Lattice Radiant Software 2026.1 User Guide

June 26, 2026

Type Conventions Used in This Document

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

Contents

Chapter 1

Introduction

Lattice Radiant™ software is the leading-edge software design environment for cost- sensitive, low-power Lattice Field Programmable Gate Arrays (FPGA) architectures. The Radiant software integrated tool environment provides a modern, comprehensive user interface for controlling the Lattice Semiconductor FPGA implementation process. Its combination of new and enhanced features allows users to complete designs faster, more easily, and with better results than ever before.

This user guide describes the main features, usage, and key concepts of the Radiant software design environment. It should be used in conjunction with the Release Notes and reference documentation included with the product software. The Release Notes document is also available on the Lattice Web site and provides a list of supported devices.

Radiant Software Overview

The Radiant software uses an expanded project-based design flow and integrated tool views so that design alternatives and what-if scenarios can easily be created and analyzed. The *Implementations* and *Strategies* concepts provide a convenient way for users to try alternate design structures and manage multiple tool settings.

System-level information—including process flow, hierarchy, and file lists—is available, along with integrated HDL code checking and consolidated reporting features.

A fast Timing Analysis loop and Programmer provide capabilities in the integrated framework. The cross-probing feature and the shared memory architecture ensure fast performance and better memory utilization.

The Radiant software is highly customizable and provides Tcl scripting capabilities from either its built-in console or from an external shell.

The Radiant software has many of the same features as Lattice Diamond software, and adds new features, such as:

- Constraints support utilizing industry standard SDC format.
- Efficient, easy-to-use integrated graphical user interface (GUI) with a new look-and-feel that gives users more efficient access to popular tools.

- Unified timing analysis engine with enhanced timing reports for faster design timing closure.

User Guide Organization

This user guide contains all the basic information for using the Radiant software. It is organized in a logical sequence from introductory material, through operational descriptions, to advanced topics.

Key concepts and work flows are explained in “Design Environment Fundamentals” on page 17 and “Radiant Software Design Flow” on page 85.

Basic operation of the design environment is described in “User Interface Operation” on page 23.

The chapter “Working with Projects” on page 46 shows how to set up project implementations and strategies.

The chapter “Working with Tools and Views” on page 118 describes the many tool views available.

Chapter 2

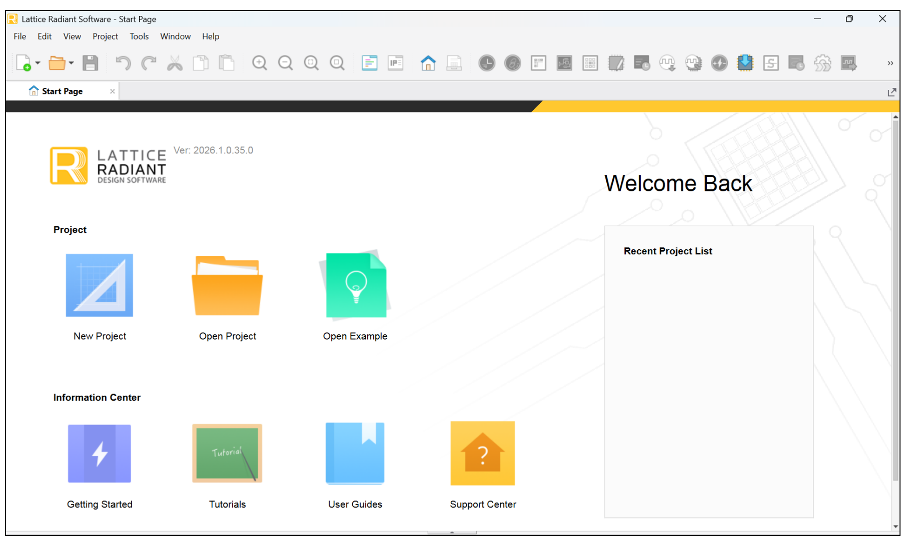
Getting Started

This chapter explains how to run the Radiant software and open or create a project. For more information about project fundamentals, see the chapters "Design Environment Fundamentals" on page 17 and "Working with Projects" on page 46.

Prerequisites

To run the Radiant software, select **Radiant Software** from the installation location. This opens the default Start Page, shown in Figure 1.

Figure 1: Default Start Page



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 a 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:

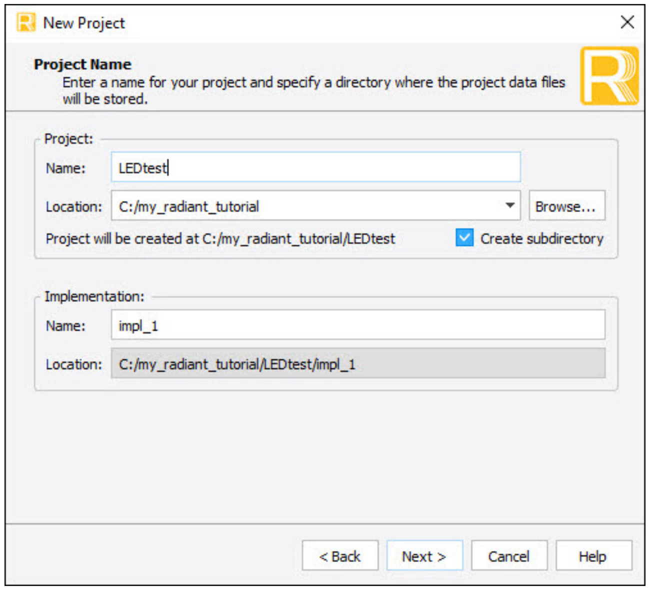
1. From the Radiant main window, click the **New Project** button, or choose **File > New > Project**.

   The New Project confirmation window opens, shown in Figure 2.

Figure 2: New Project Confirmation Window

2. Click **Next**. The New Project wizard opens, shown in Figure 3.

Figure 3: New Project Wizard



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 "Implementations" on page 48.

   - To create a sub-directory with the same name as your location directory, click **Create Subdirectory**. This will allow you to keep your project implementations separate. If this box is left unchecked, no sub-directory will be created in the project directory.

   - 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**.

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

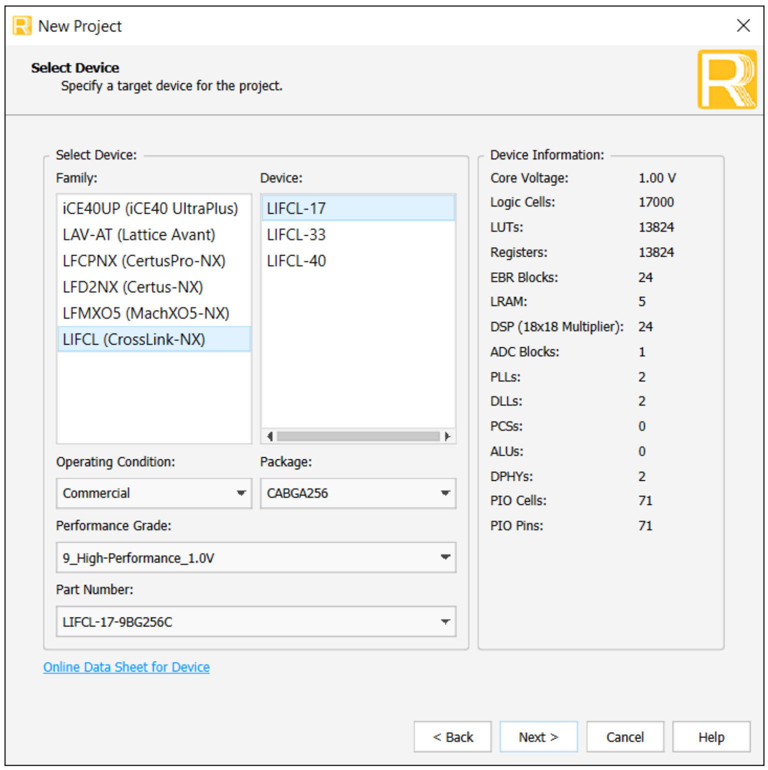
   c. To copy the added source files to the implementation directory, select **Copy source to implementation directory**. If you prefer to reference these files, clear this option.

d. To create empty Lattice Design Constraint (.ldc) file and Physical Constraint File (.pdc) files that can be edited at a later time, select **Create empty constraint files.** Refer to the chapter "Implementations" on page 48 for more information about constraint files.

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

5. In the Select Device dialog box, shown in Figure 4, select a device family and a specific device within that family. Then choose the options you want for that device. When you finish, click **Next**.

Figure 4: Select Device Dialog Box



6. In the Select Synthesis Tool dialog box, select the synthesis tool that you want to use. This choice can be changed at any time. When you finish, click **Next**.

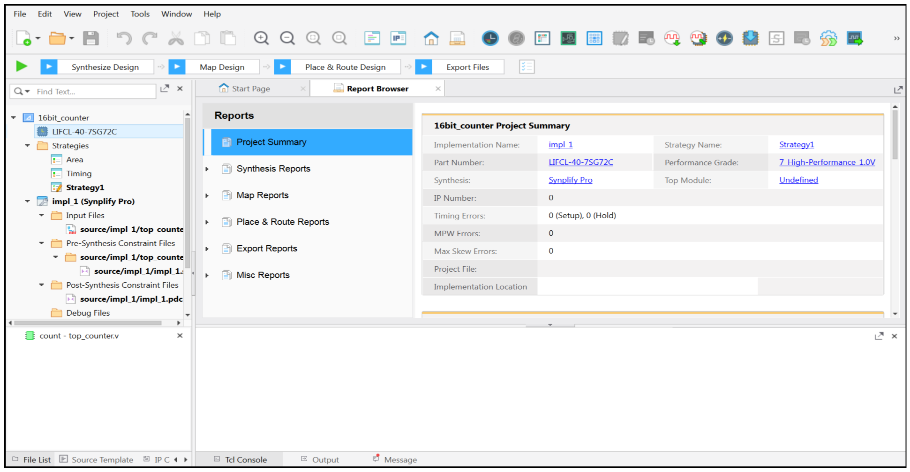
7. In the Project Information dialog box, make sure the project settings are correct.

   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.

   Click **Finish**. The newly created project, shown in Figure 5, is now created and open.

Figure 5: Opened Project



Select the **File List** tab under the left pane, to view the Test project file list.

To close a project, choose **File > Close Project**.

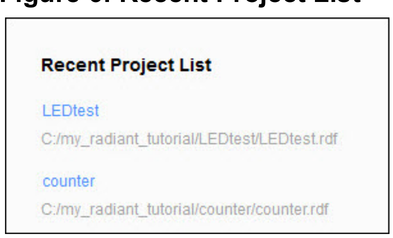
Opening an Existing Project

Use one of the following methods to open an existing Radiant software project:

- On the Start Page, click the **Open Project** button.
- From the File menu, choose **Open > Project**.
- On the Start Page, select the desired project from the Recent Projects List. Alternatively, choose a recent project from the **File > Recent Projects** menu.

You can use the Options dialog box to increase the number of projects that are shown in the Recent Projects list and to automatically load the previous project at startup. Choose **Tools > Options** to open the dialog box. To increase the number of recent projects listed, click the **General** tab and enter a number for “Maximum items shown in Recent Project List” (up to 32). To automatically open the previous project during startup, click the **Startup** tab and then choose **Open Previous Project** from the “At Lattice Radiant Software startup” menu.

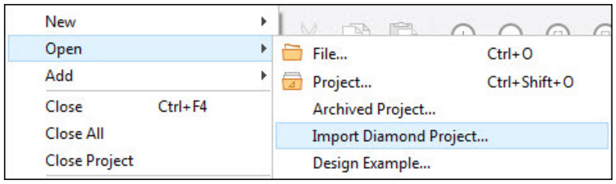
Figure 6: Recent Project List



Importing a Lattice Diamond Project

To import a Lattice Diamond project into the Radiant software, choose **File > Open > Import Diamond Project**.

Figure 7: Import Lattice Diamond Project



The file browser applies an ***.ldf** file filter to help you find Lattice Diamond project files. The Lattice Diamond project is converted to a Radiant project.

For more information about importing Lattice Diamond projects into the Radiant software, refer to the *Lattice Radiant Software Guide for Diamond Users*.

Next Steps

After you have a project opened in the Radiant software, you can go sequentially through the rest of this user guide to learn how to work with the entire design environment, or you can go directly to any topic of interest.

- The chapters "Design Environment Fundamentals" on page 17 and "Radiant Software Design Flow" on page 85 provide explanations of key concepts.
- "User Interface Operation" on page 23 provides descriptions of the functions and controls that are available in the Radiant software environment.
- The chapters "Working with Projects" on page 46 and "Working with Tools and Views" on page 118 explain how to run processes and use the design tools.
- **Reference Guides > Tcl Command Reference Guide** in the Radiant Help provides an introduction to the scripting capabilities available, plus command-line shell examples.
- "Advanced Topics" on page 460 provides further details about environment options, shared memory, and Tcl scripting.

Chapter 3

Design Environment Fundamentals

This chapter provides background and discussion on the technology and methodology underlying the Radiant software design environment. Important key concepts and terminology are defined.

Overview

Understanding some of the fundamental concepts behind the Radiant software framework technology will increase your proficiency with the tool and allow you to quickly come up to speed on its use.

The Radiant software is a next-generation software design environment that uses a new project-based methodology. A single project can contain multiple implementations and strategies to provide easily managed alternate design structures and tool settings.

The process flow is managed at a system level with run management controls and reporting. Context-sensitive views ensure that you only see the data that is available for the current state in the process flow.

The shared memory technology enables many of the advanced functions in the Radiant software. Easy cross-probing between tool views and faster process loops are among the benefits.

Note

By loading the Radiant software multiple times, you can run different Radiant projects simultaneously. However, you must not load the same project in more than one Radiant software instance, as software conflicts can occur.

The Radiant software can also be run remotely. Refer to the *Lattice Radiant Software Installation Guide for Windows* or *Lattice Radiant Software Installation Guide for Linux* for more information.

Project-Based Environment

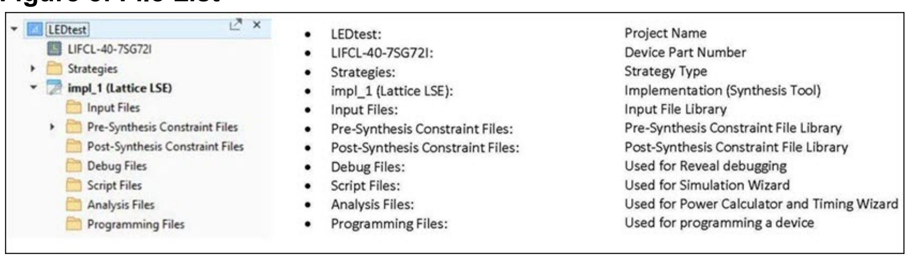
A project in the Radiant software consists of the following file types:

- HDL source files
- Constraint files
- Reveal debug files
- Script files for simulation
- Analysis files for power calculation and timing analysis
- Programming files

The Radiant software also includes settings for the targeted device and the different tools. The project data is organized into implementations, which define the project structural elements, and strategies, which are collections of tool settings.

The following File List shows the items in a sample project.

Figure 8: File List



Each item that is displayed in **bold** means that it has been selected as the active item for an implementation. An implementation displayed in **bold** means that it has been selected as the currently active implementation for the project. Your project must have one active implementation, and the implementation must have one active strategy. Optional items, such as Reveal hardware debugger files, can be set as active or inactive.

