4
Starting job 5_5 on node NODE5
```

When one of the jobs finishes, this message will appear:

```
Finished job 5_3 on node NODE3
```

These messages continue until there are no jobs left to run.

**See Also** ▶"Implementing the Design" in the Radiant software online help

- "Command Line Data Flow" on page 563
- "Command Line Program Overview" on page 561

# Running Timing from the Command Line

The **MAP Timing** and **Place & Route Timing** processes in the Radiant software environment can also be run through the command line using the **timing** program. Timing can be run on designs using the placed and routed Unified Design Database (.udb) and associated timing constraints specified in

the design's (.ldc,.fdc, .sdc or .pdc) file or device constraints extracted from the design. Using these input files, **timing** provides static timing analysis and outputs a timing report file (.tw1/.twr).

Timing checks the delays in the Unified Design Database (.udb) file against your timing constraints. If delays are exceeded, Timing issues the appropriate timing error. See “Implementing the Design” in the Radiant software online help and associated topics for more information.

Subjects included in this topic:

- Running Timing
- Command Line Syntax
- Timing Options
- Examples

**Running Timing** Timing uses your input mapped or placed-and-routed Unified Design Database (.udb) file and associated constraint file to create a Timing Report.

- To run Timing, type **timing** on the command line with, at minimum, the names of your input .udb and sdc files to output a timing report (.twr) file. A sample of a typical Timing command would be as follows:

  timing design.udb (constraint is embedded in udb)

**Note**

The above command automatically generates the report file named design.twr which is based on the name of the .udb file.

There are several command line options that give you control over the way Timing generates timing reports for analysis. Please refer to the rest of the subjects in this topic for more details. See “Examples” on page 106.

**Command Line Syntax** **timing <udb file name> [ -sdc <sdc file name> ] [ -hld | -sethld ] [ -o <output file name> ] [ -v <integer> ] [-endpoints <integer>] [-help]**

**Timing Options** The following tables contain descriptions of all valid options for Timing.

**Table 9: Compulsory Timing Command Line Options**

| Compulsory Option | Description |
|---|---|
| **-db-file** arg | Name of input UDB file. |

**Table 10: Optional Timing Command Line Options**

| Optional Option | Description |
|---|---|
| **-endpoints** arg (=10) | number of end points. |
| **-u** arg (=10) | number of unconstrained end points printed in the table. |
| **-ports** (=10) | number of top ports printed in the table. |
| **-help** | print the usage and exit. |
| **-hld** | hold report only. |
| **-rpt-file** arg | timing report file name. |
| **-o** arg | timing report file name. |
| **-sdc-file** arg | sdc file name. |
| **-sethld** | both setup and hold report. |
| **-v** arg (=10) | number of paths per constraint. |
| **-report_sdc** | Parsed file appears in report file. |
| **-time_through_async** | Timer will time through async resets. |
| **-iotime** | compute the input setup/hold and clock to output delays of the FPGA. |
| **-nperend** arg (=1) | Number of paths per end point. |
| **-html** | HTML format report. |
| **-gui** | Call from GUI. |
| **-msg** arg | Message log file. |
| **-msgset** arg | Message setting. |

**Examples** Following are a few examples of Timing command lines and a description of what each does.

**Example 1** The following command verifies the timing characteristics of the design named design1.udb, generating a summary timing report. Timing constraints contained in the file group1.prf are the timing constraints for the design. This generates the report file design1.twr.

timing design1.udb (constraint is embedded in udb)

**Example 2** The following command produces a file listing all delay characteristics for the design named design1.udb. Timing constraints contained in the file group1.prf are the timing constraints for the design. The file output.twr is the name of the verbose report file.

timing -v design1.udb -o output.twr

**Example 3** The following command analyzes the file design1.udb and reports on the three worst errors for each constraint in timing.prf. The report is called design1.twr.

timing -e 3 design1.udb

**Example 4** The following command analyzes the file design1.udb and produces a verbose report to check on hold times on any FREQUENCY, CLOCK_TO_OUT, INPUT_SETUP and OFFSET constraints in the timing.prf file. With the output report file name unspecified here, a file using the root name of the .udb file (i.e., design1.twr) will be output by default.

timing -v -hld design1.udb

**Example 5** The following command analyzes the file design1.udb and produces a summary timing report to check on both setup and hold times on any INPUT_SETUP and CLOCK_TO_OUT timing constraints in the timing.prf file. With the output report file name unspecified here, a file using the root name of the .udb file (i.e., design1.twr) will be output by default.

timing -sethld design1.udb

**See Also** ▶"Command Line Program Overview" on page 561

▶ "Command Line Data Flow" on page 563

# Running Backannotation from the Command Line

The **Generate Timing Simulation Files** process in the Radiant software environment can also be run through the command line using the **backanno** program. The **backanno** program back-annotates physical information (e.g., net delays) to the logical design and then writes out the back-annotated design in the desired netlist format. Input to **backannobackanno** is a Unified Database file (.udb) a mapped and partially or fully placed and/or routed design.

Subjects included in this topic:

- Running Backanno
- Command Line Syntax
- Backanno Options
- Examples

**Running Backanno** backanno uses your input mapped and at least partially placed-and-routed Unified Database (.udb) file to produce a back-annotated netlist (.v) and standard delay (.sdf) file. This tool supports all FPGA design architecture flows. Only Verilog netlist is generated.

- To run backanno, type **backanno** on the command line with, at minimum, the name of your input .udb file. A sample of a typical backanno command would be as follows:

  **backanno** backanno.udb

**Note**

The above command back annotates backanno.udb and generates a Verilog file backanno.v and an SDF file backanno.sdf. If the target files already exist, they will not be overwritten in this case. You would need to specify the **-w** option to overwrite them.

There are several command line options that give you control over the way backanno generates back-annotated netlists for simulation. Please refer to the rest of the subjects in this topic for more details.

**Command Line Syntax (Verilog)** **backanno [-w] [-pre** <prfx>] **[-sp** <grade>] **[-neg] [-pos ] [-sup] [-min] [-x] [-fc ] [-slice] [-slice0 ] [-slice1] [-noslice] [-t] [-dis** [<del>]] **[-m** [<limit>]] **[-u] [-i] [-nopur] [-l** <libtype>] **[-s** <separator>] **[-o** <verilog<<.v>>] **[-d** <delays[.sdf]>] **[-gui] [-msg** <msglogfile>] **[-msgset** <msgtypefile>] [<udbfile>]

**Backanno Options** The table below contains descriptions of all valid options for backanno.

**Table 11: Backanno Options**

| Option | Description |
|---|---|
| **-w** | Overwrite the output files. |
| **-sp** <grade> | Override performance grade for backannotation. |
| **-pre** <prfx> | Prefix to add to module name to make them unique for multi-chip simulation. |
| **-min** | Override performance grade to minimum timing for hold check. |
| **-dis** <del> | Distribute routing delays by splitting the signal and inserting buffers. <del> is the maximum delay (in ps) between each buffer (1000ps by default). |
| **-m** <limit> | Shortens the block names to a given character limit in terms of some numerical integer value. |
| **-u** | Add pads for top-level dangling nets. |
| **-neg** | Negative setup/hold delay support. Without this option, all negative numbers are set to 0 in SDF. |
| **-pos** | Write out 0 for negative setup/hold time in SDF for SC. |
| **-x** | Generate x for setup/hold timing violation. |
| **-i** | Create a buffer for each block input that has interconnection delay. |

**Table 11: Backanno Options**

| Option | Description |
|---|---|
| **-nopur** | Do not write PUR instance in the backannotation netlist. Instead, user has to instantiate it in a test bench. |
| <type> | Netlist type to write out. |
| <libtype> | Library element type to use. |
| <netfile> | The name of the output netlist file. The extension on this file will change depending on which type of netlist is being written. Use -h <type>, where <type> is the output netlist type, for more specific information. |
| <udb file> | Input file '.udb '. |

**Examples** Following are a few examples of backanno command lines and a description of what each does.

**Example 1** The following command back annotates design.udb and generates a Verilog file design.vo and an SDF file design.sdf. If the target files exist, they will be overwritten.

```
backanno -w design.udb
```

**Example 2** The following command back annotates design.udb and generates a Verilog file backanno.vo and an SDF file backanno.sdf. Any signal in the design that has an interconnection delay greater than 2000 ps (2 ns) will be split and a series of buffers will be inserted. The maximum interconnection delay between each buffer would be 2000 ps.

```
backanno -dis 2000 -o backanno design.udb
```

**Example 3** The following command re-targets backannotation to peforcxmance grade -2, and puts a buffer at each block input to isolate the interconnection delay (ends at that input) and the pin to pin delay (starts from that input).