The project is the top-level organizational element in the Radiant software, and it can contain multiple implementations and multiple strategies. This enables you to try different design approaches within the same project. If you want to have a Verilog version of your design, for example, make an implementation that consists of only the Verilog source files. If you want another version of the design with primarily Verilog files but a Structural Verilog (.vm) netlist for one module, create a new implementation using the Verilog and .vm source files. Each implementation can have Verilog, VHDL or Structural Verilog source or mixed of them. The same project and design is used, but with a different set of modular blocks.

Similarly, if you want to try different implementation tool options, you can create a new strategy with the new option values.

You manage these multiple implementations and strategies for your project by setting them as active. There can only be a single active implementation with its one active strategy at a time.

Process Flow

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.

Figure 9: 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 and Lattice Synthesis Engine (LSE) runs to synthesize the design. By default, this process runs the LSE tool.

  If you are using Synplify Pro, choose Synplify Pro as the synthesis tool (**Project > Active Implementation > Select Synthesis Tool**).

  - **Post-Synthesis Timing Analysis**

    Runs timing analysis after the Synthesize Design process.

  - **Post-Synthesis Simulation File**

    Generates a netlist file <*file_name*>_syn.vo used for functional verification.

- **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 (for example, configurable logic blocks).

  - **Map Timing Analysis**

    Runs timing analysis after the Map Design process.

- **Place & Route Design**

  After a design has undergone the necessary translation to bring it into the Unified Database (.udb) format, you can run the Place & Route Design process. This process takes a mapped physical design .udb file, places and routes the design, and then outputs a file that can then be processed by the design implementation tools.

  - **Place & Route Timing Analysis**

    Runs timing analysis after Place & Route process.

  - **I/O Timing Analysis**

    Runs I/O timing analysis that allows you to view the path delay tables and Timing Analyzer report of your timing constraints after placement and routing.

- **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 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.

  - **IBIS Model**

    This process generates a design-specific IBIS (I/O Buffer Information Specification) model file (<*project_name*>.ibs).

    IBIS models provide a standardized way of representing the electrical characteristics of a digital IC's pins (input, output, and I/O buffers).

  - **Gate-Level Simulation File**

    This process backannotates the routed design with timing information so that you may run a simulation of your design. The backannotated design is a Verilog netlist.

  The Reports view allows you to examine and print process reports.

  Messages are displayed in the Messages window at the bottom of the Radiant software main window.

The process status icons are defined as follows:

Process in initial state (not processed)

Process completed successfully

Process completed with unprocessed subtasks

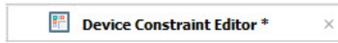
Process failed

Shared Memory

The Radiant software uses a shared memory architecture. All tool and data views look at the same design data at any point in time. This means that when you change a data element in one view of your design, all other views will see the change, whether they are active or not.

When project data has been changed but not yet saved, an asterisk (*) is displayed in the title tab of the view.

Figure 10: Title Tab with Changed Content Indication



Notice that the asterisks indicating changed data will appear in all views referencing the changed data.

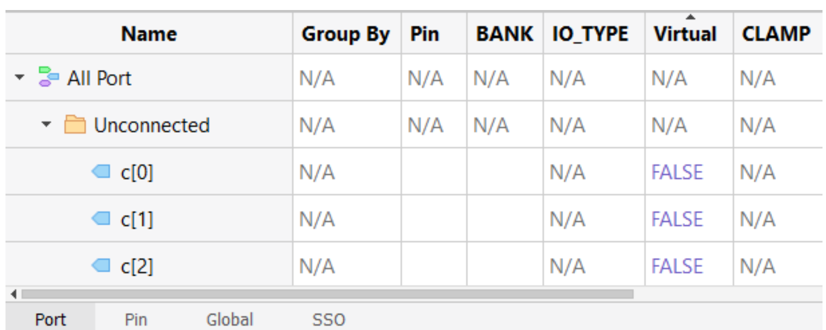
If a tool view becomes unavailable, the Radiant software environment will need to be closed and restarted.

Context-Sensitive Data Views

The data in shared memory reflects the state or context of the overall process flow. This means that views such as Device Constraint Editor Spreadsheet View will display only the data that is currently available, depending on process steps that have been completed.

For example, Figure 11 shows the Process flow before Synthesis. Therefore, Spreadsheet View shows no IO Type or PULLMODE.

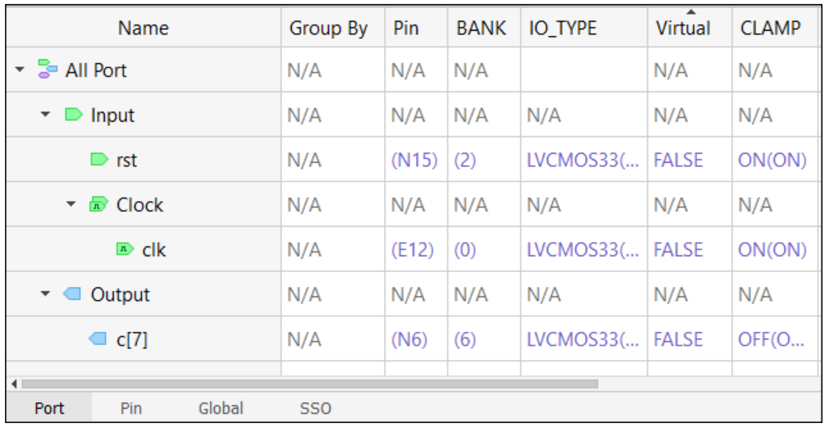
Figure 11: Process Completed Before Synthesis



Name	Group By	Pin	BANK	IO_TYPE	Virtual	CLAMP
All Port	N/A	N/A	N/A	N/A	N/A	N/A
Unconnected	N/A	N/A	N/A	N/A	N/A	N/A
c[0]	N/A			N/A	FALSE	N/A
c[1]	N/A			N/A	FALSE	N/A
c[2]	N/A			N/A	FALSE	N/A

After Synthesis has been completed, Spreadsheet View displays IO Type and PULLMODE assignments, as shown in Figure 12.

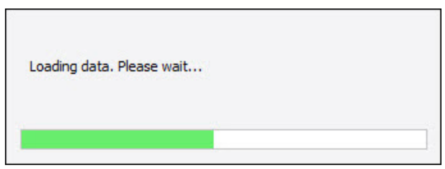
Figure 12: Process Completed Through Synthesis



Name	Group By	Pin	BANK	IO_TYPE	Virtual	CLAMP
All Port	N/A	N/A	N/A		N/A	N/A
Input	N/A	N/A	N/A	N/A	N/A	N/A
rst	N/A	(N15)	(2)	LVCMOS33(...	FALSE	ON(ON)
Clock	N/A	N/A	N/A	N/A	N/A	N/A
clk	N/A	(E12)	(0)	LVCMOS33(...	FALSE	ON(ON)
Output	N/A	N/A	N/A	N/A	N/A	N/A
c[7]	N/A	(N6)	(6)	LVCMOS33(...	FALSE	OFF(O...

When you see the “Loading Data” message displayed in Figure 13, it means that a process has been completed and that the shared memory is being updated with new data.

Figure 13: Loading Data



All tool views are dynamically updated when new data becomes available. This means that when you rerun an earlier process while a view is open and displaying data, the view will remain open but dimmed because its data is no longer available.

Chapter 4

User Interface Operation

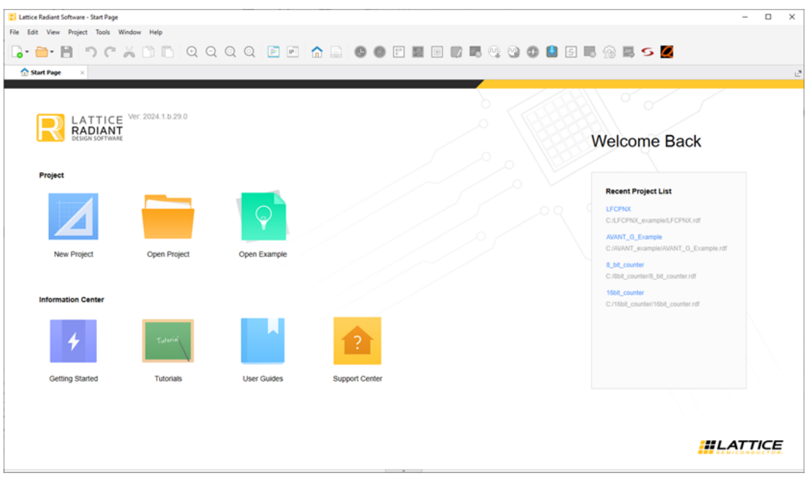
The Radiant user interface (UI) provides a comprehensive, integrated tool environment. The UI is very flexible and configurable, enabling you to store constraints for the layout you choose.

This chapter describes the user interface, controls, and basic operation of the Radiant software. Each major element of the interface is explained. The last section in the chapter describes common user interface tasks.

Start Page

The Start Page contains three major sections, as shown in Figure 14.

Figure 14: Default Start Page



- **Project**: This section allows you to create a new project; open an existing Project, and open an example.
- **Information Center:** This section has links to Getting Started, Tutorials, User Guides, and Support Center.
- **Recent Project List:** Provides a quick way to load a recent project you've been working on.

The Start Page appears in the View area by default when the Radiant software is first launched, and can be opened from the **View** tab on the menu.

The Start Page can be closed, opened, detached, and attached using the Attach button. See "Basic UI Controls" on page 27.

Menus and Toolbars

At the top of the main window is the menu and toolbar area. High-level controls for accessing tools, managing files and projects, and controlling the layout are contained here. All toolbar functionality is also contained in the menus. The menus also have functions for system, project and toolbar control.

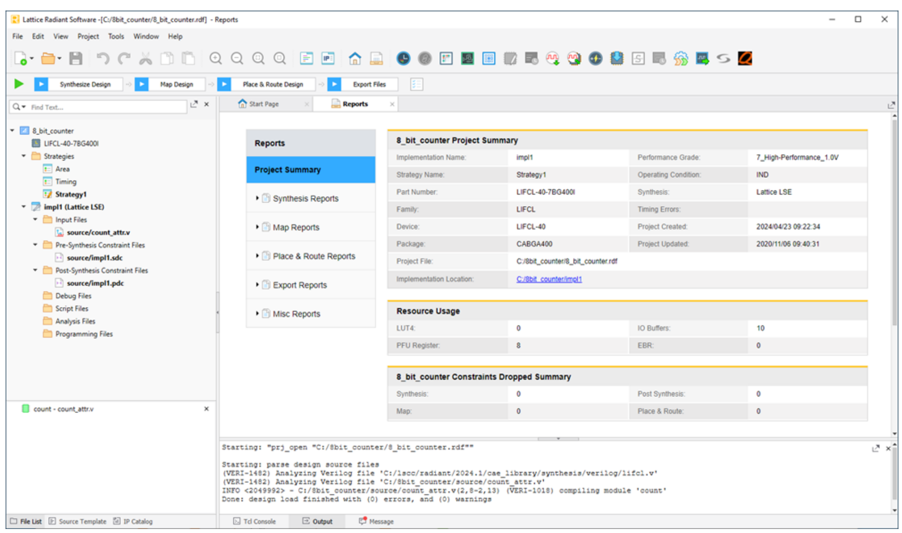
The Process Toolbar lists all the processes available, such as Synthesize Design, Map Design, Place & Route Design, and Export Files. A process is a specific task in the overall processing of a source or project. You can view the available processes for a design in the Process Toolbar. Click **Task Detail View** to see detailed information of the processes available.

Reports and Messages Views

The Reports view allows you to examine and print process reports. There are two panes in the Reports view. The left pane lists the reports. The right pane displays the reports.

Log messages are displayed in the Output frame of the Radiant software main window.

Figure 15: Reports and Log Message Views



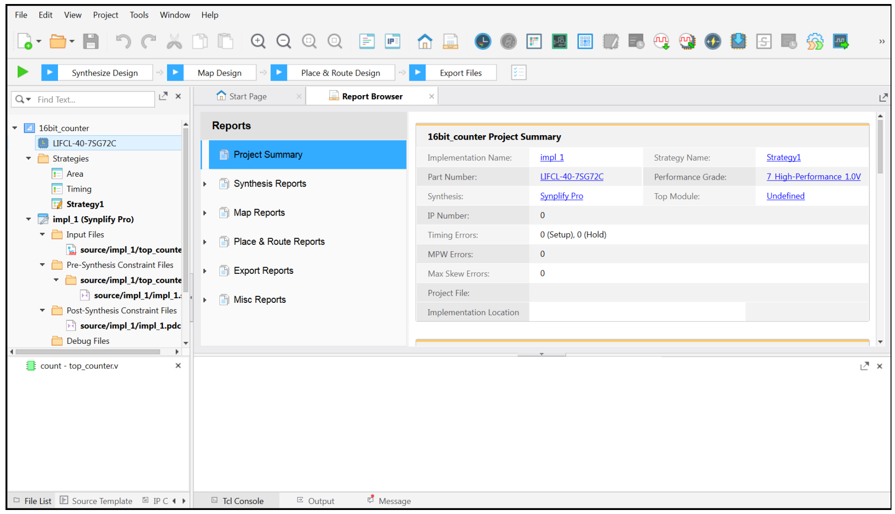
File List Views

In the middle of the main window on the left side is the File List area. This is where the overall project and process flow is displayed and controlled.

Figure 16: File List Area

File List

Hierarchy View



Tabs at the bottom of the File List area allow you to select between the following views:

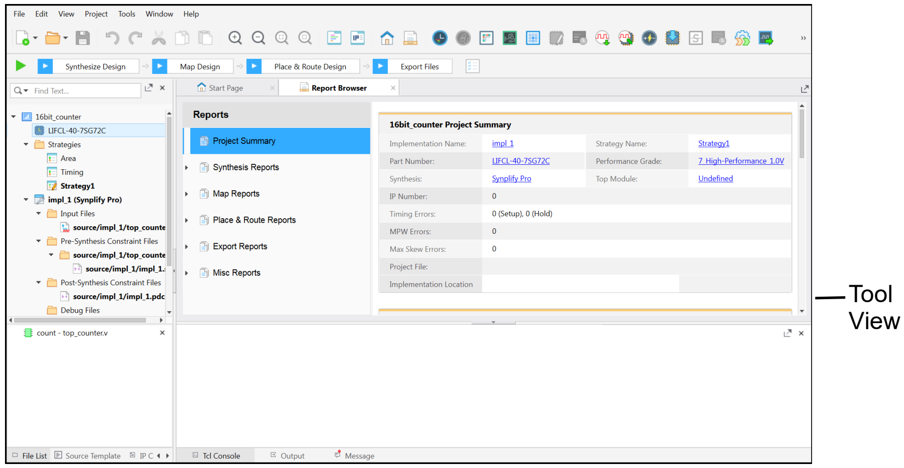
- File List – shows the files in the project organized by implementations and strategies. This is not a hierarchical listing of the design.
- Source Template – provides templates for creating VHDL, Verilog, and Constraint files.
- IP Catalog – lists available modules/intellectual properties (IP).

Underneath the File List is the Hierarchy View area. It allows you to view the hierarchical design representation. Hierarchy view shares the left pane with File List view.

Tool View Area

In the middle of the main window on the right side is the Tool View area. This is where the Start Page, Reports View, and all the Tool views are displayed.