```
backanno -sp 2 -i design.udb
```

**Example 4** The following command generates Verilog netlist and SDF files without setting the negative setup/hold delays to 0:

```
backanno -neg -n verilog design.udb
```

**See Also** ▶"Command Line Program Overview" on page 561

▶ "Command Line Data Flow" on page 563

# Running Bit Generation from the Command Line

The **Bitstream** process in the Radiant software environment can also be run through the command line using the bit generation (**bitgen**) program. This topic provides syntax and option descriptions for usage of the **bitgen** program from the command line. The **bitgen** program takes a fully routed Unified Database (.udb) file as input and produces a configuration bitstream (bit images) needed for programming the target device.

Subjects included in this topic:

- Running BITGEN
- Command Line Syntax
- BITGEN Options
- Examples

**Running BITGEN** BITGEN uses your input, fully placed-and-routed Unified Database (.udb) file to produce bitstream (.bit, .msk, or .rbt) for device configuration.

- To run BITGEN, type **bitgen** on the command line with, at minimum, the **bitgen** command. There is no need to specify the input .udb file if you run **bitgen** from the directory where it resides and there is no other .udb present.

There are several command line options that give you control over the way BITGEN outputs bitstream for device configuration. Please refer to the rest of the subjects in this topic for more details.

**Command Line Syntax** **bitgen** [**-d**] [**-b**] [**-a**] [**-w**] [**-nvcm**] <infile> [<outfile>]

**BITGEN Options** The table below contains descriptions of all valid options for BITGEN.

**Note**

Many BITGEN options are only available for certain architectures. Please use the **bitgen** -h <*architecture*> help command to see a list of valid bitgen options for the particular device architecture you are targeting.

**Table 12: BITGEN Command Line Options**

| Option | Description |
|---|---|
| **-d** | Disable DRC. |
| **-b** | Produce .rbt file (ASCII form of binary). |
| **-a** | Produce .hex file. |
| **-w** | Overwrite an existing output file. |
| **-nvcm** | Produce NVCM file. |

**Table 12: BITGEN Command Line Options**

| Option | Description |
|---|---|
| **-h** <architecture> or **-help** <architecture> | Display available BITGEN command options for the specified architecture. The **bitgen -h** command with no architecture specified will display a list of valid architectures. |
| <infile> | The input post-PAR design database file (.udb). |
| <outfile> | The output file. If you do not specify an output file, BITGEN creates one in the input file's directory If you specify -b, the extension is .rbt. If you specify –a, the extension is .hex. If you specify –nvcm, the extension is .nvcm. Otherwise the extension is .bin.<br><br>A report (.bgn) file containing all of BITGEN's output is automatically created under the same directory as the output file. |

**Example** The following command tells **bitgen** to overwrite any existing bitstream files with the **-w** option, prevents a physical design rule check (DRC) from running with **-d**, specifies a raw bits (.rbt) file output with **-b**. Notice how these three options can be combined with the **-wdb** syntax.

```
bitgen -wdb PERSIST:Yes
```

**See Also** ▶"Command Line Program Overview" on page 561

▶ "Command Line Data Flow" on page 563

# Running Various Utilities from the Command Line

The command line utilities described in this section are not commonly used by command line users, but you often see them in the auto-make log when you run design processes in the Radiant software environment. Click each link below for its function, syntax, and options.

**Note**

For information on commonly-used FPGA command line tools, see "Command Line Basics" on page 563.

## Synpwrap

The **synpwrap** command line utility (wrapper) is used to manage Synplicity Synplify and Synplify Pro synthesis programs from the Radiant software environment processes: **Synplify Synthesize Verilog File** or **Synplify Synthesize VHDL File**.

The **synpwrap** utility can also be run from the command line to support a batch interface. For details on Synplify see the Radiant software online help. The **synpwrap** program drives **synplify_pro** programs with a Tcl script file containing the synthesis options and file list.

**Note**

This section supersedes the “Process Optimization and Automation” section of the *Synplicity Synplify and Synplify Pro for Lattice User Guide*.

This section illustrates the use of the **synpwrap** program to run Synplify Pro for Lattice synthesis scripts from the command line. For more information on synthesis automation of Synplify Pro, see the “User Batch Mode” section of the *Synplicity Synplify and Synplify Pro for Lattice User Guide*.

If you use Synplify Pro, the Lattice OEM license requires that the command line executables **synplify_pro** be run by the Lattice “wrapper” program, **synpwrap**.

**Command Line Syntax** **synpwrap** [**-log** <*log_file*>] [**-nolog**] [**-int** <*command_file*>] [**-gui**] [**-int** <*project_file*> | **-prj** <*project_file*>] [**-dyn**] [**-notoem**] [**-oem**] [**-notpro**] [**-pro**] [**-rem**] [**-scriptonly** <*script_file*>] **-e** <*command_file*> **-target** <*device_family*> **-part** <*device_name*> [**-options** <*arguments*>]

**Table 13: SYNPWRAP Command Line Options**

| Option | Description |
|---|---|
| **-log** <*log_file*> | Specifies the log file name. |
| **-nolog** | Does not print out the log file after the process is finished. |
| **-options** <*arguments*> | Passes all arguments to Synplify/Pro. Ignores all other options except -notoem/-oem and -notpro/-pro. The -options switch must follow all other synpwrap options. |
| **-prj** <*project_file*> | Runs Synplify or Synplify Pro using an external prj Tcl file instead of the Radiant software command file. |
| **-rem** | Does not automatically include Lattice library files. |
| **-e** <*command_file*> | Runs the batch interface based on a Radiant software generated command file. The synpwrap utility reads <*project*>.cmd with its command line to obtain user options and creates a Tcl script file. |
| **-gui** | Invokes the Synplify or Synplify Pro graphic user interface. |
| **-int** <*command_file*> | Enables the interactive mode. Runs Synplify/Pro UI with project per command file. |
| **-dyn** | Brings the Synplify installation settings in the Radiant software environment. |
| **-notoem** | Does not use the Lattice OEM version of Synplify or Synplify Pro. |

**Table 13: SYNPWRAP Command Line Options**

| Option | Description |
|---|---|
| **-oem** | Uses the Lattice OEM version of Synplify or Synplify Pro. |
| **-notpro** | Does not use the Synplify Pro version. |
| **-pro** | Uses the Synplify Pro version. |
| **-target** <*device_family*> | Specifies the device family name. |
| **-part** <*device_name*> | Specifies the device. For details on legal <*device_name*> values. |
| **-scriptonly** <*script_file*> | Generates the Tcl file for Synplify or Synplify Pro. Does not run synthesis. |

**Example** Below shows a synpwrap command line example.

```
synpwrap -rem -e prep1 -target iCE40UP
```

**See Also** ▶"Command Line Program Overview" on page 561

- "Command Line Data Flow" on page 563

# Using Command Files

This section describes how to use command files.

**Creating Command Files** The command file is an ASCII file containing command arguments, comments, and input/output file names. You can use any text editing tool to create or edit a command file, for example, **vi**, **emacs**, **Notepad**, or **Wordpad**.

Here are some guidelines when you should observe when creating command files:

- Arguments (executables and options) are separated by space and can be spread across one or more lines within the file.
- Place new lines or tabs anywhere white space would otherwise be allowed on the UNIX or DOS command line.
- Place all arguments on the same line, or one argument per line, or any combination of the two.
- There is no line length limitation within the file.
- All carriage returns and other non-printable characters are treated as space and ignored.
- Comments should be preceded with a **#** (pound sign) and go to the end of the line.

**Command File Example** This is an example of a command file:

```
#command line options for par for design mine.udb
-a -n 10
```

```
-w
-l 5
-s 2 #will save the two best results
/home/users/jimbob/designs/mine.udb
#output design name
/home/users/jimbob/designs/output.dir
#use timing constraint file
/home/users/jimbob/designs/mine.prf
```

**Using the Command File The –f Option** Use the **–f** option to execute a command file from any command line tool. The **–f** option allows you to specify the name of a command file that stores and then executes commonly used or extensive command arguments for a given FPGA command line executable tool. You can then execute these arguments at any time by entering the UNIX or DOS command line followed by the name of the file containing the arguments. This can be useful if you frequently execute the same arguments each time you perform the command, or if the command line becomes too long. This is the recommended way to get around the DOS command line length limitation of 127 characters. (Equivalent to specifying a shell Options file.)

The **–f** indicates fast startup, which is performed by not reading or executing the commands in your .cshrc | .kshrc | .shrc (C-shell, Korn-shell, Bourne-shell) file. This file typically contains your path information, your environment variable settings, and your aliases. By default, the system executes the commands in this file every time you start a shell. The **–f** option overrides this process, discarding the 'set' variables and aliases you do not need, making the process much faster. In the event you do need a few of them, you can add them to the command file script itself.

**Command File Usage Examples** You can use the command file in two ways:

- To supply all of the command arguments as in this example:

**par** -f <command_file>

where:

<command_file> is the name of the file containing the command line arguments.

- To insert certain command line arguments within the command line as in the following example:

**par** -i 33 -f placeoptions -s 4 -f routeoptions design_i.udb design_o.udb

where:

**placeoptions** is the name of a file containing placement command arguments.

**routeoptions** is the name of a file containing routing command arguments.

# Using Command Line Shell Scripts

This topic discusses the use of shell scripts to automate either parts of your design flow or entire design flows. It also provides some examples of what you can do with scripts. These scripts are UNIX-based; however, it is also possible to create similar scripts called batch files for PC but syntax will vary in the DOS environment.

**Creating Shell Scripts** A UNIX shell script is an ASCII file containing commands targeted to a particular shell that interprets and executes the commands in the file. For example, you could target Bourne Shell (**sh**), C-Shell (**csh**), or Korn Shell (**ksh**). These files also can contain comment lines that describe part of the script which then are ignored by the shell. You can use any text editing tool to create or edit a shell script, for example, **vi** or **emacs**.

Here are some guidelines when you should observe when creating shell scripts:

- It is recommended that all shell scripts with "**#!**" followed by the path and name of the target shell on the first line, for example, #!/bin/ksh. This indicates the shell to be used to interpret the script.
- It is recommended to specify a search path because oftentimes a script will fail to execute for users that have a different or incomplete search path. For example:

  PATH=/home/usr/lsmith:/usr/bin:/bin; export PATH
- Arguments (executables and options) are separated by space and can be spread across one or more lines within the file.
- Place new lines or tabs anywhere white space would otherwise be allowed on the UNIX command line.
- Place all arguments on the same line, or one argument per line, or any combination of the two.
- There is no line length limitation within the file.
- All carriage returns and other non-printable characters are treated as space and ignored.
- Comments are preceded by a **#** (pound sign) and can start anywhere on a line and continue until the end of the line.
- It is recommended to add exit status to your script, but this is not required.