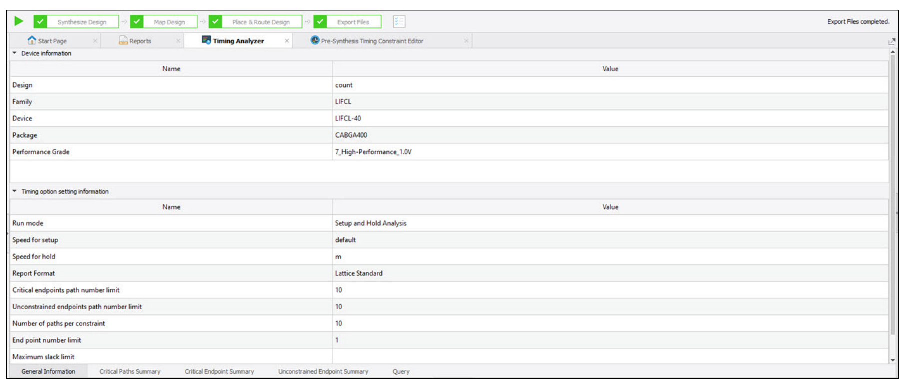
Figure 17: Tool View Area



Multiple tools can be displayed at the same time. The tool tabs include controls for grouping the tool views as well as integrating all tool views back into the main window.

Each tool view is specific to its tool and can contain additional toolbars and multiple panes or windows controlled by additional tabs. The chapter "Working with Tools and Views" on page 118 provides more details about each tool and view.

Figure 18: Multiple Tools

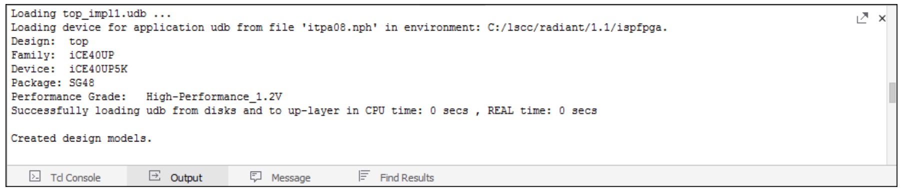


Output and Tcl Console

Near the right bottom of the main window is the Tcl Console, Output, and Message area.

Tabs at the bottom of this area allow you to select between Tcl Console, Output, and Message. Tool output is automatically displayed in the Output tab, and Errors and Warnings in the Message tab.

Figure 19: Output and Tcl Console Area



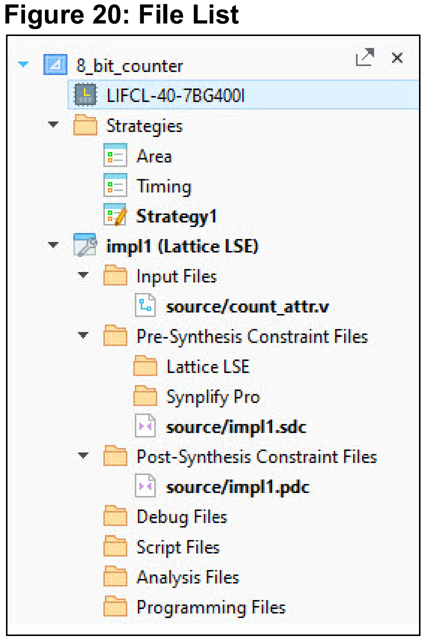
Basic UI Controls

The Radiant software environment is based on modern industry standard user interface concepts. The menus, toolbars, and mouse pointer all behave in familiar ways. You can resize any of the window panes, drag and drop elements, right-click a design element to see available actions, and hold the mouse pointer over an object to view the tool tip. Window panes can also be detached from the main window and operated as independent windows.

File List

The File List is a project view that shows the files in the project, including implementations and strategies. It is not a hierarchical listing of the design, but rather a list of all the design source, configuration and control files that make up the project.

Figure 20: File List



At the top of the File List is the project name. Directly below the project name is the target device, followed by the strategies, and then the implementations. There must be one active implementation, and it must have one active strategy. Active elements are indicated in **bold**.

You can right-click any file or item in the File List to access a pop-up menu of currently available actions for that item. The pop-up menu contents vary, depending on the type of item selected.

The File List view can be hidden by clicking the small arrow in right border: “Click to show/hide side panel.”

Using Revision Control for Radiant Projects

Radiant supports revision control for your projects. When changes are logged into the revision control system, you can get all the change logs and switch to previous milestones if necessary. Compile times can be reduced when using revision control with Radiant design. The compilation will only start when inputs are changed.

Recommended Revision Control System

Radiant recommends the following revision control systems:

- GIT
- SVN
- Perforce

Radiant Revision Control Strategies

You can select different revision control systems to work with Radiant, the software does not directly integrate a specific revision control into the system. Radiant recommends two strategies:

- **Minimum Files Strategy (Radiant Recommendation)** – The revision control system only manages the necessary files of the project. Other files – including some intermediate and configuration files, are only used in your project and will not be submitted to the version management system.

  The following files are recommended to commit to revision control systems:

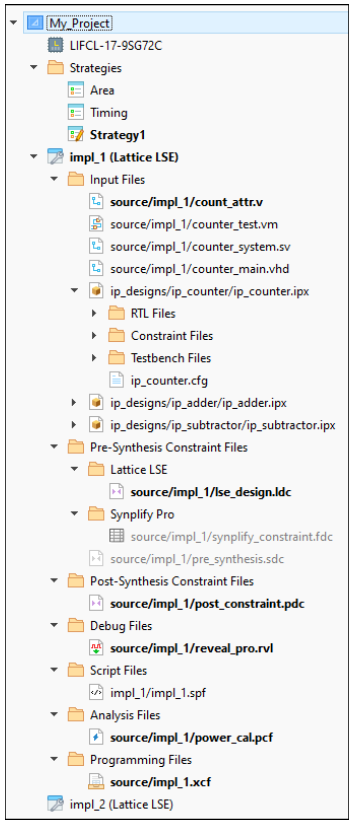
  - .rdf project file and .sty strategy file
  - All source files under source directory
  - All files under ip_directory

- **Complete Strategy** – You can also add some intermediate files on the project to the revision control system.

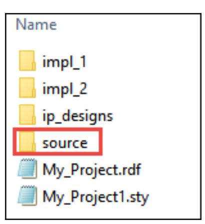
  For this strategy, the files under implementation can be committed to the revision control system. You can select files to submit to the system under each implementation, which mainly includes intermediate and analysis files.

It is recommended to use the following strategy directory structure to manage source code and designs:

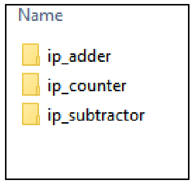
- Radiant design can use one source directory to put all source or design files. The software can automatically identify various types of files and display them in categories in the **File List View**.



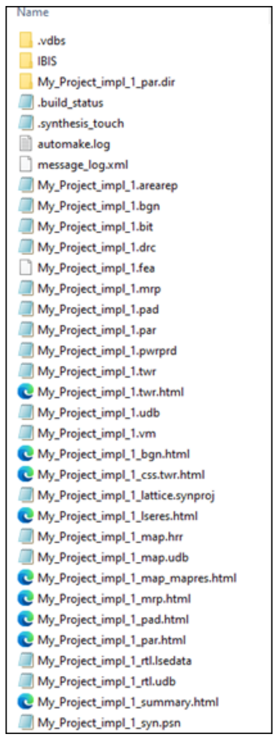
- Use a source directory to place all source and design files for every implementation. Under the source folder, each implementation has its own file directory (e.g. impl_1, impl_2).



- Create an IP directory (e.g. ip_designs) under the source directory of the project to store all IP designs of Radiant. Each IP design has a subdirectory under the source directory to store IP source code and design files.



- Use one implementation directory (e.g. impl_1, impl_2) to compile the relative implementation of the project. All intermediate files will be generated under this folder (.udb, status, report, and other intermediate type files are all here).



Radiant File Types

The Radiant project directory contains the following file types:

- .rdf file – Radiant project file
- .sty file – Radiant strategy file

Common source or design file types under the source directory:

- .v file – Verilog file
- .vhdl file – VHDL file
- .sv file – System Verilog file
- .ldc file – LSE Design Constraints file
- .pdc file – Post-Synthesis Constraint file

- .pcf file – Power Calculator file
- .sdc file – Pre-Synthesis Constraint file
- .rvl file – Reveal Project file
- .rva file – Reveal Analyzer file
- .vm file – Structural Verilog file
- .fdc file – Synplify Pro Constraint file
- IP_ folder – One IP instance file under the folder

Other intermedia file types:

- .ini file – Initialization file
- .log file – Log file
- .build_status file – Status file
- .xml file – XML configuration file
- .html file – HTML file
- .ibs file – Input/Output Buffer Information Specification file
- .udb file – User design Database file
- .bit file – Bitstream
- synthesis.log – LSE synthesis report
- .mrp – Radiant map report
- .twr – Radiant timing report

Note:

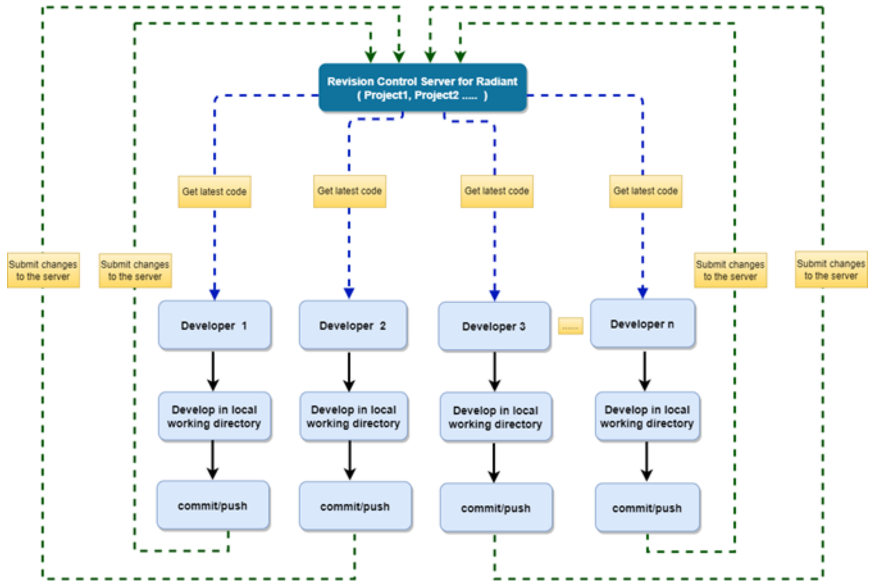
Intermedia files might be re-generated if you rerun the flow.

Radiant Revision Control Workflow

The following diagrams show the revision control workflow of Radiant.

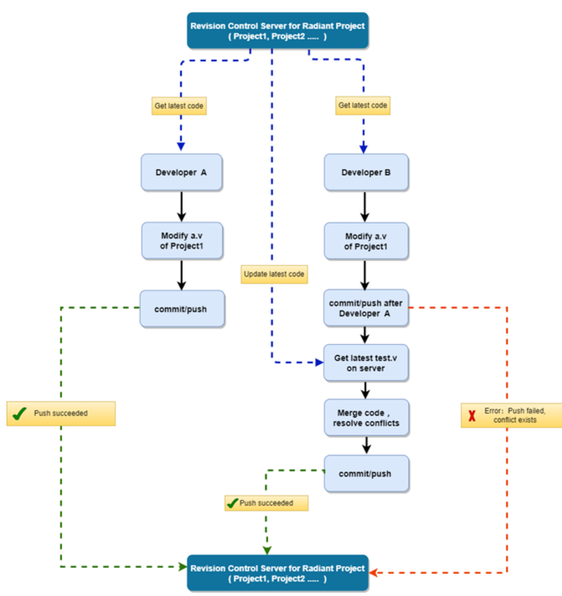
Revision Control Normal Workflow

You can get the latest code from the radiant version server and develop in your own environment and working directory. Later on, you can submit the modified code or new design to the radiant version server as needed.



Multiple developers working on the same file

You may have your code changes and modify the same source file or design. For example, if you submit a file and another user updates it, you will get an error. At this time, you need to update the latest code locally from the server, integrate it with your modifications, and submit it again. This is the collaborative operation of multiple users for the revision control system.

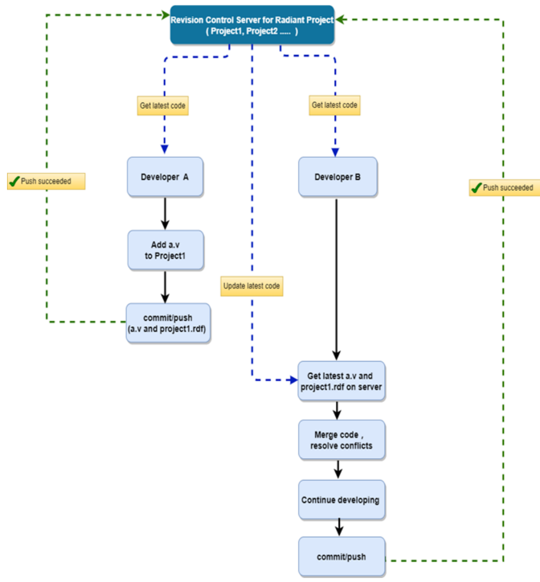


Multiple developers adding files

If you add a new a.v file in the design, and submit this file and the Radiant project file to the revision control server, another user can do the following:

- Get the latest code from the server

- Merge the local code
- Resolve the conflict
- Continue the subsequent development



Source Template

The Source Template is a project view that provides templates for creating VHDL, Verilog, and constraint files. Templates increase the speed and accuracy of design entry. You can drag and drop a template directly to the source file. You can also create your own templates.

To access templates, choose **View > Show Views > Source Template**, or click icon in the toolbar, or click on the **Source Template** tab in the bottom-left pane, to locate and access the following templates:

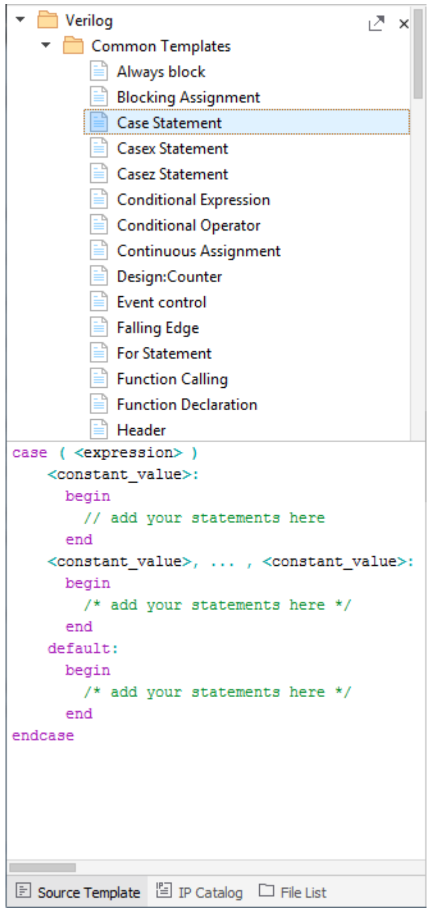
- Verilog, including common and Parameterized Module Instantiation (PMI), Primitives, Attributes, Encryption, and User Templates
- VHDL, including common, PMI, Primitives, Attributes, Encryption, and User Templates
- Constraints for LSE, including Timing and Physical constraints and User Templates

Note

For more information on PMI, refer to the Radiant Software Help. See **User Guides > Entering the Design > Designing with Soft IP, Modules, and PMI > PMI or IP Catalog?**

You can simply drag any template and drop it into your source file.

Figure 21: Source Template

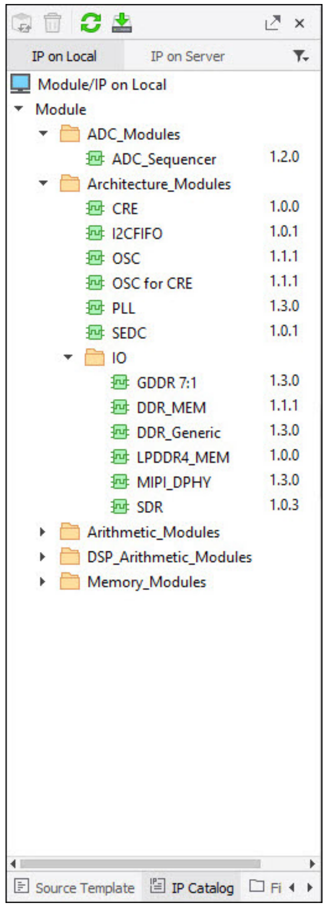


IP Catalog

IP Catalog enables you to customize a collection of functional blocks from Lattice Semiconductor. Through the IP Catalog, you can access two types of functional blocks, Modules and IP.

To access IP catalog, choose **View > Show Views > IP Catalog**, or click icon in the toolbar, or click on the **IP Catalog** tab in the bottom-left pane.

Figure 22: IP Catalog

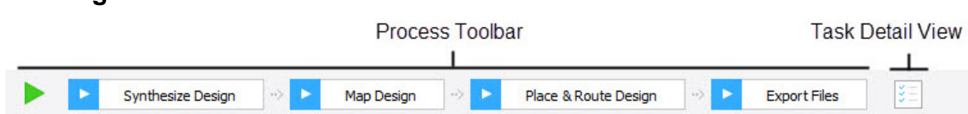


Each module is configurable with a unique set of properties. Once generated, the module or IP appears in your design's File List.

Process

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.

Figure 23: Process Toolbar



The process status icons are defined as follows:

- Process in initial state (not processed)
- Process completed successfully

Process completed with unprocessed subtasks

Process failed

For more detail of different designs and Export Files available, see "Process Flow" on page 19.

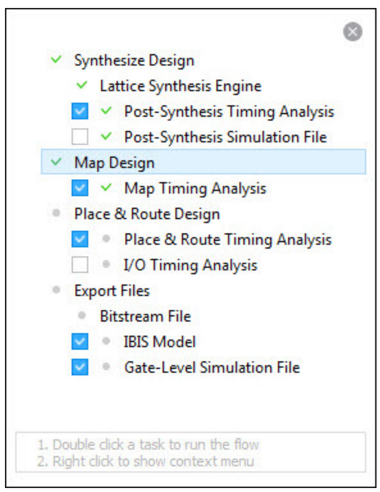
Task Detail View

Click Task Detail View to see detailed information of each process.

The default design flow processes are marked by check marks. To enable the remaining tasks, either check-mark the specific task and rerun the process step, or double-click the task's name. You can also right-click on the task to show the context menu.

Once the process has finished, the process status icon next to the task replaces the gray dot.

Figure 24: Task Detail View

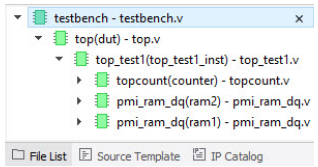


Processes are grouped into categories according to their functions. To learn more about each process, view "Design Flow Processes" on page 86.

Hierarchy

The Hierarchy view is a project view that displays the design hierarchy and is displayed by default. The hierarchical view is available when File List tab is selected.

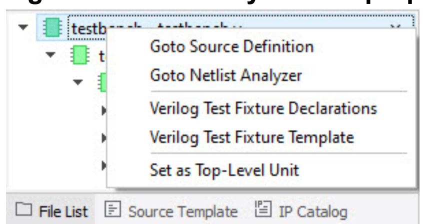
Figure 25: Hierarchy View



If you would prefer that it not open by default, simply close Hierarchy View. The next time you launch the Radiant software, the Hierarchy View will not be opened. You can open it manually by selecting it from the View > Show View menu.

Right-click any of the objects in the Hierarchy View to see the available actions.

Figure 26: Hierarchy Item Pop-up Menu



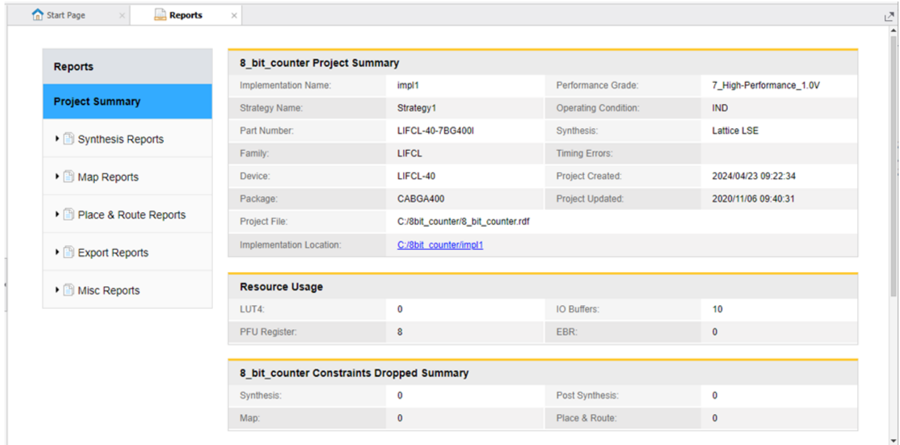
The Hierarchy view can be selected, closed, and opened.

Reports

The Reports View provides a centralized reporting mechanism in the Tools view area. The Reports View is automatically displayed and updated when processes are run. It provides a separate tab for the current implementation, enabling you to compare results quickly.

The right pane displays the report for the selected step. You can also click the icon in the toolbar.

Figure 27: Reports View



The Reports pane on the right shows the detail of the project summary and resource usage. It also shows the number of dropped constraints from Synthesis, Post-Synthesis, MAP, and Place & Route.

The Report View can be selected, closed, opened, detached, and attached with the Attach button. See “Basic UI Controls” on page 27.

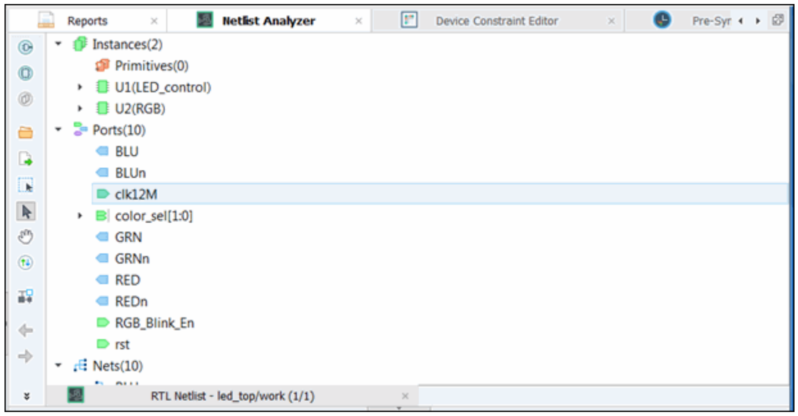
Tool Views

The Tool view area of the UI displays the tools that are currently active. Each tool that you have opened from the toolbar or the Tools menu is displayed. The Reports and Start page, which can be opened from the toolbar or the Windows menu, are also displayed. When multiple tools are active, the display can be controlled with the tab group functions in the Window menu. See “Common Tasks” on page 41 for more information on tab group functions.

Each tool view is specific to its tool and can contain additional toolbars, multiple panes, or multiple windows controlled by additional tabs. See “Working with Tools and Views” on page 118 for descriptions of each tool and view, plus details on controlling their display.

The Tool views can be selected, closed, opened, detached, and attached using the Attach button. See “Basic UI Controls” on page 27

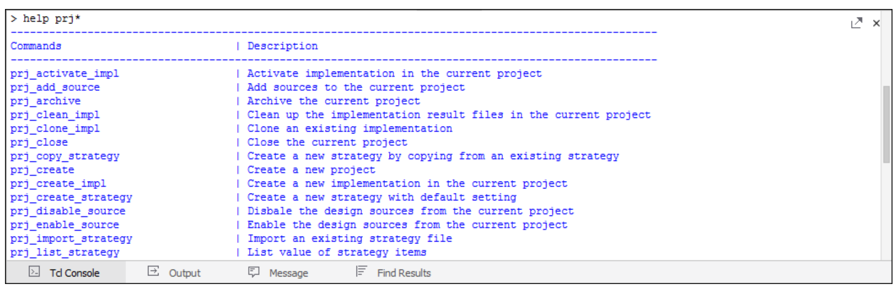
Figure 28: Tool View Tab Title



Tcl Console

The Tcl Console is an integrated console for Tcl scripting. You can enter Tcl commands in the console to control all of the functionality of the Radiant software. Use the Tcl help command (help <*tool_name*>*) to display a list of valid extended Tcl commands.

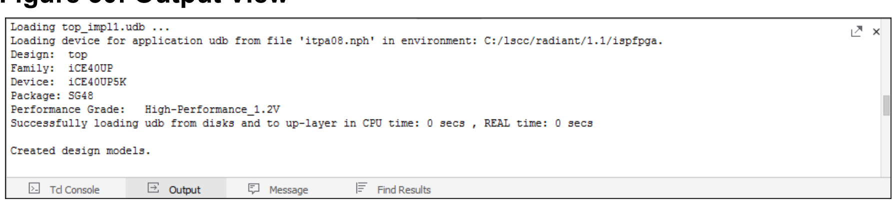
Figure 29: Tcl Console



Output

The Output View is a read-only area where tool output is displayed.

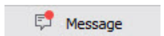
Figure 30: Output View



Message

There are three message types available:

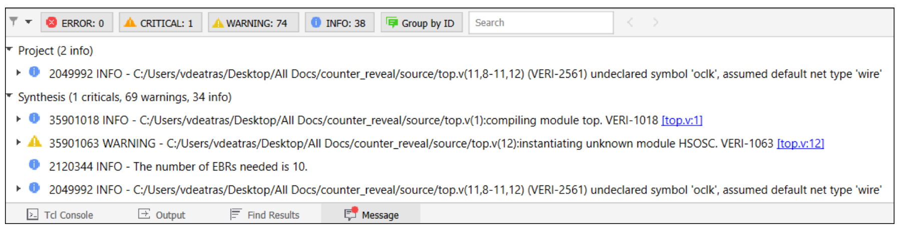
- Error -- displayed in red.
- Critical -- displayed in orange.
- Warnings -- displayed in yellow.
- General Information -- displayed in blue.



A red dot in the Message tab provides a visual notification that a new message/warning was received. Once you view the notification, the dot disappears.

Right-clicking a message provides a menu of commands, including **Location in > Text Editor**, which opens the source file in the Source Editor and highlights the location of the problem.

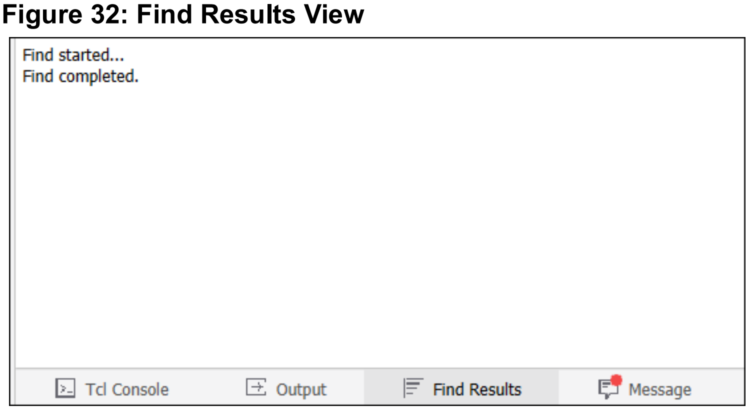
Figure 31: Message Display



Find Results

The **Edit > Find in Files** command enables you to search for information in the files within your project directory. The search results are then displayed in the Find Results view.

Figure 32: Find Results View



Common Tasks

The Radiant software UI controls many tools and processes. The following sections describe some of the more commonly performed tasks.

Controlling Views

All of the views in the Radiant software are controlled in a similar manner, even though the information they contain varies widely. Here are some of the most common operations:

- Open – Use the **View > Show Views** menu selections or right-click in the menu or toolbar areas to select a view from the pop-up menu.
- Select – If a view is already open you can select its tab to bring it to the front.
- Detach – Click the detach button in the upper right corner of the view.
- Attach – Click the attach button in the upper right corner of the view.
- Move – Click and hold a view's tab, and then drag and drop the view to a different position among the open views.

Using a Tab Group You can use the Window menu to split off a view and control it as a separate tab group. This allows you to examine two open views side by side. The controls work as follows:

- Split Tab Group – displays two views side by side. See Figure 33.
- Move to Another Tab Group – moves the selected tab to the other tab group. See Figure 34.
- Merge Tab Group – merges a split tab group back into the primary view
- Switch Tab Group Position – switches the positions of the two tab groups.

Figure 33: After Split Tab Group Command Used on Physical Designer

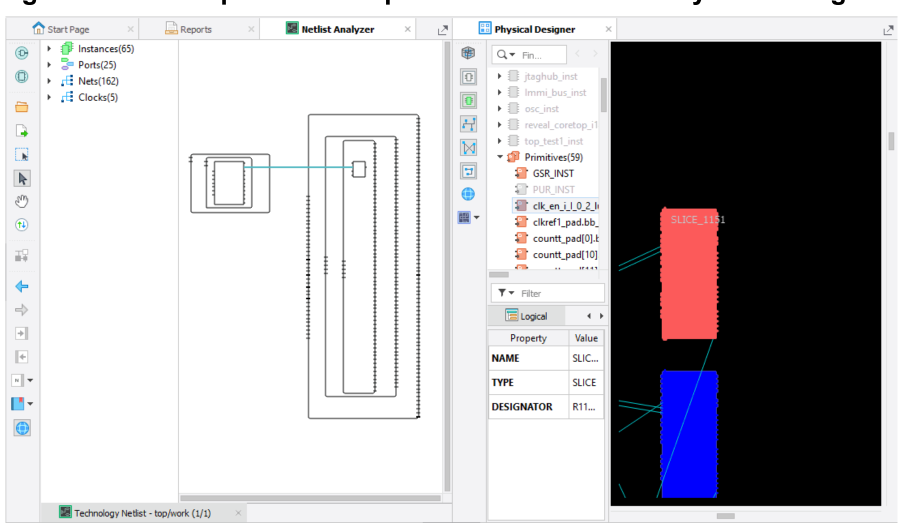
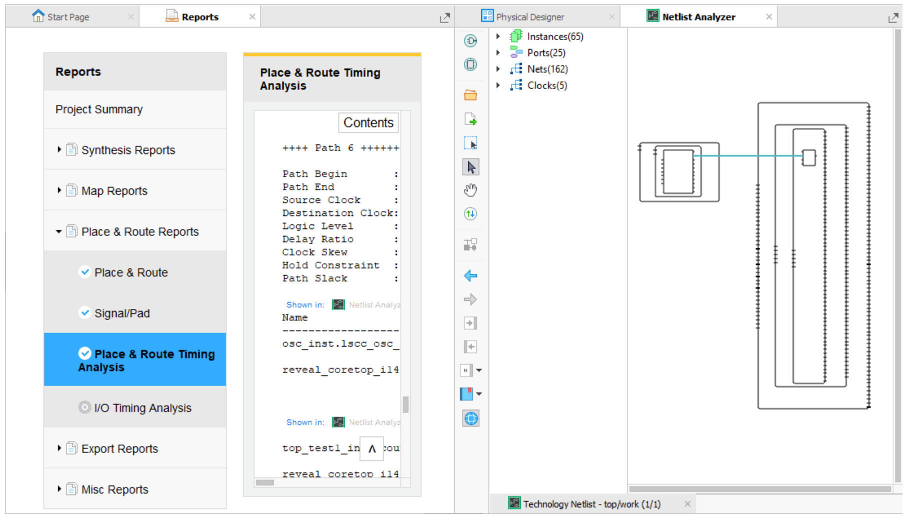


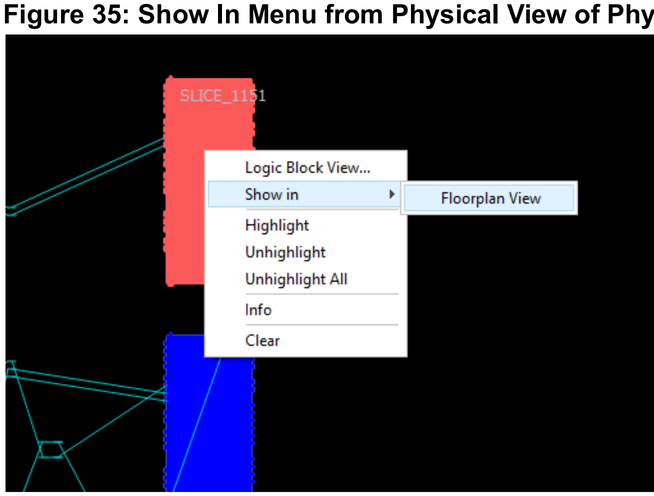
Figure 34: After Move to Another Tab Group Used on Netlist Analyzer



Cross-Probing

It is possible to select a data object in one view and see that same data object in a different view or views. Right-click an object to see if cross-probing is available. If it is, you will see a **Show In** sub-menu with the available views listed. If you select a view that is not yet open, the Radiant software will open it automatically. Cross-probing is available between Floorplan View and Physical View of Physical Designer, and from Netlist Analyzer to Physical Designer.