```
# Does global timing meet acceptable requirement range?
    if [ $timing -lt 5  -o  $timing -gt 10 ]; then
         echo 1>&2 Timing \"$timing\" out of range
         exit 127
    fi
    etc...
    # Completed, Exit OK
    exit 0
```

**Advantages of Using Shell Scripts** Using shell scripts can be advantageous in terms of saving time for tasks that are often used, in

reducing memory usage, giving you more control over how the FPGA design flow is run, and in some cases, improving performance.

**Scripting with DOS** Scripts for the PC are referred to as batch files in the DOS environment and the common practice is to ascribe a .bat file extension to these files. Just like UNIX shell scripts, batch files are interpreted as a sequence of commands and executed. The COMMAND.COM or CMD.EXE (depending on OS) program executes these commands on a PC. Batch file commands and operators vary from their UNIX counterparts. So, if you wish to convert a shell script to a DOS batch file or vice-versa, we suggest you find a good general reference that shows command syntax equivalents of both operating systems.

**Examples** The following example shows running design “counter” on below device package

Architecture: iCE40UP

Device: iCE40UP3K

Package: UWG30

Performance: Worst Case

**Command 1: logic synthesis**

```
synthesis -f counter_impl1_lattice.synproj
     which the *.synproj contains
-a "iCE40UP"
-p iCE40UP3K
-t UWG30
-sp "Worst Case"
-optimization_goal Area
-bram_utilization 100
-ramstyle Auto
-romstyle auto
-dsp_utilization 100
-use_dsp 1
-use_carry_chain 1
-carry_chain_length 0
-force_gsr Auto
-resource_sharing 1
-propagate_constants 1
-remove_duplicate_regs 1
-mux_style Auto
-max_fanout 1000
-fsm_encoding_style Auto
-twr_paths 3
-fix_gated_clocks 1
-loop_limit 1950
-use_io_reg auto
-use_io_insertion 1
-resolve_mixed_drivers 0
-sdc "impl1.ldc"
-path "C:/lscc/radiant/1.0/ispfpga/ice40tp/data" "impl1"
-ver "C:/lscc/radiant/1.0/ip/pmi/pmi.v"
-ver "count_attr.v"
-path "."
```

```
-top count
-udb "counter_impl1.udb"
-output_hdl "counter_impl1.vm"
```

**Command 2: post synthesis process**

```
postsyn -a iCE40UP -p iCE40UP3K -t UWG30 -sp Worst Case -top -
ldc counter_impl1.ldc -keeprtl -w -o counter_impl1.udb
counter_impl1.vm
```

**Command 3: Mapper**

```
map "counter_impl1_syn.udb" "impl1.pdc" -o "counter_impl1.udb"
```

**Command 4: Placer and router**

```
par -f "counter_impl1.p2t" "counter_impl1_map.udb"
"counter_impl1.udb"
```

**Command 5: Timer**

```
timing -sethld -v 10 -u 10 -endpoints 10  -nperend 1 -html -rpt
"counter_impl1_twr.html" "counter_impl1.udb"
```

**Command 6: back annotation**

```
backanno "counter_impl1.udb" -n Verilog  -o
"counter_impl1_vo.vo"        -w -neg
```

**Command 7: bitstream generation**

```
bitgen -w "counter_impl1.udb" -f "counter_impl1.t2b"
```

# Tcl Command Reference Guide

The Radiant software supports Tcl (Tool Command Language) scripting and provides extended Radiant software Tcl commands that enable a batch capability for running tools in the Radiant software's graphical interface. The command set and the Tcl Console used to run it affords you the speed, flexibility and power to extend the range of useful tasks that the Radiant software tools are already designed to perform.

In addition to describing how to run the Radiant software's Tcl Console, this guide provides you with a reference for Tcl command line usage and syntax for all Radiant software point tools within the graphical user interface so that you can create command scripts, modify commands, or troubleshoot existing scripts.

**About the Radiant software Tcl Scripting Environment** The Radiant software development software features a powerful script language system. The user interface incorporates a complete Tcl command interpreter. The command interpreter is enhanced further with additional Radiant software-specific support commands. The combination of fundamental Tcl along with the commands specialized for use with the Radiant software allow the entire Radiant software development environment to be manipulated.

Using the command line tools permits you to do the following:

- Develop a repeatable design environment and design flow that eliminates setup errors that are common in GUI design flows
- Create test and verification scripts that allow designs to be checked for correct implementation
- Run jobs on demand automatically without user interaction

The Radiant software command interpreter provides an environment for managing your designs that are more abstract and easier to work with than using the core Radiant software engines. The Radiant software command interpreter does not prevent use of the underlying transformation tools. You

can use either the TCL commands described in this section or you can use the core engines described in the “Command Line Reference Guide” on page 561.

**Additional References** If you are unfamiliar with the Tcl language you can get help by visiting the Tcl/tk web site at http://www.tcl.tk. If you already know how to use Tcl, see the Tcl8.5/Tk 8.5 Manual supplied with this software. For information on command line syntax for running core tools that appear as Radiant software processes, such as synthesis, map, par, backanno, and timing, see the “Command Line Reference Guide” on page 561.

**See Also**
- “Running the Tcl Console” on page 599
- “Accessing Command Help in the Tcl Console” on page 600
- “Radiant Software Tool Tcl Command Syntax” on page 605
- “Creating and Running Custom Tcl Scripts” on page 601
- “Accessing Command Help in the Tcl Console” on page 600
- Tcl 8.4/Tk 8.4 Manual

# Running the Tcl Console

The Radiant software TCL Console environment is made available for your use in multiple different ways. In order to take full advantage of the FPGA development process afforded by the Radiant software you must gain access to the Radiant Tcl Console user interface.

**On Windows** In Windows 7 you can interact with the Tcl Console by any one of the following methods:

- To launch the Radiant software GUI from the Windows Start menu, choose **Start > All Programs > Lattice Radiant Software > Radiant Software**.

  After the the Radiant software loads you can click on the **TCL Console** tab. With the **TCL Console** tab active, you are able to start entering standard syntax TCL commands or the Radiant software specific support commands.
- To launch the **TCL Console** independently from the Radiant software GUI from the Windows Start menu choose **Start > All Programs > Lattice Radiant Software > Accessories > TCL Console**.

  A Windows command interpreter will be lauched that automatically runs the **TCL Console**.
- To run the interpreter from the command line, type the following:

  `c:/lscc/radiant/<version_number>/bin/nt64/pnmainc`

  The Radiant **TCL Console** is now available to run.
- To run the interpreter from a Windows 7 PowerShell from the Windows Start menu choose **Start > All Programs > Accessories > Windows PowerShell > Windows PowerShell (x86)**.

A PowerShell interpreter window will open. At the command line prompt type the following:

```
c:/lscc/radiant/<version_number>/bin/nt64/pnmainc
```

The Radiant **TCL Console** is now available to run.

**Note**

The arrangement and location of each of the programs in the Windows Start menu will differ depending on the version of Windows you are running.

**On Linux** In Linux operating systems you can interact with the Tcl Console by one of the following methods:

- To launch the Radiant software GUI from the command line, type the following:

  ```
  /usr/local/radiant/<version_number>/bin/lin64/radiant
  ```

  The path provided assumes the default installation directory and that the Radiant software is installed. After the Radiant software loads you can click on the **TCL Console** tab. With the **TCL Console** tab active, you are able to start entering standard syntax TCL commands or the Radiant software specific support commands.

- To launch the **TCL Console** independently from the Radiant software GUI from the command line, type the following:

  ```
  /usr/local/Radiant/<version_number>/bin/lin64/radiantc
  ```

  The path provided assumes the default installation directory and that the Radiant software is installed, and that you have followed the Radiant software for Linux installation procedures. The Radiant **TCL Console** is now ready to accept your input.

  The advantage of running the **TCL Console** from an independent command interpreter is the ability to directly pass the script you want to run to the Tcl interpreter. Another advantage is that you have full control over the Tk graphical environment, which allows you to create you own user interfaces.

**See Also**
- "Running the Tcl Console" on page 599
- "Radiant Software Tool Tcl Command Syntax" on page 605
- "Creating and Running Custom Tcl Scripts" on page 601
- "Accessing Command Help in the Tcl Console" on page 600
- Tcl 8.4/Tk 8.4 Manual

# Accessing Command Help in the Tcl Console

You can access command syntax help for all of the tools in the Tcl Console.

**To access command syntax help in the Tcl Console:**

1. In the prompt, type **help <tool_name>*** and press **Enter** as shown below:

   ```
   help prj*
   ```

   A list of valid command options appears in the Tcl Console. If you just type **help <tool_name>*** in the console, you will be given a list of tools.

2. As directed in the window, type the name of the command or function for more details on syntax and usage. For the prj tool, for example, type and enter the following:

   ```
   prj_open
   ```

   A list of valid arguments for that function will appear.

**Note**

Although you can run the Radiant software's core tools such as synthesis, postsyn, map, par, and timing from the Tcl Console, the syntax for accessing help is different. For proper usage and syntax for accessing help for core tools, see the "Command Line Reference Guide" on page 561.

**See Also** "Running the Tcl Console" on page 599

- "Radiant Software Tool Tcl Command Syntax" on page 605
- "Creating and Running Custom Tcl Scripts" on page 601
- "Running Tcl Scripts When Launching the Radiant Software" on page 604
- Tcl 8.4/Tk 8.4 Manual

# Creating and Running Custom Tcl Scripts

This topic describes how to easily create Tcl scripts using the Radiant software's user interface and manual methods. FPGA design using Tcl scripts provides some distinct advantages over using the graphical user interface's lists, views and menu commands. For example, Tcl scripts allow you to do the following:

- Set the tool environment to exactly the same state for every design run. This eliminates human errors caused by forgetting to manually set a critical build parameter from a drop-down menu.
- Manipulate intermediate files automatically, and consistently on every run. For example, .vm file errors can be corrected prior to performing additional netlist transformation operations.
- Run your script automatically by using job control software. This gives you the flexibilty to run jobs at any time of day or night, taking advantage of idle cycles on your corporate computer system.

**Creating Tcl Scripts** There are a couple of different methods you can use to create the Radiant software Tcl scripts. This section will discuss each one and provide step-by-step instructions for you to get started Tcl scripting repetitious Radiant software commands or entire workflows.

One method you have available is to use your favorite text editor to enter a sequence of the Radiant software Tcl commands. The syntax of each the Radiant software Tcl commands is available in later topics in this portion of the online help. This method should only be used by very experienced Radiant software Tcl command line users.

The preferred method is to let the Radiant software GUI assist you in getting the correct syntax for each Tcl command. When you interact with the Radiant software user interface each time you launch a *scriptable* process and the corresponding Radiant software Tcl command is echoed to the Tcl Console. This makes it much simpler to get the correct command line syntax for each Radiant software command. Once you have the fundamental commands executed in the correct order, you can then add additional Tcl code to perform error checking, or customization steps.

**To create a Tcl command script in the Radiant software:**

1. Start the Radiant software design software and close any project that may be open.
2. In the Tcl Console execute the custom **reset** command. This clears the Tcl Console command history.
3. Use the Radiant software graphical user interface to start capturing the basic command sequence. The Tcl Console echos the commands in its window. Start by opening the project for which you wish to create the TCL script. Then click on the processes in the Process list to run them. For example, run these processes in their chronological order in the design flow:
   - Synthesize Design
   - Map Design
   - Place & Route Design
   - Export Files
4. In the Tcl Console window enter the command,

   ```
   save_script <filename.ext>
   ```

   The <filename.ext> is any file identifier that has no spaces and contains no special characters except underscores. For example, **myscript.tcl** or **design_flow_1.tcl** are acceptable save_script values, but **my$script** or **my script** are invalid. The **<filename.ext>** entry can be preceded with a absolute or relative path. Use the **"/"** (i.e. forward slash) character to delimit the path elements. If the path is not specified explicitly the script is saved in the current working directory. The current working directory can be determined by using the TCL *pwd* command.
5. You can now use your favorite text editor to make any changes to the script you feel are necessary. Start your text editor, navigate to the

directory the *save_script* command saved the base script, and open the file.

**Note**

In most all cases, you will have to clean up the script you saved and remove any invalid arguments or any commands that cannot be performed in the Radiant software environment due to some conflict or exception. You will likely have to revisit this step later if after running your script you experience any run errors due to syntax errors or technology exceptions.

**Sample Radiant software Tcl Script** The following the Radiant software Tcl script shows a very simple script that opens a project, runs the entire design flow through the Place & Route process, then closes the project. A typical script will contain more tasks and will check for failure conditions. Use this simple example as a general guideline.

**Figure 1: Simple Radiant software Script**

```
prj_open "C:/lscc/radiant/examples/counter/counter.rdf"
prj_run_par
prj_close
```

**Running Tcl Scripts** The Radiant software TCL scripts are run exclusively from the Radiant TCL Console. You can use either the TCL Console integrated into the Radiant software UI, or by launching the stand-alone TCL Console.

**To run a Tcl script in the Radiant software:**

1. Launch the Radiant software GUI, or the stand-alone TCL Console.

   Open the Radiant software but do not open your project. If your project is open, choose **File > Close Project**.

2. If you are using the Radiant software main window, click the small arrow pane switch in the bottom of the Radiant software main window, and then click on the **Tcl Console tab** in the Output area at the bottom to open the console.

3. Use the TCL *source* command to load and run your TCL script. The *source* command requires, as it's only argument, the filename of the script you want to load and run. Prefix the script file name with any required relative or absolute path information. To run the example script shown in the previous section use:

   ```
   source C:/lscc/radiant/<version_number>/examples/counter/
   myscript2.tcl
   ```

   As long as there are no syntax errors or invalid arguments, the script will open the project, synthesize, map, and place-and-route the design. Once the design finishes it closes the project. If there are errors in the script, you will see the errors in red in the Tcl Console after you attempt to run it. Go back to your script and correct the errors that prevented the script from running.

**See Also** ▶"Running the Tcl Console" on page 599

- "Radiant Software Tool Tcl Command Syntax" on page 605
- "Running Tcl Scripts When Launching the Radiant Software" on page 604
- Tcl 8.4/Tk 8.4 Manual

# Running Tcl Scripts When Launching the Radiant Software

This topic describes how launch the Radiant software and automatically run Tcl scripts using a command line shell or the stand-alone Tcl console. Your Tcl script can be standard Tcl commands as well as the Radiant software-specific Tcl commands.

Refer to "Creating and Running Custom Tcl Scripts" on page 601 for more information on creating custom Tcl scripts.

**To launch the Radiant software and run a Tcl script from a command line shell or the stand-alone Tcl console:**

- Enter the following command:

  On Windows:

  ```
  pnmain.exe -t <tcl_path_file>
  ```

  On Linux:

  ```
  radiant -t <tcl_path_file>
  ```

**Sample Radiant software Tcl Script** The following Radiant software Tcl script shows a very simple script, running in Windows, that opens a project and runs the design flow through the MAP process. Use this simple example as a general guideline.

**Figure 2: Simple Radiant Software Script**

```
prj_open C:/test/iobasic_radiant/io1.rdf
prj_run_map
```

The above example is saved in Windows as the file mytcl.tcl in the directory C:/test. By running the following command from either a DOS shell or the Tcl console in Windows, the Radiant software GUI starts, the project i01.rdf opens, and the MAP process automatically runs.