Figure 35: Show In Menu from Physical View of Physical Designer



During the Radiant flow, various timing analyses and reports are created. You can view a specific path in Netlist Analyzer, Physical Designer's Floorplan View, and Physical Designer's Physical View. This allows for flexibility and reduced debugging effort.

NOTE

Cross-probing to Netlist Analyzer is available only if the selected synthesis tool is LSE.

In the Reports tab, view any timing analysis report and identify a path to view. If cross-probing is available, the specific icon tools become visible, as shown in the following figure.

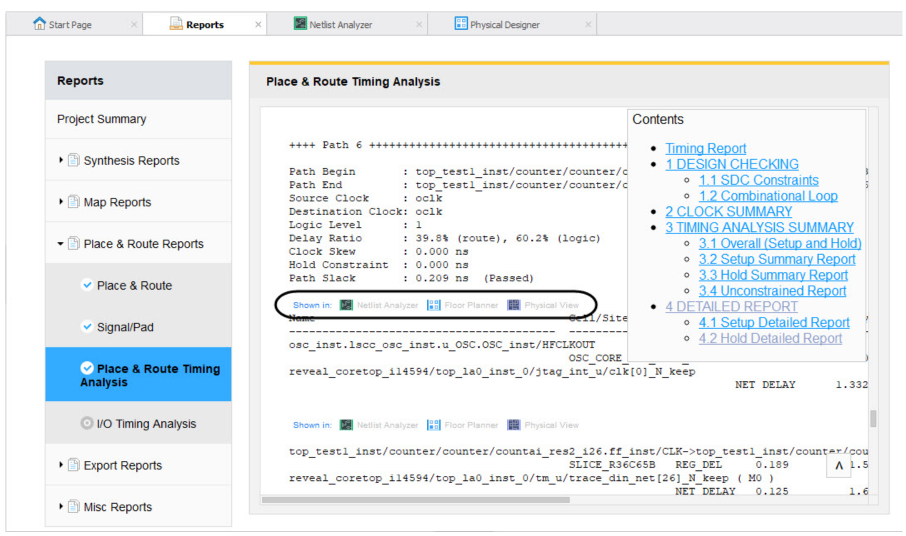
Figure 36: Available tools for Path Cross-Probing

Click on an icon and the tool opens with the selected path.

In some cases, the tool is unable to find the path. The message "Can't show the schematic of this timing path." appears. In an encrypted design, in some cases, cross-probing is not available. The message "Cannot open encrypted design." appears.

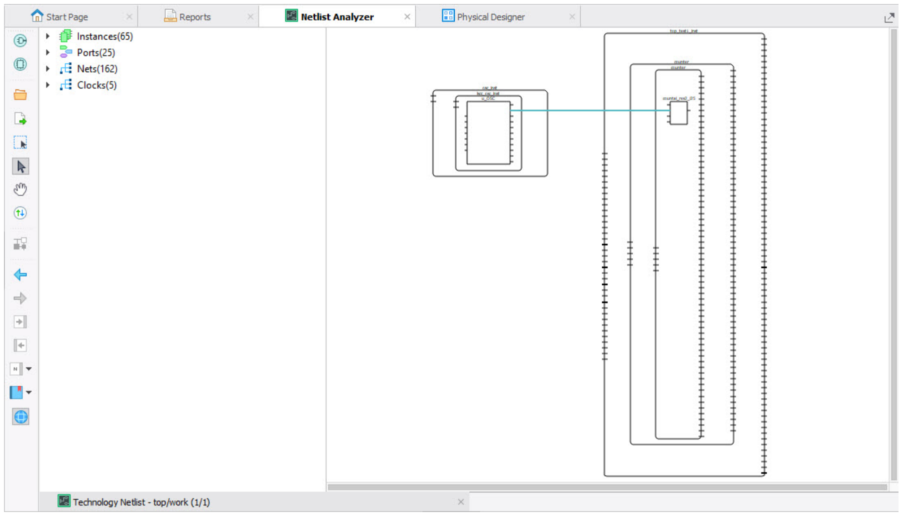
The following figures show cross-probing a path from the Place & Route Timing Analysis report to Netlist Analyzer, Physical Designer's Floorplan View, and Physical Designer's Physical View.

Figure 37: Path Cross-Probing in Reports



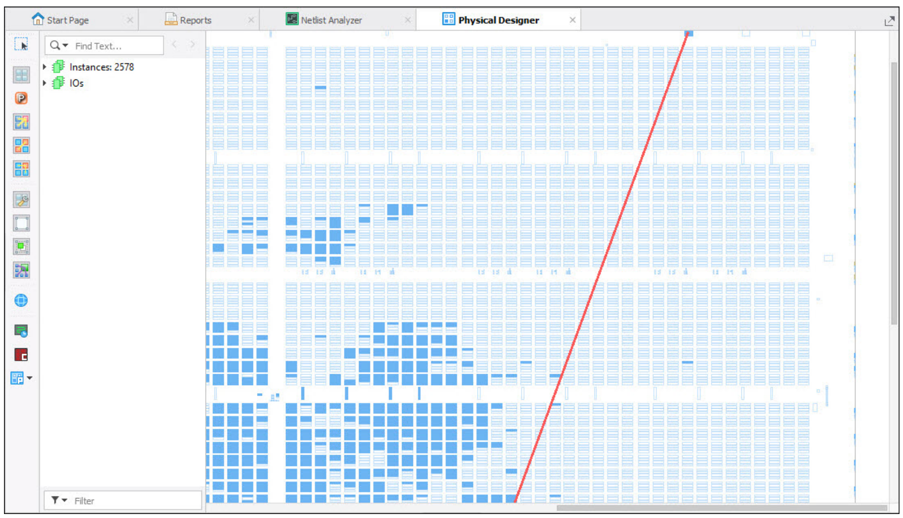
By clicking on the Netlist Analyzer icon, you can preview the data path in Netlist Analyzer.

Figure 38: Path Cross-Probing in Netlist Analyzer



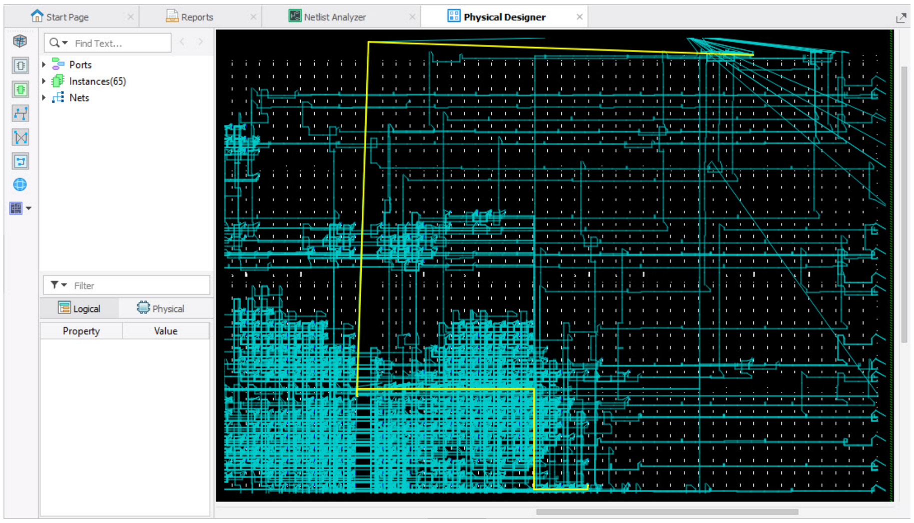
Similarly, by clicking on the Floor Planner icon, you can easily view the same path in Physical Designer's Placement Mode.

Figure 39: Path Cross-Probing in Physical Designer's Placement Mode



The same path is viewable in Physical Designer's Routing Mode by clicking on the Physical View icon in a timing report.

Figure 40: Path Cross-Probing in Physical Designer's Routing Mode



Chapter 5

Working with Projects

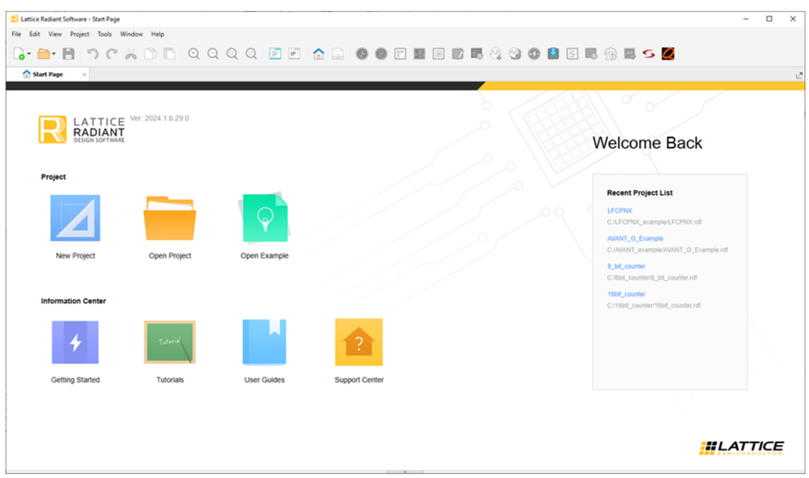
This chapter covers projects and their elements. Implementations and strategies are explained and some common project tasks are shown.

Overview

A project is the top organizational element in the Radiant software design environment. Projects consist of design, constraint, configuration and analysis files. Only one project can be open at a time, and a single project can include multiple design structures and tool settings.

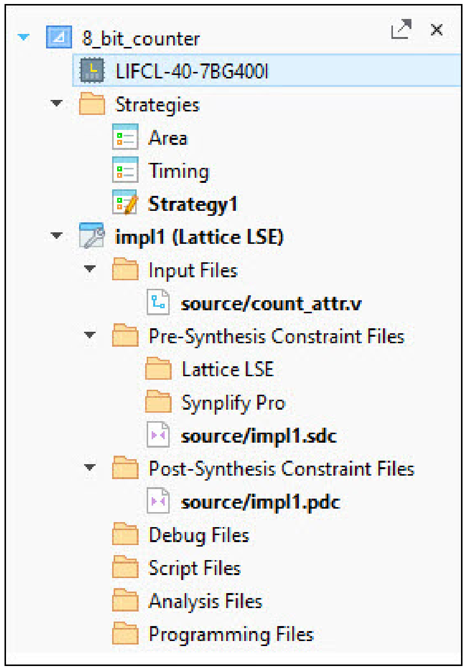
You can create, open, or import a project from the Start Page. Refer to "Getting Started" on page 11 for instructions on creating a new project.

Figure 41: Default Start Page



The File List view shows a project and its main elements.

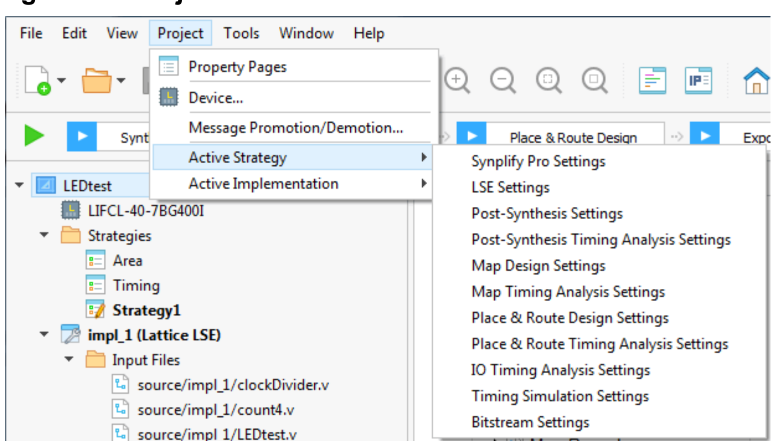
Figure 42: Project Files in File List



The Project menu commands enable you to do the following:

- Examine the project properties.
- Change the target device.
- Change the severity level of warning messages.
- Set the synthesis tool.
- Show the active strategy tool settings.
- Set the top level design unit.

Figure 43: Project Menu



Implementations

An implementation is the structure of a design and can be thought of as *what* is in the design. For example, one implementation might use inferred memory while another implementation uses instantiated memory. Implementations also define the constraint and analysis parameters for a project.

There can be multiple implementations in a project, but only one implementation can be active at a time. And there must be one active implementation. Every implementation has an associated active strategy. Strategies are a shared pool of resources for all implementations and are discussed in the next section. An implementation is created whenever you create a new project.

Implementations consist of the following files:

- Input files
- Pre-Synthesis constraint files
- Post-Synthesis constraint files
- Debug files
- Script files
- Analysis files
- Programming files

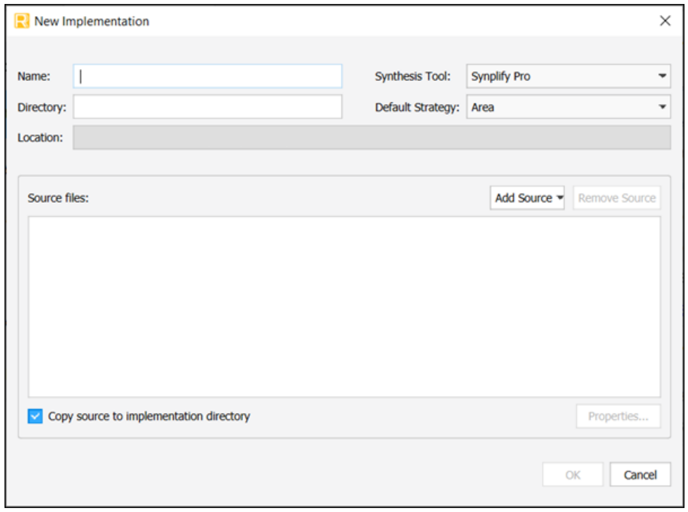
Adding Implementations

To add a new implementation to an existing project:

1. Right-click the project name in the File List project view.

Select **Add > New Implementation**. In the New Implementation dialog box, you can set the implementation name, directory, default strategy, and add source files. When you select **Add Source** you have a choice of browsing for the source files or using a source from an existing implementation.