```
pnmain.exe -t c:/test/mytcl.tcl
```

**See Also**
- "Running the Tcl Console" on page 599
- "Radiant Software Tool Tcl Command Syntax" on page 605
- "Creating and Running Custom Tcl Scripts" on page 601

- Tcl 8.4/Tk 8.4 Manual

# Radiant Software Tool Tcl Command Syntax

This part of the Tcl Command Reference Guide introduces the syntax of each of the Radiant software tools and provides you with examples to help you construct your own commands and scripts.

The Radiant software tries to make it easy to develop TCL scripts by mirroring the correct command syntax in the Tcl Console based on the actions performed by you in the GUI. This process works well for most designs, but there are times when a greater degree of control is required. More control over the build process is made available through additional command line switches. The additional switches may not be invoked by actions taken by you when using the GUI. This section provides additional information about all of the Tcl commands implemented in the Radiant software.

The Tcl Commands are broken into major categories. The major categories are:

- Radiant Software Tcl Console Commands
- Radiant Software Project Tcl Commands
- Reveal Inserter Tcl Commands
- Reveal Analyzer Tcl Commands
- Power Calculator Tcl Commands
- Programmer Tcl Commands

## Radiant Software Tcl Console Commands

The Radiant software Tcl Console provides a small number of commands that allow you to perform some basic actions upon the Tcl Console Pane. The Radiant software Tcl Console commands differ from the other Tcl commands provided in the Radiant software. This dtc program's general Tcl Console commands do not use the *dtc_* prefix in the command syntax as is the convention with other tools in the Radiant software.

**Note**

TCL Command Log is always listed after the project is closed. You can find it in the Reports section under Misc Report > TCL Command Log.

**Radiant Software Tcl Console Command Descriptions** The following table provides a listing of all valid Radiant software Tcl Console-related commands.

**Table 14: Radiant Software Tcl Console Commands**

| Command | Arguments | Description |
|---|---|---|
| **history** | N/A | The *history* command lists the command history in the Tcl Console that you executed in the current session.<br>Every command entered into the Tcl Console, either by the GUI, or by direct entry in the Tcl Console, is recorded so that it can be recalled at any time.<br>The command history list is cleared when a project is *opened* or when the Tcl Console *reset* command is executed. |
| **reset** | N/A | The *reset* command clears anything present in the Tcl Console pane, and erases all entries in the command line history.<br>**It's only used in GUI Tcl console and not supported in stand-alone Tcl console. |
| **clear** | N/A | The *clear* command erases anything present in the Tcl Console pane, and prints the current *prompt* character in the upper left corner of the Tcl Console pane without erasing the command history.<br>**It's only used in GUI Tcl console and not supported in stand-alone Tcl console. |
| **save_script** | <filename.ext> | Saves the contents of the command line history memory buffer into the script file specified. The script is, by default, stored into the current working directory. File paths using forward slashes used with an identifier are valid if using an absolute file path to an existing script folder.<br>**It's only used in GUI Tcl console and not supported in stand-alone Tcl console. |

**Table 14: Radiant Software Tcl Console Commands**

| Command | Arguments | Description |
|---|---|---|
| **set_prompt** | <new_character> | The default prompt character in the Tcl Console is the “greater than” symbol or angle bracket (i.e., >). You can change this prompt character to some other special character such as a dollar sign ($) or number symbol (#) if you prefer.<br>**It’s only used in GUI Tcl console and not supported in stand-alone Tcl console. |
| **create_clock** | **create_clock** -period <period_value> [-name <clock_name>] [-waveform <edge_list>] [<port_list\|pin_list\|net_list>] | Create a named or virtual clock. |
| **create_generated_clock** | **create_generated_clock** [-name <clock_name>] -source <master_pin> [-edges <edge_list>] [-divide_by <factor>]<br>[-multiply_by <factor>] [-duty_cycle <percent>] [-invert] <pin_list\|net_list\|port_list> | Create a generated clock object. |
| **set_clock_groups** | **set_clock_groups** -group <clock_list> <-logically_exclusive \| -physically_exclusive \| -asynchronous> | Set clock groups. |
| **set_clock_latency** | **set_clock_latency** [-rise] [-fall] [-early \| -late] <-source> <latency> <object_list> | Defines a clock's source or network latency |
| **set_clock_uncertainty** | **set_clock_uncertainty** [-setup] [-hold] [-from <clock>] [-to <clock>] <uncertainty> [<clock_list>] | Set clock uncertainty. |
| **set_false_path** | **set_false_path** [-from <port_list\|pin_list\|instance_list\|net_list\|clock_list>]<br>[-to <port_list\|pin_list\|instance_list\|net_list\|clock_list>]<br>[-through <port_list\|pin_list\|instance_list\|net_list>]<br>[-rise_from <clock_list>] [-rise_to <clock_list>]<br>[-fall_from <clock_list>] [-fall_to <clock_list>]<br>[-comment string] | Define false path |
| **set_hierarchy_separator** | **set_hierarchy_separator** <separator> | Set or get hierarchy separator |

**Table 14: Radiant Software Tcl Console Commands**

| Command | Arguments | Description |
|---|---|---|
| **set_input_delay** | **set_input_delay** -clock <clock_name> [-clock_fall] [-max] [-min] [-add_delay] <delay_value> <port_list> | Set input delay on ports |
| **set_max_delay** | **set_max_delay** [-from <port_list\|pin_list\|instance_list\|net_list\|clock_list>]<br>[-to <port_list\|pin_list\|instance_list\|net_list\|clock_list>]<br>[-through <port_list\|pin_list\|instance_list\|net_list>]<br>[-rise_from <clock_list>] [-rise_to <clock_list>]<br>[-fall_from <clock_list>] [-fall_to <clock_list>] <delay_value><br>[-comment string] | Specify maximum delay for timing paths |
| **set_min_delay** | **set_min_delay** [-from <port_list\|pin_list\|instance_list\|net_list\|clock_list>]<br>[-to <port_list\|pin_list\|instance_list\|net_list\|clock_list>]<br>[-through <port_list\|pin_list\|instance_list\|net_list>]<br>[-rise_from <clock_list>] [-rise_to <clock_list>]<br>[-fall_from <clock_list>] [-fall_to <clock_list>] <delay_value> | Specify maximum delay for timing paths |
| **set_multicycle_path** | **set_multicycle_path** [-from <port_list\|pin_list\|instance_list\|net_list\|clock_list>]<br>[-to <port_list\|pin_list\|instance_list\|net_list\|clock_list>]<br>[-through <port_list\|pin_list\|instance_list\|net_list>]<br>[-rise_from <clock_list>] [-rise_to <clock_list>]<br>[-fall_from <clock_list>] [-fall_to <clock_list>]<br>[-setup\|-hold] [-start\|-end] <path_multiplier> | Define multicycle path |

**Table 14: Radiant Software Tcl Console Commands**

| Command | Arguments | Description |
|---|---|---|
| **set_output_delay** | **set_output_delay** -clock <clock_name> [-clock_fall] [-max] [-min] [-add_delay] <delay_value> <port_list> | Set output delay on ports |
| **ldc_create_group** | l**dc_create_group** -name <group_name> [-bbox {height width}] <objects> | Defines a single identifier that refers to a group of objects |
| **ldc_create_region** | l**dc_create_region** -name <region_name> -site <site> -width <width> -height <height> | Define a rectangular area |
| **ldc_create_vref** | **ldc_create_vref** -name <vref_name> -site <site_name> | Define a voltage reference |
| **ldc_set_location** | **ldc_set_location** [-site <site_name>] [-bank <bank_num>] [-region <region_name>] <object> | Set object location |
| **ldc_set_vcc** | **ldc_set_vcc** [-bank bank\|-core] [-derate derate] [voltage] | Sets the voltage and/or derate for the bank or core |
| **ldc_set_port** | **ldc_set_port** [-iobuf [-vref <vref_name>]]\|[-sso] <key-value list> <ports> | Set port constraint attributes |
| **ldc_set_sysconfig** | **ldc_set_sysconfig** <key-value list> | Set sysconfig attributes |
| **ldc_set_attribute** | **ldc_set_attribute** <key-value list> [objects] | Set object attributes |
| **ldc_prohibit** | **ldc_prohibit** -site <site> | Probibit site |

**Radiant Software Tcl Console Command Examples** This section illustrates and describes a few samples of Radiant Tcl Console commands.

**Example 1** To save a script, you simply use the **save_script** command in the Tcl Console window with a name or file path/name argument. In the first example command line, the file path is absolute, that is, it includes the entire path. Here you are saving “myscript.tcl” to the existing current working directory. The second example creates the same “myscript.tcl” file in the current working directory.

```
save_script C:/lscc/radiant/myproject/scripts/myscript.tcl

save_script myscript.tcl
```

See “Creating and Running Custom Tcl Scripts” on page 601 for details on how to save and run scripts in the Radiant software.

**Example 2** The following **set_prompt** command reassigns the prompt symbol on the command line as a dollar sign ($). The default is an angle bracket or “greater than” sign (>).

```
set_prompt $
```

**Example 3** The following **history** command will print all of the command history that was recorded in the current Tcl Console session.

```
history
```

# Radiant Software Project Tcl Commands

The Radiant software Project Tcl Commands allow you to control the contents and settings applied to the tools, and source associated with your design. Projects can be opened, closed, and configured to a consistent state using the commands described in this section.

**Radiant Software Project Tcl Command Descriptions** The following table provides a listing of all valid Radiant software project-related Tcl command options and describes option functionality.

**Table 15: Radiant Software Project Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **prj_create** | **prj_create** -name <project name> [-dev <device name>] [-performance <performance grade>] [-impl <initial implementation name>] [impl_dir <initial implementation directory>] [-synthesis <synthesis tool name>] | Creates a new project inside the current working directory. The *new* command can only be used when no other project is currently open.<br>The -name <project name> argument specifies the name of the project. This creates a <project name>.rdf file in the current working directory.<br>The -impl <initial implementation name> argument specifies the active implementation when the project is created. If this left unspecified a default implemenation called “Implemenation0” is created.<br>The -dev <device name> argument specifies the FPGA family, density, footprint, performance grade, and temperature grade to generate designs for. Use the Lattice OPN (Ordering Part Number) for the <device name> argument.<br>The -performance <performance grade> argument specifies the device performance grade explicitly. For iCE40UP device, performance grade can’t be inferred from the device part name such as iCE40UP3K-UWG30ITR. If no performance grade specified, default performance value is used.<br>The -impl_dir <initial implemenation directory> argument defines the directory where temporary files are stored. If this is not specified the current working directory is used. |
| **prj_close** | **prj_close** | Exits the current project. Any unsaved changes are discarded. |
| **prj_open** | **prj_open** <projectfile.rdf> | Opens the specified project in the software environment. |
| **prj_save** | **prj_save** [projectfile.rdf] | Updates the project with all changes made during the current session and the project file is saved. |
| **prj_saveas** | **prj_saveas** -name <new project name> -dir <new project directory> [-copy_gen] | Save the current project as a new project with specified name and directory. |

**Table 15: Radiant Software Project Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **prj_set_opt** | **prj_set_opt**<br>: List all the options in the current project<br>**prj_set_opt** <option name> [option value list]<br>: List or set the option value<br>**prj_set_opt** -append <option name> <option value><br>: Append a value to the specified option value<br>**prj_set_opt** -rem <option name>...<br>: Remove the options of the current project | List, set or remove a project option. |
| **prj_archive** | **prj_archive** [-includeAll] <archive_file><br>: Archive the current project into the archive_file<br>**prj_archive** -extract -dir <destination directory> <archive_file><br>: Extract the archive file and load the project | Archive the current project. |
| **prj_set_device** | **prj_set_device** [-family <family name>] [-device <device name>]<br>[-package <package name>] [-performance <performance grade>]<br>[-operation <operation>] [-part <part name>]<br>: Change the device to the specified family, device, package, performance,<br>operation, part | Set the device. |

**Table 15: Radiant Software Project Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **prj_add_source** | **prj_add_source** [-impl <implement name>] [-simulate_only\|-synthesis_only]<br>[-include <path list for Verilog include search path>]<br>[-work <VHDL lib name>] [[-opt <name=value>] ...] [-exclude]<br><src file>... | Adds a VHDL source file to the specified or active implementation. The syntax used for the Add function depends upon the source file's implementation language.<br>[-work <VHDL lib name>]: Assigns the source code to the specified library name space.<br>[-impl <implementation name>]: This switch is used to add a source file to a Radiant software implementation. If this switch is not specified the source file is added to the active implementation.<br>[-opt name=value]: The -opt argument allows you to set a custom, user-defined option. See Example 7 for guidelines and usage.<br><sre file>...: One or more VHDL source files to add to the specified implementation. |
| **prj_enable_source** | **prj_enable_source** [-impl <implement name>] <src file> ... | Enables the excluded design sources from the current project, that is, it will activate a source file for synthesis, to be used as a constraint, or for Reveal debugging. |
| **prj_disable_source** | **prj_disable_source** [-impl <implement name>] <src file> ... | Disables the excluded design sources from the current project, that is, it will activate a source file for synthesis, to be used as a constraint or for Reveal debugging. |
| **prj_remove_source** | **prj_remove_source** [-impl <implement name>] -all<br>:Remove all the design sources in project<br>**prj_remove_source** [-impl <implement name>] <src file> ... | Deletes the specified source files from the specified implementation. If an implementation is not listed explicitly the source files are removed from the active implementation. The source files are not removed from the file system, they are only removed from consideration in the specified implementation. |

**Table 15: Radiant Software Project Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **prj_set_source_opt** | **prj_set_source_opt** -src <source name> [-impl <implement name>]<br>: List all the options in the specified source<br>**prj_set_source_opt** -src <source name> [-impl <implement name>]<br><option name> [option value list]<br>: List or set the source's option value<br>**prj_set_source_opt** -src <source name> [-impl <implement name>]<br>-append <option name> <option value><br>: Append a value to the specified option value<br>**prj_set_source_opt** -src <source name> [-impl <implement name>]<br>-rem <option name>...<br>: Remove the options of the source | List, set or remove a source option. |
| **prj_syn_sim_source** | **prj_syn_sim_source** [-impl <implement name>] -src <source name> [SimulateOnly\|SynthesisOnly\|SynthesisAndSimulate] | Return or change the setting of a design HDL source as a synthesis or simulation source. |
| **prj_create_impl** | **prj_create_impl** <new impl name> [-dir <implementation directory>] [-strategy <default strategy name>] [-synthesis <synthesis tool name>] | Create a new implementation in the current project with '<new impl name>'.<br>The new implementation will use the current active implementation's strategy as the default strategy if no valid strategy is set. |
| **prj_remove_impl** | **prj_remove_impl** <implement name> | Delete the specified implementation in the current project with '< impl name>'. |

**Table 15: Radiant Software Project Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **prj_set_impl_opt** | **prj_set_impl_opt** [-impl <implement name>]<br>: List all the options in the specified implementation<br>**prj_set_impl_opt** [-impl <implement name>] <option name> [option value list]<br>: List or set the implementation's option value<br>**prj_set_impl_opt** [-impl <implement name>] -append <option name> <option value><br>: Append a value to the specified option value<br>**prj_set_impl_opt** [-impl <implement name>] -rem <option name>...<br>: Remove the the options in the implementation | Allows you to add, list, or remove implementation options with the name <implement name> in the specified or active implementation of the current project.<br>If the -rem option is used, the following option names appearing after it will be removed.<br>If no argument is used (i.e., “prj_impl option”), the default is to list all implementation options.<br>If only the <option name> argument is used (i.e., “prj_impl option <option name>), then the value of that option in the project will be returned.<br>The command will set the option value to the option specified by <option name>. If the <option value> is empty then the option will be removed and ignored (e.g., prj_impl option -rem).<br>The -run_flow argument allows you to switch from the normal mode to an “initial” incremental flow mode and “incremental” which is the mode you should be in after an intial design run during the incremental design flow. With no value parameters specified, -run_flow will return the current mode setting. |
| **prj_activate_impl** | **prj_activate_impl** <implement name> | Activates the implementation with the name <implement name>. |
| **prj_clean_impl** | **prj_clean_impl** [-impl <implement name>] | Clean up the implementation result files in the current project. |
| **prj_clone_impl** | **prj_clone_impl** <new impl name> [-dir <new impl directory>] [-copyRef] [-impl <original impl name>] | Clone an existing implementation. |
| **prj_run_synthesis** | **prj_run_synthesis** | Run synthesis process. |
| **prj_run_map** | **prj_run_map** | Run map process. |
| **prj_run_par** | **prj_run_par** | Run par process. |
| **prj_run_bitstream** | **prj_run_bitstream** | Run bitstream process. |
| **prj_create_strategy** | **prj_create_strategy** -name <new strategy name> -file <strategy file name> | Create a new strategy with default setting. |
| **prj_remove_strategy** | **prj_remove_strategy** <strategy name> | Deletes an existing strategy. |

**Table 15: Radiant Software Project Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **prj_copy_strategy** | **prj_copy_strategy** -from <source strategy name> -name <new strategy name> -file <strategy file name> | Copies an existing strategy and saves it to a newly created strategy file. |
| **prj_import_strategy** | **prj_import_strategy** -name <new strategy name> -file <strategy file name> | Import an existing strategy file. |
| **prj_set_strategy** | **prj_set_strategy** [-impl <implementation name>] <strategy name> | Associate the strategy with the specified implementation. |
| **prj_list_strategy** | **prj_list_strategy** [-strategy <strategy name>] <pattern> | List value to a strategy item. |
| **prj_import_diamond** | **prj_import_diamond** [-dev <target device>] [-performance <performance grade>] -save_to <new Radiant project file> [-copy_src] <Dimaond project file> | Imports an existing Diamond project file to a new Radiant software project file.<br>The -dev <target device> argument specifies the FPGA family, density, footprint, performance grade, and temperature grade to generate designs for. Use the Lattice OPN (Ordering Part Number) for the <device name> argument.<br>The -performance <performance grade> argument specifies the device performance grade explicitly. For iCE40UP device, performance grade can't be inferred from the device part name such as iCE40UP3K-UWG30ITR. If no performance grade specified, default performance value is used.<br>-save_to <new Radiant project file> argument defines new Radiant software project file. If this is not specified the new project file will be created.<br>-copy_src] <Dimaond project file> argument defines where a Diamond project file is copied from. |

**Radiant Software Project Tcl Command Examples** This section illustrates and describes a few samples of Radiant software Project Tcl commands.

**Example 1** To create a new project, your command may appear something like the following which shows the creation of a ThunderPlus device.

```
prj_create -name "m" -impl "m" -dev iCE40UP3K-UWG30ITR
```

**Example 2** To save a project and give it a certain name (save as), use the project save command as shown below:

```
prj_save "my_project"
```

To simply save the current project just use the save function with no values:

```
prj_save
```

**Example 3** To open an existing project, the command syntax would appear with the absolute file path on your system as shown in the following example:

```
prj_open "C:/projects/radiant/adder/my_project.rdf"
```

**Example 4** To add a source file, in this case a source LDC file, use the prj_src add command as shown below and specify the complete file path:

```
prj_add_source "C:/my_project/radiant/counter/counter.ldc"
```

**Example 5** The following examples below shows the prj_run command being used:

```
prj_run_par
```

In this final example, synthesis is run.

```
prj_run_synthesis
```

**Example 6** To copy another project strategy that is already established in another Radiant software project from your console, use the prj_copy_strategy copy command as shown below and specify the new strategy name and the strategy file name.

```
prj_copy_strategy -from source_strategy -name new_strategy -
file strategy.stg
```

**Example 7** The prj_add_source command allows you to set a custom, user-defined option. This -opt argument value, however, cannot conflict with existing options already in the system, that is, its identifier must differ from system commands such as "include" and "lib" for example. In addition, a user-defined option may not affect the internal flow but can be queried for any usage in a user's script to arrange their design and sources. All user-defined options can be written to the Radiant software project RDF file.

In the example below, the -opt argument is used as a qualifier to make a distinction between to .rvl file test cases.

```
prj_add_source test1.rvl -opt "debug_case=golden_case"
prj_add_source test2.rvl -opt "debug_case=bad_case"
```

**Example 8** After you modify your strategy settings in the Radiant software interface the values are saved to the current setting via a Tcl command. For example, a command similar to the following will be called if Synplify frequency and area options are changed.

```
prj_set_strategy_value -strategy strategy1 SYN_Frequency=300
SYN_Area=False
```

# Simulation Libraries Compilation Tcl Command

This section provides Simulation Libraries Compilation extended Tcl command syntax and usage examples.

**Simulation Libraries Compilation Tcl Command Descriptions** The following table provides a listing of all valid Simulation Libraries Compilation Tcl Command arguments and describes their usage.

**Table 16: Simulation Libraries Compilation Tcl Command**

| Command | Function (Argument) | Description |
|---|---|---|
| **cmpl_libs** | **-sim_path** <sim_path><br>[-sim_vendor {mentor<default>}]<br>[-lang {verilog\|all<default>}]<br>[-device {ice40up\|all<default>}]<br>[-target_path <target_path>] | The -sim_path argument specifies the path to the simulation tool executable (binary) folder. This option is mandatory. Currently only Modelsim and Questa simulators are supported. NOTE: If you are a Windows user and prefer the \ notation in the path, you must surround it with {}. And "" or {} will be needed if the path has spaces.<br><br>NOTE: To execute this command error free, Questasim 10.4e or a later 10.4 version, or Questasim 10.5b or a later version should be used for compilation.<br><br>The -sim_vendor argument is optional, and intended for future use. It currently supports only Mentor Graphics simulators (Modelsim / Questa).<br><br>The -device argument specifies the Lattice FPGA device to compile simulation libraries for. This argument is optional, and the default is to compile libraries for all the Lattice FPGA devices.<br><br>The -target_path argument specifies the target path, where you want the compiled libraries and modelsim.ini file to be located. This argument is optional, and the default target path is the current folder. NOTES: (1) This argument is recommended if you did not change the current folder from the Radiant software startup (binary) folder, or if the current folder is write-protected. (2) If you are a Windows user and prefer the \ notation in the path, you must surround it with {}. And "" or {} will be needed if the path has spaces. |

**Simulation Libraries Compilation Tcl Command Examples** This section illustrates and describes a few examples of Simulation Libraries Compilation Tcl command.

**Example 1** The following command will compile all the Lattice FPGA libraries for both Verilog and VHDL simulation, and place them under the folder specified by -target_path. The path to Modelsim is specified by -sim_path.