Figure 44: New Implementation



Notice that you have the option to "Copy source to implementation directory." If this option is selected, the source files will be copied from the existing implementation to the new implementation, and you will be working with different source files in the two implementations. If you want the two implementations to share the same source files and stay in sync, make sure that this option is not selected.

To make an implementation active, right-click its name in the File List and choose **Set as Active Implementation**.

To add a file to an implementation, right-click the implementation name or any file folder in the implementation and choose **Add > New File** or **Add > Existing File**.

Cloning Implementations

To clone an implementation:

1. In the 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 to new implementation source directory**

  The new implementation will have its own copies that can be changed without effecting the original implementation.

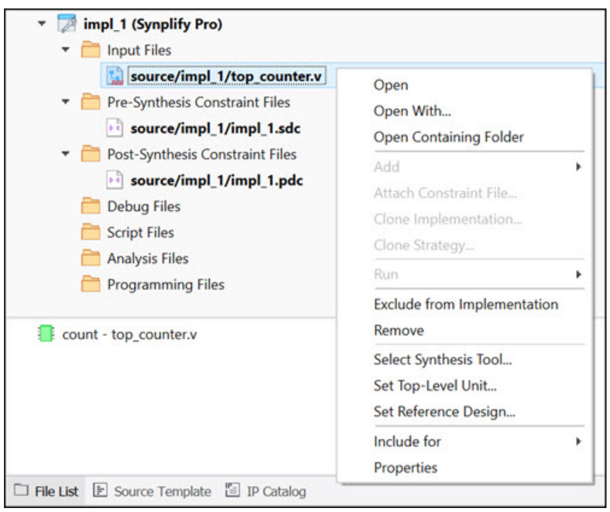
5. The Synthesis Tool text box specifies the currently selected synthesis tool. Go to **Project** > **Active Implementation** > **Select Synthesis Tool** to update your selection.
6. The Default Strategy text box specifies the currently selected default strategy.
7. Click **OK**.

Input Files

Input files are the design source files for the project. Input files can be any combination of Verilog, SystemVerilog, and VHDL.

Right-click an input file name to open a pop-up menu of possible actions for that file.

Figure 45: Input File Actions



You can use the “Include for” commands to specify that a source file be included for both synthesis and simulation, synthesis only, or simulation only.

Pre-Synthesis Constraint Files

Synopsys timing constraints are specified in the new .fdc file format. Legacy .sdc formats are still supported in the Radiant software and Synopsys has provided a script called sdc2fdc, which does a one-time conversion of .sdc files to the new .fdc format. More information about this script can be found in the Synplify Pro release notes.

An .fdc file can be added to an implementation if the selected synthesis tool is Synplify Pro. When using Synplify Pro or the Lattice Synthesis Engine (LSE), the constraints files can be saved as an .sdc file. If the selected synthesis tool is LSE, a Lattice design constraint (.ldc) synthesis file can be added. Constraints in the .ldc file use the Synopsys constraint format.

An implementation can have multiple synthesis constraint files. Only one synthesis constraint file can be active at a time. Unlike Post-Synthesis constraints, a synthesis constraint file must be set as active by the user.

Post-Synthesis Constraint Files

Post-Synthesis constraint files (.pdc) contain both timing and non-timing constraint .pdc source files for storing logical timing/physical constraints. Constraints that are added using the Device Constraint Editor are saved to the active .pdc file. The active post-synthesis design constraint file is then used as input for post-synthesis processes.

An implementation can have multiple .pdc files, but only one can be active at a time.

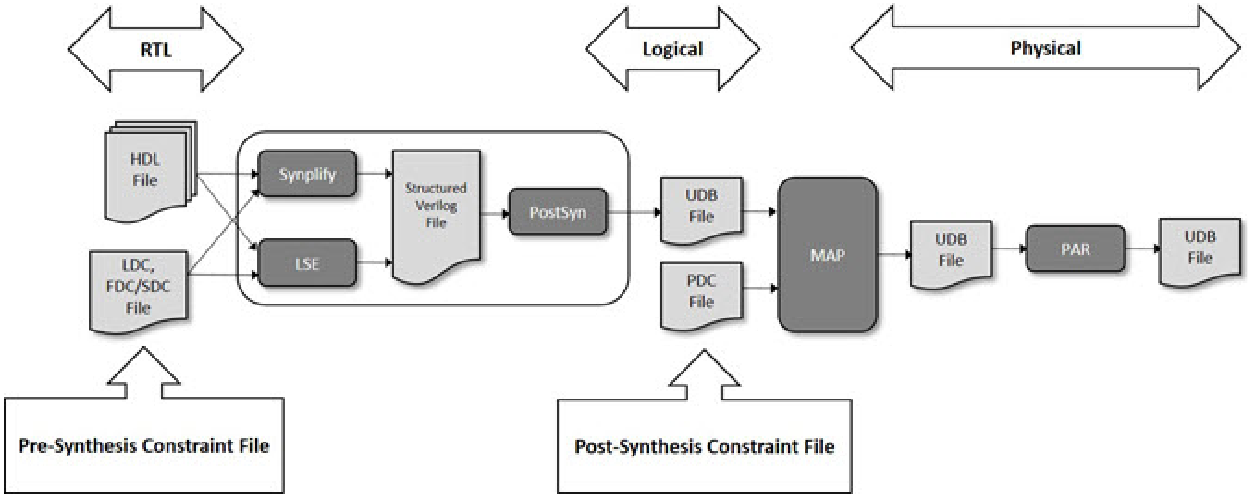
Figure 46: Sample .pdc File

impl1.pdc

```
ldc_set_location -site P5 [get_ports clk]
#ldc_create_region -name RgnA -site R8C2D -width 18 -height 12
#ldc_create_group -name UgrpA [get_cells counter1]
#ldc_set_location -region RgnA [ldc_get_groups UgrpB]
#ldc_create_group -name UgrpC -bbox {2 3} [get_cells counter3]
#ldc_set_location -site R8C25D [ldc_get_groups UgrpC]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count2t[2]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count2t[3]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count2t[0]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count2t[1]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count3t[0]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count3t[1]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count3t[2]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count3t[3]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count3t[4]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count3t[5]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count3t[6]}]
ldc_set_port -iobuf {IO_TYPE=LVCMOS25} [get_ports {count3t[7]}]
#ldc_set_location -site 35 [get_ports clk1]
create_clock -period 15 [get_nets counter1/clk1_c]
```

Figure 47 shows a high-level flow of how constraints from multiple sources can be used and modified in the Radiant software.

Figure 47: Radiant software Constraints Flow Chart



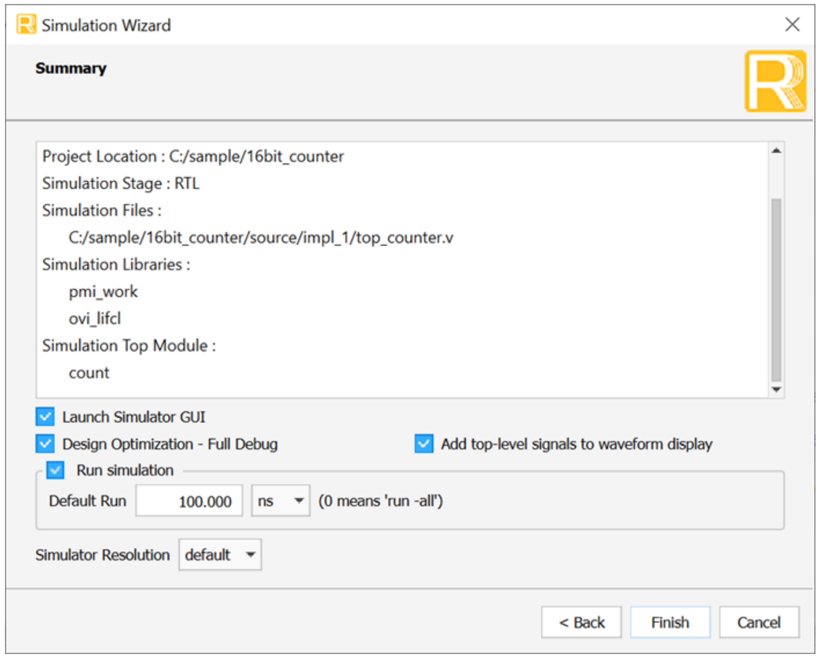
Debug Files

The files in the Debug folder are project files for Reveal Inserter. They are used to insert hardware debug into your design. There can be multiple debug files, and one can be set as active. To insert hardware debug into your design, right-click a debug file name and choose **Set as Active Debug File** from the pop-up menu. The debug file name becomes bold, indicating that it is active. It is not required to have an active debug file.

Script Files

The Script Files folder contains the scripts that are generated by the Simulation Wizard. After you run the Simulation Wizard, the steps are stored in a simulation project file (.spf), which can be used to control the launching of the simulator.

Figure 48: Simulation Script File



Analysis Files

The Analysis Files folder contains Power Calculator files (.pcf). The folder can contain multiple analysis files, and one (or none) can be set as active. The active or non-active status of an analysis file affects the behavior of the associated tool view.

Programming Files

Programming files (.xcf) are configuration scan chain files used by the Radiant Programmer for programming devices. The .xcf file contains information about each device, the data files targeted, and the operations to be performed.

An implementation can have multiple .xcf files, but only one can be active at a time.The file must be set as active by the user.

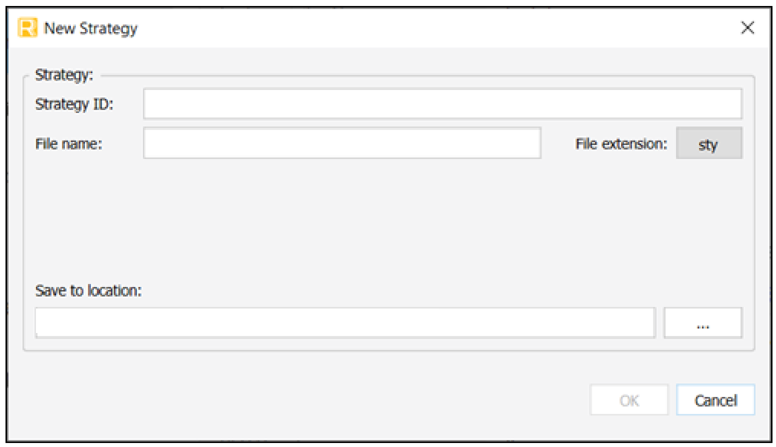
Strategies

Strategies are collections of all the implementation-related tool settings in one convenient location. Strategies can be thought of as recipes for how the design will be implemented. An implementation defines *what* is in the design, and a strategy defines *how* that design will be run. There can be many strategies, but only one can be active at a time. There must be one active strategy for each implementation.

The Radiant software provides two predefined strategies: Area and Timing. It also enables you to create customized strategies. Predefined strategies cannot be edited, but they can be cloned, modified, and saved as customized user strategies. Customized user strategies can be edited, cloned, and removed. All strategies are available to all of the implementations, and any strategy can be set as the active one for an implementation.

To create a new strategy from scratch, choose **File > New > Strategy**. In the New Strategy dialog box, enter a name for the new strategy. Specify a file name for the new strategy and choose a directory to save the strategy file (.sty).

Figure 49: Creating a New Strategy from Scratch

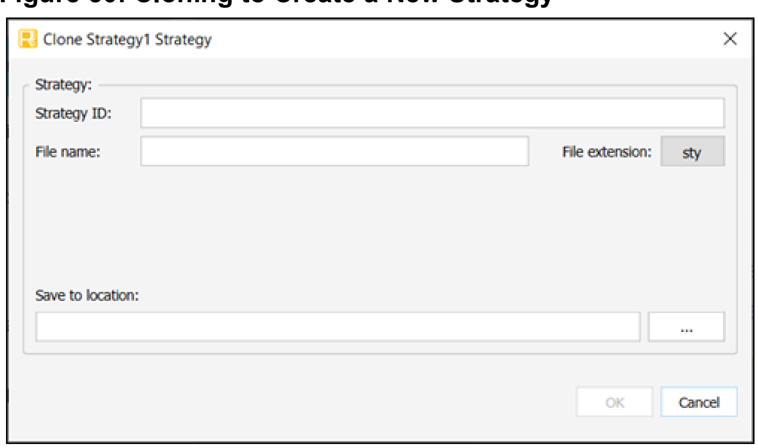


The new strategy is with all the default settings of the current design. You can modify its settings in the Strategies dialog box.

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

To create a new strategy from an existing one, right-click the existing strategy and choose **Clone <*strategy name*> Strategy**. Set the new strategy's ID and file name.

Figure 50: Cloning to Create a New Strategy



To make a strategy active, right-click the strategy name and choose **Set as Active Strategy**.

To change the settings in a strategy:

1. Double-click the strategy name in the File List view
2. Select the option type to modify
3. Double-click the Value of the option to be changed

The default values are displayed in plain blue text. Modified values are displayed in italic bold text.

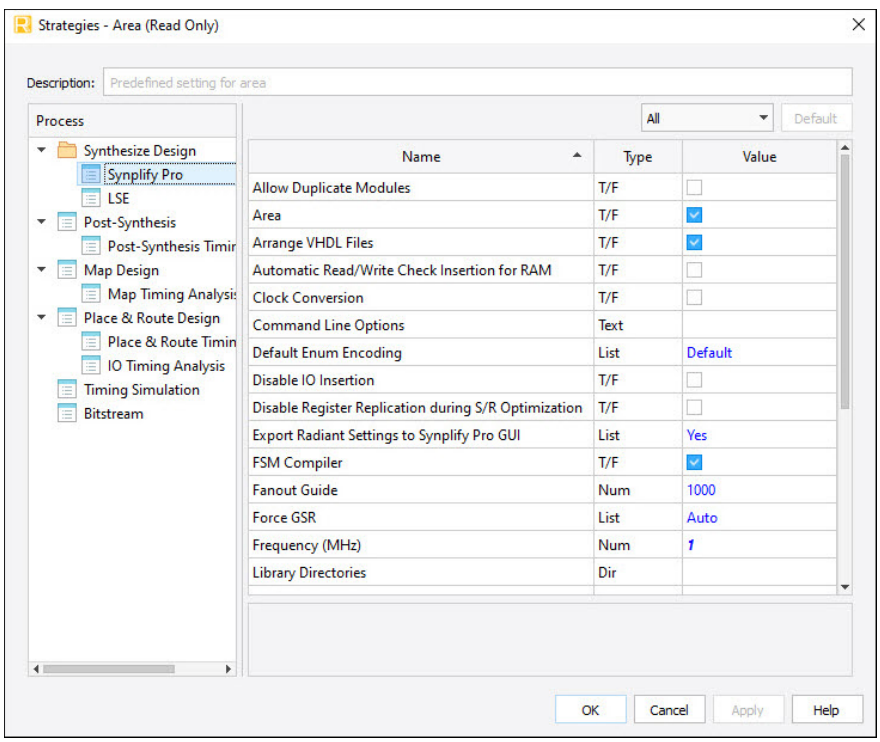
Strategies are design data independent and can be exported and used in multiple projects.

Area

The Area strategy is a predefined strategy for area optimization. Its purpose is to minimize the total logic gates used while enabling the tight packing option available in Map.

Applying this strategy to large and dense designs might cause difficulties in the place and route process, such as longer time or incomplete routing.