```
cmpl_libs -sim_path C:/questasim64_10.4e/win64 -target_path c:/
mti_libs
```

# Reveal Inserter Tcl Commands

This section provides Reveal Inserter extended Tcl command syntax, command options, and usage examples.

**Reveal Inserter Tcl Command Descriptions** The following table provides a listing of all valid Reveal Inserter Tcl command options and describes option functionality.

**Table 17: Reveal Inserter Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **rvl_new_project** | **rvl_new_project**<rvl file> | Create a new reveal inserter project. |
| **rvl_open_project** | **rvl_open_project** <rvl file> | Open a reveal inserter project file. |
| **rvl_save_project** | **rvl_save_project** <rvl file> | Save the current reveal inserter project. |
| **rvl_close_project** | **rvl_close_project** | Close the current reveal inserter project. |
| **rvl_run_project** | **rvl_run_project** [-save] [-saveAs <file>] [-overwrite] [-drc] [-insert_core <core_name>] | ▶ "Run inserting debug core task or DRC checking on the current reveal inserter project<br>▶ -save: Save the project before run command<br>▶ -saveAs: Save as a different file before run command<br>▶ -overwrite: Overwrite the existing file if the saved as to file exists already<br>▶ -drc: Run DRC checking only<br>▶ -insert_core: Specify the core to be inserted. All cores will be inserted if none is specified" |
| **rvl_add_core** | **rvl_add_core** <core name> | Add a new core in current project. |
| **rvl_del_core** | **rvl_del_core** <core name> | Remove an existing core from current project. |

**Table 17: Reveal Inserter Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **rvl_rename_core** | **rvl_rename_core** <core name> <new core name> | Rename an existing core from current project. |
| **rvl_set_core** | **rvl_set_core** [core name] | List the default core or select a core as the default core in current project. |
| **rvl_list_core** | **rvl_list_core** | List all cores in current project. |
| **rvl_add_serdes** | **rvl_add_serdes** | Add the Serdes core into current project. |
| **rvl_del_serdes** | **rvl_del_serdes** | Remove the Serdes core from current project. |
| **rvl_set_serdes** | **rvl_set_serdes** [clk=<clock name>] [rst=<reset signal, default value is VLO>] | List or set options of Serdes core. |
| **rvl_add_trace** | **rvl_add_trace** [-core <core name>] [-insert_at <position>] <signals list> | Add trace signals in a debug core in current project. You can specify an existing trace signal/bus name or a postion number in a trace bus as the inserting position. |
| **rvl_del_trace** | **rvl_del_trace** [-core <core name>] <signals list> | Delete trace signals in a debug core in current project. |
| **rvl_rename_trace** | **rvl_rename_trace** [-core <core name>] -bus <bus name> <new bus name> | Change the name of a trace bus in a debug core in current project. |
| **rvl_list_trace** | **rvl_list_trace** [-core <core name>] | List all trace signals in a debug core in current project. |
| **rvl_move_trace** | **rvl_move_trace** [-core <core name>] [-move_to <position>] <signals list> | Move and rearrange the order of trace signals in a debug core in current project.<br>You can specify an existing trace signal/bus name or a postion number in a trace bus as the new position. |
| **rvl_group_trace** | **rvl_group_trace** [-core <core name>] -bus <bus name> <signals list> | Group specified trace signals in a debug core in current project into a bus. |
| **rvl_ungroup_trace** | **rvl_ungroup_trace** [-core <core name>] <bus name> | Ungroup trace signals in a trace bus in a debug core in current project. |
| **rvl_set_traceoptn** | **rvl_set_traceoptn** [-core <core name>] [option=value] | List or set trace options of a debug core in current project.<br>You can set the following option:<br>SampleClk = [signal name]. |

**Table 17: Reveal Inserter Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **rvl_set_trigoptn** | **rvl_set_trigoptn** [-core <core name>] [option=value] | List or set trigger options of a debug core in current project.<br>You can set the following option:<br>DefaultRadix = [bin\|oct\|dec\|hex]<br>EventCounter = [on\|off]<br>CounterValue = [2,4,8,16,...,65536] (depend on FinalCounter is on)<br>TriggerOut = [on\|off]<br>OutNetType = [IO\|NET\|BOTH] (depend on TriggerOut is on)<br>OutNetName = [net name] (depend on TriggerOut is on)<br>OutNetPri = [Active_Low\|Active_High] (depend on TriggerOut is on)<br>OutNetMPW = [pulse number] (depend on TriggerOut is on). |
| **rvl_list_tu** | **rvl_list_tu** [-core <core name>] | List all trigger units in a debug core in current project. |
| **rvl_add_tu** | **rvl_add_tu** [-core <core name>] [-radix <bin\|oct\|dec\|hex>] [-name <new TU name>] <TU definition> | Add a new trigger unit to a debug core in current project.<br>TU definition format: "{signal list} Operator Value"<br>Operator must be "==", "!=", ">", ">=", "<", "<=",<br>".RE."(rising edge), ".FE."(falling edge) and ".SC."(serial compare).<br>A default trigger unit name will be created if it's omitted in command.. |
| **rvl_del_tu** | **rvl_del_tu** [-core <core name>] <TU name> | Remove an existing core from current project. |
| **rvl_rename_tu** | **rvl_rename_tu** [-core <core name>] <old name> <new name> | Rename an existing core in current project. |
| **rvl_set_tu** | **rvl_set_tu** [-core <core name>] [-radix <bin\|oct\|dec\|hex>] -name <TU name><br>[-add_sig <signal list>] [-del_sig <signal list>] [-set_sig <signal list>]<br>[-expr <TU definition>] [-op operator] [-val value] | Set a trigger unit in a debug core in current project.<br>TU definition format: "{signal list} Operator Value"<br>Operator must be "==", "!=", ">", ">=", "<", "<=",<br>".RE."(rising edge), ".FE."(falling edge) and ".SC."(serial compare).. |
| **rvl_list_te** | **rvl_list_te** [-core <core name>] | List all trigger expressions in a debug core in current project. |

**Table 17: Reveal Inserter Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **rvl_add_te** | **rvl_add_te** [-core <core name>] [-ram <EBR\|Slice>] [-name <new TE name>] [-expression <expression string>] [-max_seq_depth <max depth>] [-max_event_count <max event count> | Add a new trigger expression to a debug core in current project.<br>A default trigger expression name will be created if it's omitted in command. |
| **rvl_del_te** | **rvl_del_te** [-core <core name>] <TE name> | Delete an existing trigger expression in a debug core in current project. |
| **rvl_rename_te** | **rvl_rename_te** [-core <core name>] <old name> <new name> | Rename an existing trigger expression in a debug core in current project. |
| **rvl_set_te** | **rvl_set_te** [-core <core name>] [-ram <EBR\|Slice>] [-expression <expression string>] [-max_seq_depth <max depth>] [-max_event_count <max event count>] <TE name> | Change an existing trigger expression in a debug core in current project. |

**Reveal Inserter Tcl Command Examples** This section illustrates and describes a few samples of Reveal Inserter Tcl commands.

**Example 1** To create a new Reveal Inserter project with the .rvl file extension in your project directory, use the rvl_project command as shown below using the new option.

```
rvl_new_project my_project.rvl
```

**Example 2** The following example shows how to set up TU parameters for Reveal Inserter:

```
rvl_set_tu -name TU -add_sig {count[7:0]} -op == -val C3 -radix
Hex
```

# Reveal Analyzer Tcl Commands

This section provides Reveal Analyzer extended Tcl command syntax, command options, and usage examples.

**Reveal Analyzer Tcl Command Descriptions** The following table provides a listing of all valid Reveal Analyzer Tcl command options and describes option functionality.

**Table 18: Reveal Analyzer Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **rva_new_project** | **rva_new_project** <file> | Create a new Reveal Analyzer project. |
| **rva_open_project** | **rva_open_project** <file> | Open a Reveal Analyzer project file. |
| **rva_save_project** | **rva_save_project** <file> | Save the current Reveal Analyzer project. |
| **rva_close_project** | **rva_close_project** <file> | Close the current Reveal Analyzer project. |
| **rva_export_project** | **rva_export_project** -vcd <file name> [-module <title>] | Export VCD file. Optional to include a title in the VCD file.<br>By default the title will be “<unknown>”. |
| | **rva_export_project** -txt <file name> [-siglist <signal list>] | Export TEXT file. Optional to export selected signal list only.<br>By default all signals are exported. |
| **rva_set_project** | **rva_set_project** [-frequency <val] [-period <val>] [-tckdelay <val>] [-cabletype <val>] [?cableport <val>] | No arguments specified will return options.<br>-frequency: sets the frequency value for sample clock in MHz<br>-period: sets a period value for sample clock in ns or ps<br>-tckdelay: sets a TCK clock pin pulse width delay value<br>-cabletype: sets the type of cable. Values are LATTICE\|USB\|USB2<br>-cableport: sets the port number as integer >= 0. |
| **rva_run** | **rva_run** | Runs until trigger condition to capture data. |
| **rva_stop** | **rva_stop** | Stops without capturing data. |
| **rva_manualtrig** | **rva_manualtrig** | Manual Trigger to capture data. |
| **rva_get_trace** | **rva_get_trace** | Lists all trace signals in a core. |
| **rva_set_core** | **rva_set_core** [-name <name>] [-run <on\|off>] | No arguments return list of core.<br>-name: Select core. Needed for other actions<br>-run: Turns run option on/off for core. |

**Table 18: Reveal Analyzer Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **rva_set_tu** | **rva_set_tu** [-name <name>] [-operator {== \| != \| > \| >= \| < \| <= \| "rising edge" \| "falling edge"}] [-value <value>] [?radix {bin \| oct \| dec \| hex \| <token>}] | No arguments, return list of TU.<br>-name: Select TU. If no options, return options and value for the selected TU.<br>-operator: Sets the comparison operator. Operators are equal to (==), not equal to (!=), greater than (>), greater than or equal to (>=), less than (<), less than or equal to (<=), "rising edge", "falling edge", and serial compare (serial).<br>-value: Sets TU value<br>-radix: Sets TU radix. Options are binary (bin), octal (oct), decimal (dec), hexadecimal (hex), or the name of a token set. |
| **rva_rename_tu** | **rva_rename_tu** <name> <new name> | This function renames TU. |
| **rva_set_te** | **rva_set_te [**-name <name>] [-expression <expression list>] [-enable <on\|off>] | No arguments, return list of TE.<br>-name: Select TE. If no options, return options and value for the selected TE.<br>-expression: Sets TE expression<br>-enable: Enables/disables TE. |
| **rva_rename_te** | **rva_rename_te** <name> <new name> | This function renames TE. |
| **rva_set_trigoptn** | **rva_set_trigoptn** [-teall <AND\|OR>] [-finalcounter <on\|off>] [-finalcountervalue <val>] [-samples <val>] [-numtriggers <val>] [-position <pre\|center\|post\|val>] | No arguments specified will return list of options.<br>-teall: Sets AND ALL or OR ALL for all TEs<br>-finalcounter: Turns final trigger counter on/off<br>-finalcountervalue: Sets final trigger counter value<br>-samples: Sets number of samples to capture<br>-numtriggers: Sets number of triggers to capture<br>-position: Sets trigger position to pre-selected or user value. |
| **rva_add_token** | **rva_add_token** <tokenset name> <name=value> | Add a token with new name and value in a specific token set. |
| **rva_del_token** | **rva_del_token** <tokenset name> <token name> | Delete a specific token in a specific token set. |

**Table 18: Reveal Analyzer Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **rva_set_token** | **rva_set_token** <tokenset name> <token name> -name <new token name> -value <new token value> | Select specific token in specific token set.<br>-name: Set token name<br>-value: Set token value. |
| **rva_add_tokenset** | **rva_add_tokenset** [-tokenset <tokenset name>] [-bits <token bits>] [-token <name=value>] | No arguments, add a token set with default name and bits.<br>-tokenset: Set token set name<br>-bits: Set toke nset bits<br>-token: Add extra tokens. |
| **rva_del_tokenset** | **rva_del_tokenset** <tokenset name> | Delete the specific token set. |
| | **rva_del_tokenset** -all | Delete all token set. |
| **rva_set_tokenset** | r**va_set_tokenset** <tokenset name> -name <new token set name> -bits <new token bits> | Select specific token set<br>-name: Rename a token set<br>-bits: Set number of bits in tokens. |
| **rva_export_tokenset** | **rva_export_tokenset** <file name> | Export all token set to a specific file. |
| **rva_import_tokenset** | **rva_import_tokenset** <file name> | Import and merge all token set from a specific file. |
| **rva_open_pcs** | **rva_open_pcs** | Open connection to Lattice device before read/write began. |
| **rva_close_pcs** | **rva_close_pcs** | Close connection to Lattice device after read/write finished. |
| **rva_read_pcs** | **rva_read_pcs** -byte 1 -addr <address> | Read one byte of data from address in hex. |
| **rva_write_pcs** | **rva_write_pcs** -byte 1 -addr <address> -data <value> | Write one byte of data to address in hex. |
| **rva_run_pcs** | **rva_run_pcs** - config_sram -file <tcl file> | Apply changes to SERDES control.<br>-config_sram: Reload all registers in current DCU with values from config SRAM.<br>-file: Run SERDES commands from Tcl file. |
| **rva_export_pcs** | **rva_export_pcs** <file name> | Export SRV file. |
| **rva_import_pcs** | **rva_import_pcs** <file name> | Import SRV file. |

**Reveal Analyzer Tcl Command Examples** This section illustrates and describes a few samples of Reveal Analyzer Tcl commands.

**Example 1** The following command line example shows how to specify a new project that uses a parallel cable port.

```
rva_new_project –rva untitled –rvl "count.rvl" –dev "LFXP2-
5E:0x01299043" –port 888 –cable LATTICE
```

**Example 2** The following example shows how to set up TU parameters for Reveal Analyzer:

```
rva_set_tu -name TU1 -operator == -value 10110100 -radix bin
```

# Power Calculator Tcl Commands

This section provides Power Calculator extended Tcl command syntax, command options, and usage examples.

**Power Calculator Tcl Command Descriptions** The following table provides a listing of all valid Power Calculator Tcl command options and describes option functionality.

**Table 19: Power Calculator Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **pwc_new_project** | **pwc_new_project** <file> | Create a new project. |
| **pwc_open_project** | **pwc_open_project** <file> | Open a project file. |
| **pwc_save_project** | **pwc_save_project** <file> | Save the current project. |
| **pwc_close_project** | **pwc_close_project** | Close the current project. |
| **pwc_set_afpervcd** | **pwc_set_afpervcd** <file> | Open vcd file and set frequency and activity factor. |
| **pwc_set_device** | **pwc_set_device** -device <value> | Set device. |
| | **pwc_set_device** -package <value> | Set package type. |
| | **pwc_set_device** -speed <value> | Set performance grade. |
| | **pwc_set_device** -operating <value> | Set operating condition. |
| **pwc_set_processtype** | **pwc_set_processtype** <value> | Set device power process type. |
| **pwc_set_ambienttemp** | **pwc_set_ambienttemp** <value> | Set ambient temperature value. |
| **pwc_set_thetaja** | **pwc_set_thetaja** <value> | Set user defined theta JA. |
| **pwc_set_freq** | **pwc_set_freq** <frequency> | Set default frequency. |
| | **pwc_set_freq** -clock <frequency> | Set Clock frequency. |
| | **pwc_set_freq** -timing <option><br>option: min \| pref \| trace | Set frequency by timing. |
| **pwc_set_af** | **pwc_set_af** <value> | Set default activity factor. |
| **pwc_set_estimation** | **pwc_set_estimation** <value> | Sets estimated routing option. |
| **pwc_set_supply** | **pwc_set_supply** -type <value> -voltage <value> -dpm <value> | Set multiplication factor and voltage of named power supply. |

**Table 19: Power Calculator Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **pwc_add_ipblock** | **pwc_add_ipblock** | Add IP Block row. |
| **pwc_set_ipblock** | **pwc_set_ipblock** -matchkeys {<key1> <value1>}+ -setkey <key> <value><br>: iptypename mapping to PGT section, key mapping to _KEY in pgt session, value is its value | Set IP Block row. |
| **pwc_remove_ipblock** | **pwc_remove_ipblock** -matchkeys {<key1> <value1>}+ | Remove IP Block row. |
| **pwc_gen_report** | **pwc_gen_report** <file> | Generate text report and write to file. |
| **pwc_gen_htmlreport** | **pwc_gen_htmlreport** <file> | Generate HTML report and write to file. |

**Power Calculator Tcl Command Examples** This section illustrates and describes a few samples of Power Calculator Tcl commands.

**Example 1** The follow command below creates a PWC project (.pcf) file namced “abc.pcf” from an input UDB file named “abc.UDB”:

```
pwc_new_project abc.pcf -udb abc.udb
```

**Example 2** To set the default frequency to, for example, 100 Mhz:

```
pwc_set_freq 100
```

**Example 3** The command below saves the current project to a new name:

```
pwc_save_project newname.pcf
```

**Example 4** To create an HTML report, you would run a command like the one shown below:

```
pwc_gen_htmlreport c:/abc.html
```

# Programmer Tcl Commands

This section provides the Programmer extended Tcl command syntax, command options, and usage examples. The below commands are only supported in standalone Programmer currently.

**Programmer Tcl Command Descriptions** The following table provides a listing of all valid Programmer Tcl command options and describes option functionality.

**Table 20: Programmer Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **pgr_project** | **pgr_project open** <project_file> | The open command will open the specified project file in-memory. |
| | **pgr_project save** [<file_path>] | Writes the current project to the specified path. If there is no file path specified then it will overwrite the original file. |
| | **pgr_project close** | Closes the current project. If a Programmer GUI is open with the associated project, then the corresponding Programmer GUI will be closed as well. |
| | **pgr_project help** | Displays help for the pgr_project command. |

**Table 20: Programmer Tcl Commands**

| Command | Function (Argument) | Description |
|---|---|---|
| **pgr_program** | <no_argument> | When pgr_program is run without arguments it will display the current status of the available settings. Note that specifying a key without a value will display the current value. The following keys can be used to modify those settings.<br>Generally, the pgr_program command and its sub-commands allow you to run the equivalent process commands from the TCL Console window in the Radiant software interface. These commands can override connection options that are set in user defaults. |
| | **pgr_program set -cable** <LATTICE \| USB \| USB2> | Sets the cable for downloading. |
| | **pgr_program set -portaddress** <0x0378\|0x0278\|0x03bc\|0x0378\|0x0278\|0x03bc\|0x<custom address>><br><EzUSB-0\|EzUSB-1\|EzUSB-2\|...\|EzUSB-15><br><FTUSB-0\|FTUSB-1\|FTUSB-2\|...\|FTUSB-15> | Sets the port address for the downloading. |
| | **pgr_program run** | Executes the current xcf with the current settings. Note that there may be warnings that are displayed in the TCL Console window. These warnings will be ignored and processing will continue. |
| | **pgr_program help** | Displays help for pgr_program command. |
| **pgr_genfile** | <no_argument> | Programmer generate files command (not supported for customer use) |
| | **pgr_genfile set -process** <svf \| vme12> | Sets file type for file generation. |
| | **pgr_genfile set -outfile** <file path> | Sets the output file. |
| | **pgr_genfile run** | Generates file based on the current xcf and current settings. |
| | **pgr_genfile help** | Displays help for pgr_genfile command. |

**Programmer Tcl Command Examples** This section illustrates and describes a few samples of Programmer Tcl commands.

**Example 1** The first command below opens a Programmer XCF project file that exists in the system. There can be many programming files associated with one project. In the GUI interface, the boldfaced file in the Radiant software is the active project file.>

```
pgr_project open /home/mdm/config_file/myfile.xcf
```

**Example 2** The following command sets programming option using a USB2 cable at port address “FTUSB-1, then using pgr_program run to program”.

```
pgr_program set -cable USB2 -portaddress FTUSB-1
```

**Example 3** The following command sets the file generation type for JTAG SVF file, then using pgr_genfile run to generates an output file “mygenfile.svf” in a relative path.

```
pgr_genfile set -process svf -outfile ../genfiles/mygenfile.svf
```

# Revision History

The following table gives the revision history for this document.

| Date | Version | Description |
|---|---|---|
| **02/13/2018** | 1.0 | Initial Release. |