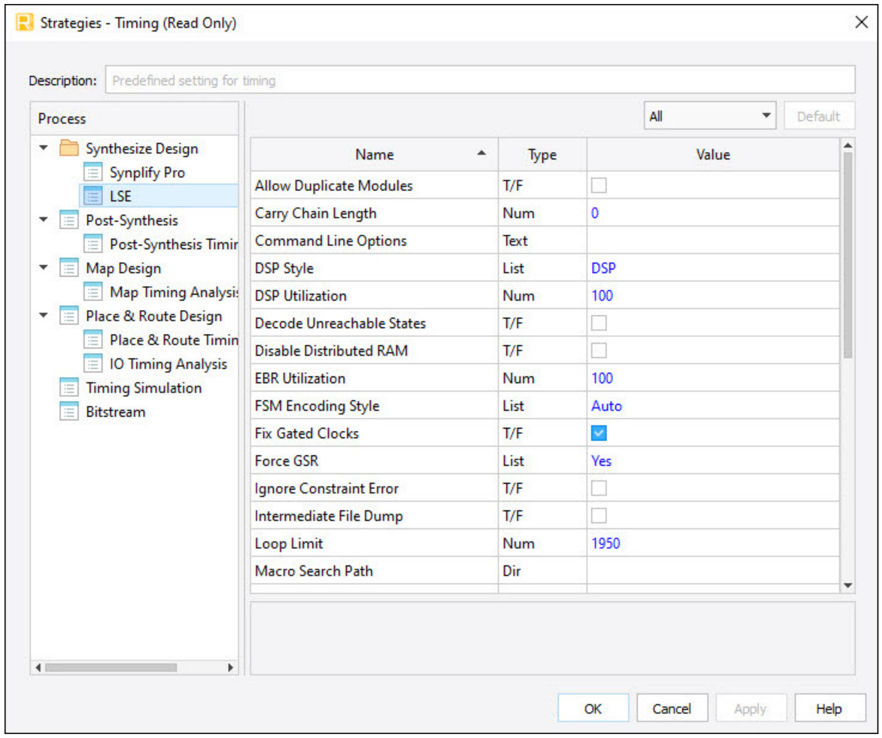
Figure 51: Area Predefined Strategy



Timing

The Timing strategy is a predefined strategy for timing optimization. Its purpose is to achieve timing closure. The Timing strategy uses a very high effort level in placement and routing. Use this strategy if you are trying to reach the maximum frequency on your design. If you cannot meet your timing requirements with this strategy, you can clone it and create a customized strategy with refined settings for your design. This strategy might increase your place-and-route run time compared to the Area strategy.

Figure 52: Timing Predefined Strategy



User-Defined

You can define your own customized strategy by cloning and modifying any existing strategy. You can start from either a predefined or a customized strategy.

Constraint Propagation Options

This page lists all the strategy options associated with the Constraint Propagation process.

Propagation Debug Mode

Using Propagation Debug Mode can affect timing of other circuits and can lead to false timing reports.

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 **Synopsys 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 (MHz) (for Synplify Pro)" on page 60.

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.

Auto Compile Point

With Automatic Compile Points, synthesis decides based on existing hierarchy and utilization estimates.

Compile Points minimize the amount of netlist changes resulting from RTL code changes. Compile points create logical boundaries where optimization will not occur. This may sacrifice some design performance for design result predictability. No user intervention is needed aside from enabling the flow.

Note:

Auto Compile Point only supports the LAV-AT device.

Automatic Read/Write Check Insertion for RAM

When this option is set in the strategy, the synthesis tool inserts glue logic around inferred RAM to avoid simulation mismatches caused by indeterminate output values when the read and write addresses are same. By default this option is OFF.

You should design to make sure the read and write addresses are never the same.

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**.

For example:

```
set_option -library_path c:/source
```

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.

Disable Register Replication During S/R Optimization

(Not applicable for iCE40 UltraPlus): When this option is set in the strategy, the synthesis tool will NOT duplicate the registers while inferring the address pointers for the RAM during Shift-register inference. By default this option is OFF and the tool will duplicate the address pointer registers to get better performance.

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, refer to the Radiant Software Help. See **User Guides > Implementing the Design > Synthesizing the Design > Interactive Synthesis**.

FDC Constraint Files

Specifies the Synopsys FPGA Design Constraint (FDC) files. The FDC file contains Synopsys SDC Standard timing constraints, such as create_clock, set_input_delay, and set_false_path, along with the non-timing constraints (design constraints), such as define_attribute, define_scope_collection, and define_io_standard.

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:

- False (default) – Does not infer Global Set/Reset in your design.
- Auto – Allows the software to decide whether to 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 (MHz) (for Synplify Pro)

Specifies the global design frequency (in MHz). Nothing in the Value column means "auto" and Synplify Pro will try to maximize the frequency of the clocks.

The setting is ignored when “Area (for Synplify Pro)” on page 57 is set to True.

This option is equivalent to the “set_option -frequency <*number*> | auto” command in Synplify Pro.

Library Directories

Specifies all the paths to the directories which contain the Verilog library files to be included in your design for the project.

You can also add custom library files with module definitions for the design in a single file. The names of files read from the library path must match module names. Mismatches result in error messages.

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.

Output Netlist Format (for Synplify Pro)

Outputs a mapped VHDL netlist for post-synthesis simulation.

Available options are: None (default) and VHDL.

This option is equivalent to the “set_option -write_vhdl 1 | 0” 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.

Resynthesize All

When this is set to True (default), Synplify Pro will resynthesize all portions, modules, or files of the RTL source. When this is set to False, Synplify Pro will synthesize only the updated portions, modules or files of the RTL source since the last run result.

If you run the “Force Run From Start' process, all portions, modules, or files of the RTL source will be resynthesized, regardless of whether this option is set to True or False.

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.

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.

Allow Mixed Assignments (for LSE)

When set to True, having both blocking and non-blocking assignments to the same variable will be allowed. A warning will be issued instead of an error.

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, refer to the Radiant Software Help. See **Reference Guides > Command Line Reference Guide**

> Command Line Tool Usage > Running SYNTHESIS from the Command Line.

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.

Force GSR (for LSE) (Not applicable for iCE40 UltraPlus)

Forces Global Set/Reset Pin usage. (Not applicable for iCE40 UltraPlus).

Available options are:

- No (default) – Does not infer Global Set/Reset in your design.
- Auto – Allows the software to decide whether to infer Global Set/Reset in your design.
- Yes – Always infers Global Set/Reset in your design.

Ignore Constraint Errors

Enables or disables the LSE synthesis process to ignore constraint errors.

By default, the box is unchecked (False), and LSE terminates processing and issue an error message when a constraint errors are encountered.

When the box is checked (True), LSE will ignore constraint errors and continue processing.

The LSE synthesis report includes the setting of this option and the constraint errors found.

Ignore Constraint Errors

Enables or disables the LSE synthesis process to ignore constraint errors.

By default, the box is unchecked (False), and LSE terminates processing and issue an error message when constraint errors are encountered.

When the box is checked (True), LSE will ignore constraint errors and continue processing.

The LSE synthesis report includes the setting of this option and the constraint errors found.

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 do not need to specify a search path if the necessary 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:

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.

Optimization Goal

Enables LSE to optimize the design for area or speed.

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” on page 70 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 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” on page 69 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” on page 70 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, refer to the Radiant Software Help. See **User Guides > Implementing the Design > Synthesizing the Design > Optimizing LSE for Area and Timing**.

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.
- 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.

Read Write Check on RAM

When set to True, LSE inserts additional glue logic to the RAM read/write port to resolve read/write conflicts. The default is False.

A warning message indicating a potential simulation mismatch will be issued if RAM has a read/write conflict.

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.

Report Trimmed User Nets (for LSE)

When set to True, LSE will report whenever a user-declared HDL net is trimmed from the netlist.

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.

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" on page 67 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 do control packing on individual registers and ports. Refer to the Radiant Software Help. See **Reference Guides > Constraints Reference Guide > Lattice Synthesis Engine Constraints > Lattice Synthesis Engine-Supported HDL Attributes > syn_useioff.**

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.

Command Line Options (for Post-Synthesis)

Enables additional command line options for the associated process.

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.

Hardware Evaluation

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.

Note:

The Hardware Evaluation option is only for a device that does not support bitstream encryption ability even with a built-in hardware evaluation timer (i.e., LAV-AT device).

Post-Synthesis Timing Analysis Options

This page lists all strategy options associated with the Post-Synthesis Timing Analysis process.

Command Line Options (for Post-Synthesis Timing Analysis)

Enables additional command line options for the associated process.

Number of End Points

Controls the number of endpoints in the critical endpoint summary.

Number of Paths Per Constraint (for Post-Synthesis Timing Analysis)

Lists detailed logic and route delays for all constrained paths and nets in the design.

Number of Paths Per Endpoint (for Post-Synthesis Timing Analysis)

Controls maximum number of paths that could be reported for each endpoint.

Number of Unconstrained Paths (for Post-Synthesis Timing Analysis)

Reports paths not covered by a timing preference.

Report Format (for Post-Synthesis Timing Analysis) Specifies the report format type.

- Lattice Standard – Displays timing report format that is similar to formats used in many industry timing tools.
- Diamond Style – Displays timing report that is similar to Diamond software TRACE report.

Timing Analysis Options (for Post-Synthesis 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.

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” on page 197.

Ignore Constraint Errors

Enables or disables the Map Design process to ignore constraint errors.

By default, the box is unchecked (False), and Map Design terminates processing and issues an error message when constraint errors are encountered.

When the box is checked (True), Map Design will ignore constraint errors and continue processing.

The Map Design process report includes the setting of this option and the constraint errors found.

Infer GSR

(Not applicable for iCE40 UltraPlus): Enables or disables the GSR inferencing.

GSR inference is only applicable to PLC slice registers by default. Each black-box can have its own rules to guide mapper to perform GSR inference. Rules to be applied according to device architecture specification. For example, in LIFCL device, it is applicable to SLICE / IO registers, block RAM, large RAM, and DDR components.

When multiple GSR instances are found in the design, mapper is able to merge them if their outputs are actually the same wire in the netlist. Otherwise mapper will issue an error.

Mapper can also create a GSR component if user specified the GSR signal in the constraint file.

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.

Map Timing Analysis Options

This page lists all strategy options associated with the Map Timing Analysis process.

Command Line Options (for Map Timing Analysis)

Enables additional command line options for the associated 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 Format (for Map Timing Analysis)

Specifies the report format type.

- Lattice Standard – Displays timing report format that is similar to formats used in many industry timing tools.
- Diamond Style – Displays timing report that is similar to Diamond software TRACE report.

Speed for Hold Analysis

Specifies performance grade for hold analysis. This option allows you to override the default m (minimum) performance grade for hold time analysis, which represents faster silicon than the fastest performance grade of the device being targeted.

Speed for Setup Analysis

Specifies performance grade for setup analysis. This option allows you to override the default performance grade which runs setup analysis against the performance grade of the device currently targeted by the project implementation.

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, refer to the Radiant Help. See **Reference Guides > Command Line Reference Guide > Command Line Tool Usage > 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.

Impose Hold Time Correction

Forces PAR to fix hold time violations even if there are setup time violations.

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, refer to the Radiant Software Help. See **User Guides > Implementing the Design > Place and Route > Running Multiple PAR Jobs in Parallel**.

Number of Host Machine Cores

Allows you to run multiple threads on your local machine by specifying how many cores to be used from your local machine. The range is from 0 to 64.

Pack Logic Block Utility

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 a percentage that depends on which device is selected. For example, for LIFCL, the default is 75 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 placer 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 placer 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 placer 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.

Note:

The Pack Logic Block Utility and Path-based Placement strategies are not supported in the LAV-AT-E device.

Placement Iteration Start Point

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.

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.

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.

Set Speed Grade for Setup Optimization

Change performance grade for setup optimization.

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.

Command Line Options (for Place & Route Timing Analysis)

Enables additional command line options for the associated 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 Format (for Place & Route Timing Analysis)

Specifies the report format type.

- Lattice Standard – Displays timing report format that is similar to formats used in many industry timing tools.
- Diamond Style – Displays timing report that is similar to Diamond software TRACE report.

Speed for Hold Analysis

Specifies performance grade for hold analysis. This option allows you to override the default m (minimum) performance grade for hold time analysis, which represents faster silicon than the fastest performance grade of the device being targeted.

Speed for Setup Analysis

Specifies performance grade for setup analysis. This option allows you to override the default performance grade which runs setup analysis against the performance grade of the device currently targeted by the project implementation.

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.

IO Timing Analysis Options

The following option is associated with the IO Timing Analysis process.

All Performance Grade

Controls whether the I/O timing report (.ior) will give an in-depth analysis on all available performance grades or will just produce a summary report on the worst-case performance grade.

- True - The I/O Timing Report will summarize the worst-case scenario of all available performance grades.
- False (default) - The I/O Timing Report will only contain a summary of the worst-case performance grade for the given device.

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.

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 Check With 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).

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 the following two special characters as the hierarchy separator: “/” (back-slash) or “.” (period).

Enter the character as is in the edit box. 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.

Bitstream Mode

Generates bitstream for use in special bitstream update flow.

Command Line Options

Enables additional command line options for the associated process.

ES Parts

Generates SMM key per device and introduce a new bitgen flag to specify the ES parts

Enable Early IO Wakeup

Enable IO output as soon as possible. (Not applicable for iCE40 UltraPlus)

Enable Timing Check

When this is set to true and there is timing error in the Place and Route report file, a dialogue box will display before executing bitstream generation.

Enable Warm Boot

Enables the Warm Boot functionality, provided the design contains an instance of the WARMBOOT primitive. (iCE40UP)

Initialize EBR Quadrant 0

Write the EBR initialization data into the bitstream for quadrant 0. (iCE40UP)

Initialize EBR Quadrant 1

Write the EBR initialization data into the bitstream for quadrant 1. (iCE40UP)

Initialize EBR Quadrant 2

Write the EBR initialization data into the bitstream for quadrant 2. (iCE40UP)

Initialize EBR Quadrant 3

Write the EBR initialization data into the bitstream for quadrant 3. (iCE40UP)

IP Evaluation

When enabled, a bitstream will be generated for evaluation purposes when an IP license is not found, but will be limited to a maximum of four hours before the device resets itself. When disabled, a bitstream will not be generated if an IP license is not found (Not applicable for iCE40 UltraPlus).

No header

Do not write the bitstream header section. (iCE40UP)

Set All Unused IO No Pullup

Removes the pullup on the unused I/Os, except Bank 3 I/Os which do not have pullup. (iCE40UP)

Oscillator Frequency Range

Options include Medium and Slow. Fast is not supported for iCE40UP. Only for NVCM configuration, not for programming

Depending on the speed of external PROM, this options adjusts the frequency of the internal oscillator used by the iCE40UP device during configuration. This is only applicable when the iCE40UP device is used in SPI Master Mode for configuration.

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 (.bin) 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.
- Hex File (Headecimal) – Generates a Hexadecimal (.hex) PROM data file used for Programming into external non-volatile memory, such as parallel or Serial Peripheral Interface (SPI) Flash devices (iCE40UP only).
- NVCM File (Non-volatile Configuration Memory) – Generates a Non-volatile Configuration Memory (.nvcm). Each line in this file is a separate NVCM instruction for bitmap programming. The exceptions are comment lines (usually the first few lines in the file) which start with the # sign. Comment lines should be ignored when programming the NVCM array. The comment lines contain comments as well as header information (iCE40UP only).

Register Initialization

Enable register initialization section of the bitstream (Not applicable for iCE40 UltraPlus).

Run DRC (iCE40UP only)

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.

SPI Flash Low Power Mode

Places the PROM in low-power mode after configuration. (iCE40UP)

This option is applicable only when the iCE40UP device is used as SPI Master Mode for configuration.

Set NVCM Security

Ensures that the contents of the Non-Volatile Configuration Memory (NVCM) are secure and the configuration data cannot be read out of the device. (iCE40UP)

Common Tasks

Working with projects includes many tasks, including: creating the project, editing design files, modifying tool settings, trying different implementations and strategies, and saving your data.

Creating a Project

See “Creating a New Project” on page 12 for step-by-step instructions.

Changing the Target Device

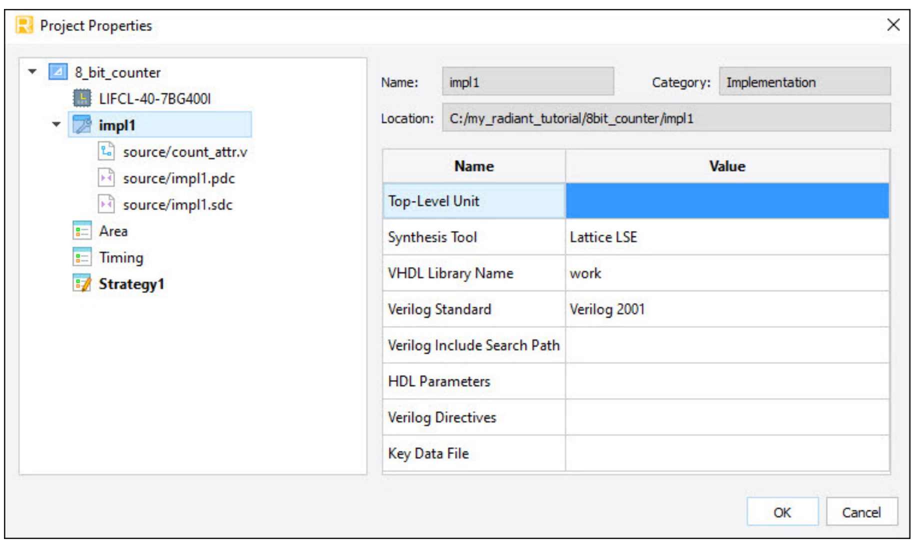
There are two ways to access the Device Selector dialog box for changing the target device:

- Double-click the device in the project File List view or right-click it and choose **Edit**.
- Choose **Project > Device**.

Setting the Top Level of the Design

If multiple top levels exist in the hierarchy of your HDL source files, you will need to set the top-level design unit. After generating the hierarchy, choose **Project > Active Implementation > Set Top-Level Unit**. Alternatively, right-click the implementation and choose this command from the pop-up menu.

Figure 53: Top-Level Design Unit



In the Project Properties dialog box, select **Value** next to **Top-Level Unit** and select the desired top level from the list.

You can also use the Hierarchy View to set the top-level. Right-click a level you want to be the top-level in the Hierarchy View and choose **Set Top-Level Unit**.

Editing Files

You can open any of the files for editing by double-clicking or by right-clicking and choosing **Open** or **Open with**.

Saving Project Data

In the File menu are the following selections for saving your design and project data:

- Save – saves the currently active item.
- Save As – saves the active item using a different file name.
- Save All – saves all changed documents.
- Save Project – saves the current project.
- Save Project As – saves the active project using a different project name.
- Archive Project – creates a zip file of the current project in a location you specify.

Chapter 6

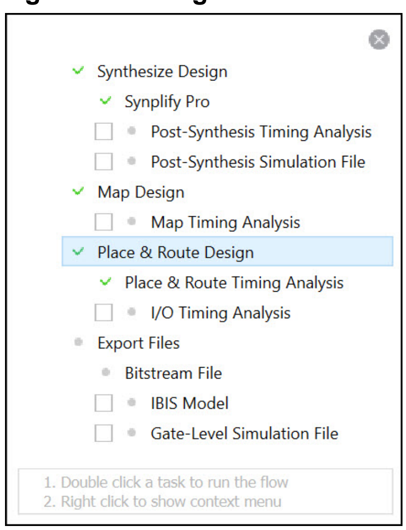
Radiant Software Design Flow

This chapter describes the design flow in the Radiant software. Running processes and controlling the flow for alternate what-if scenarios are explained.

Overview

The FPGA implementation design flow in the Radiant software provides extensive what-if analysis capabilities for your design. The design flow is displayed in the Task Detail View at the right end of the Process Toolbar.

Figure 54: Design Flow Shown in Task Detail View



Design Flow Processes

The design flow is organized into discrete processes, where each step allows you to focus on a different aspect of the FPGA implementation.

Synthesize Design

This process runs the selected synthesis tool (Lattice Synthesis Engine is the default) in batch mode to synthesize your HDL design.

- Synthesis Tool - identifies the selected synthesis tool, Synplify Pro (default) or Lattice Synthesis Engine.
- Post-Synthesis Timing Analysis - generates timing analysis files.
- Post-Synthesis Simulation File - generates a post-synthesis netlist file <*file_name*>_syn.vo used for Post-Synthesis Simulation.

Map Design

This process maps the design to the target FPGA and produces a mapped Unified Database (.udb) design file. Map Design converts a design's logical components into placeable components.

- Map Timing Analysis - generates timing analysis files.

Place & Route Design

This process takes mapped physical design files and places and routes the design. The output can be processed by the design implementation tools. Timing analysis files can also be generated.

- Place & Route Timing Analysis - generates timing analysis files.
- I/O Timing Analysis - generates I/O timing analysis files.

Export Files

This process generates the IBIS, simulation, and programming files that you have selected for export:

- Bitstream File – generates a configuration bitstream (bit images) file, which contains all of the design's configuration information that defines the internal logic and interconnections of the FPGA, as well as device-specific information from other files.
- IBIS Model – generates a design-specific I/O Buffer Information Specification model file (.ibs). IBIS models provide a standardized way of representing the electrical characteristics of a digital IC's pins (input, output, and I/O buffers).
- Gate-Level Simulation File – generates a Verilog netlist of the routed design that is back annotated with timing information. This generated .vo file enables you to run a timing simulation of your design.

The files for export can also be generated separately by double-clicking each one.

Running Processes

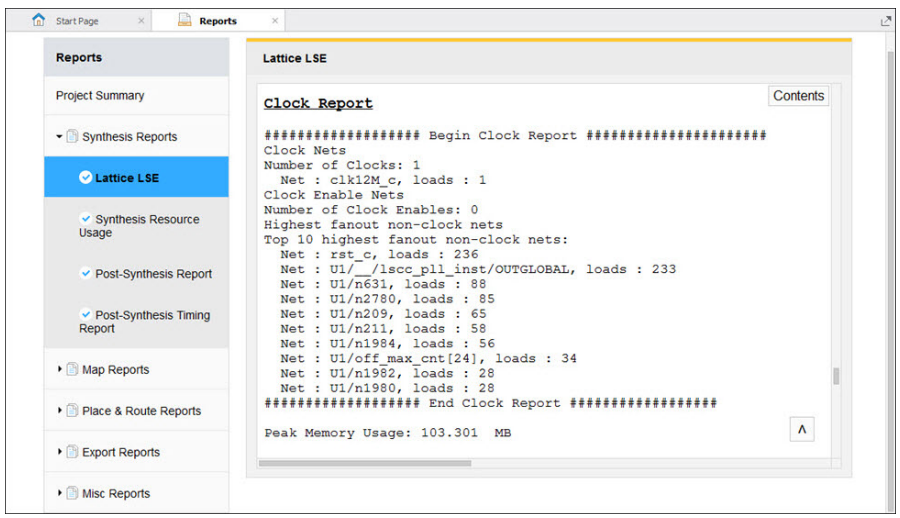
You can perform the following actions for each step in the process flow:

- Run – runs the process, if it has not yet been run.
- Force Run– reruns a process that has already been run.
- Force Run From Start – reruns all processes from the start to the selected process.
- Stop – stops a running process.
- Clean Up Process – clears the process state and puts a process into an initial state as if it had not been run.

The state of each process step is indicated with an icon to the left of the process. The process status icons description is described in “Process” on page 35

The Reports View displays detailed information about the process results, including the last process run. The Messages section shows warning and error messages and allows you to filter the types of messages that are displayed. See “Reports” on page 37.

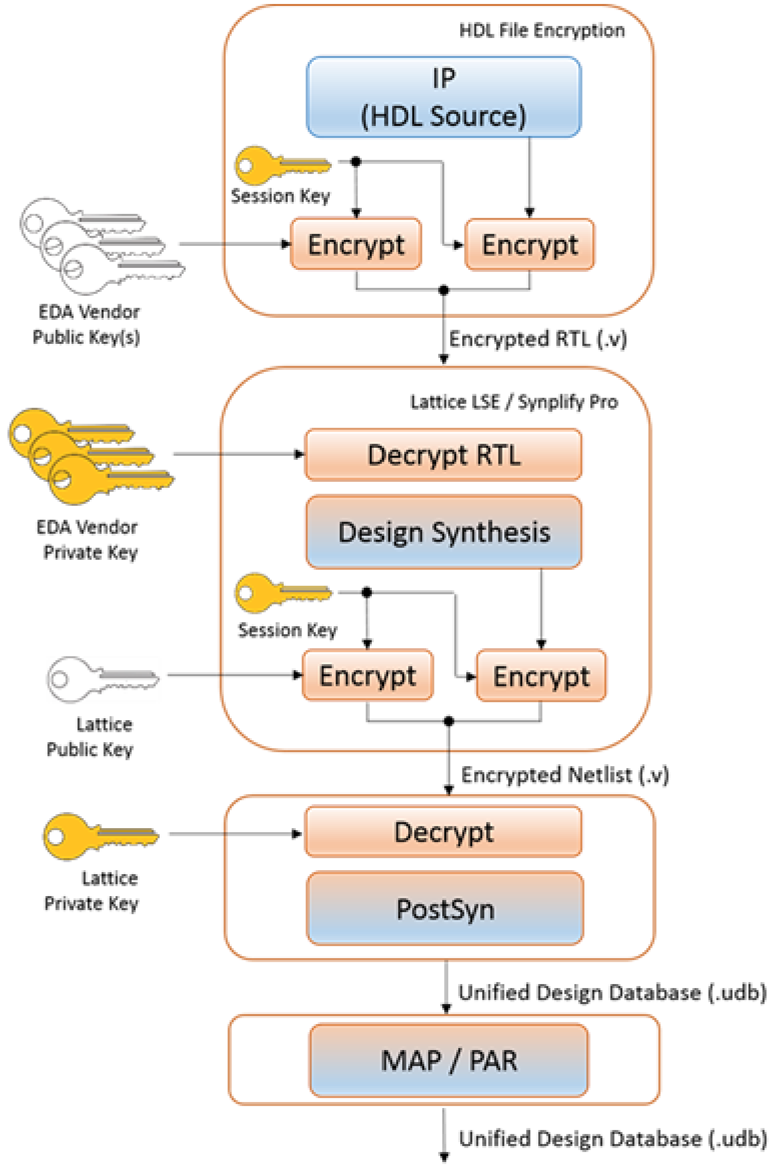
Figure 55: Reports View of Last Process Run



IP Encryption Flow

IP encryption flow enables you to protect your IP design. Following the industry standard, the Radiant software, through the IP encryption flow, allows the partnership between the IP Vendor, supported EDA vendor, and Lattice Semiconductor.

Figure 56: IP Encryption Flow



The encryption flow uses symmetric and asymmetric encryption methods to maximize the design security. The symmetric method only involves a single symmetric key for both, encryption and decryption. The asymmetric method involves the public-private key pair. The public key is published by a vendor and is used by the Radiant software. The private key is never revealed to the public.

The Radiant software supports these cryptographic algorithms:

- AES-128/AES-256: symmetric algorithm used to encrypt the content of the HDL source file.
- RSA-2048: asymmetric algorithm used to obfuscate a key used in HDL file encryption. The RSA is defined by the public-private key pair. You must be familiar with both keys in order to perform RSA decryption.

HDL File Encryption Flow

The current software version supports encryption of a single HDL source file per a single command.

The overall HDL file encryption flow is summarized in these steps:

- The source file of the IP design is AES encrypted using a symmetric session key. The AES encryption uses the CBC-128 or CBC-256 algorithm. In the source files, this section is referred to as a data block.
- The session key is RSA encrypted using the vendor's Public Key. In the source files, this section is referred to as a key block. Multiple key blocks may be present in the source file.
- The encrypted Session key and the encrypted design are merged to a file generally named the Encrypted RTL.

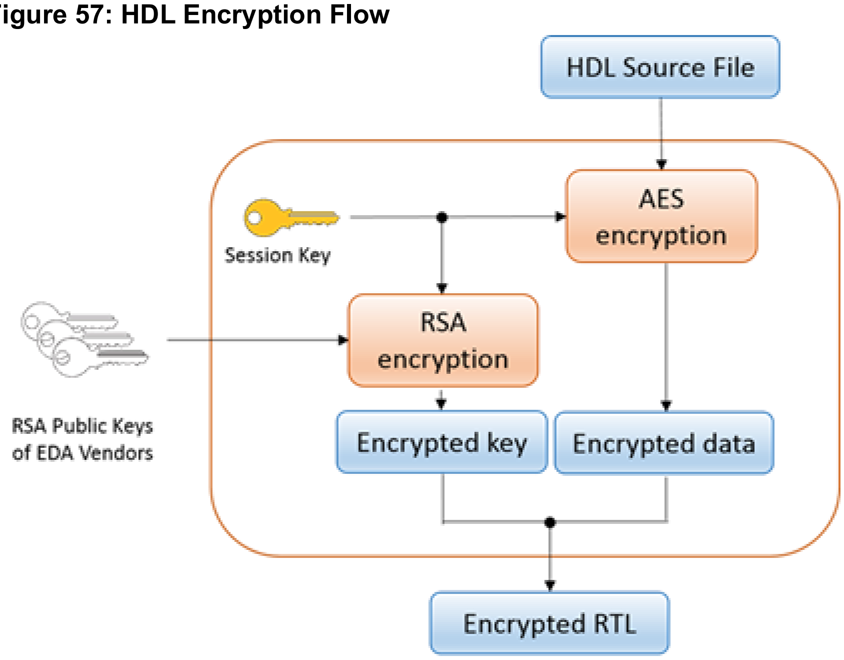
Each encrypted source file contains a single data block and one or more key blocks. The number of key blocks depends on the number of provided vendor's public keys.

Note

To decrypt an encrypted source file, you must perform the IP encryption flow steps in the reverse order.

During the next step in the design flow, typically synthesis, the Encrypted RTL is decrypted to access the original IP design, as shown in the following figure.

Figure 57: HDL Encryption Flow



By separating the encryption of data and key, you can use public keys from different vendors to encrypt the same HDL file.

For more information on how to perform HDL encryption, refer to the Radiant Software Help. See **Reference Guides > Command Line Reference Guide > Command Line Tool Usage > Running HDL Encryption from the Command Line**.

HDL File Encryption Steps

This section provides step-by-step instructions on how to encrypt an HDL file.

The Radiant software provides the key templates you can simply drag-and-drop into an HDL file. Each key template is specific to an EDA vendor providing the value of a public key.

To view the templates in Project Navigator, go to the Source Template tool. Open the **Verilog > Encryption Templates** folder and select the EDA-specific key template.

Currently, the Radiant software supports these encryption templates:

- Lattice Semiconductor
- Synplicity-1
- Synplicity-2
- Mentor
- Synopsys
- Aldec
- Cadence
- Combined Sample: provides an example of file holding multiple keys.

Step 1: Prepare the HDL file.

Annotate the HDL source file with protected pragmas. Protected pragmas provide information regarding the type of the key used to encrypt the HDL file, the name of the key, and the encryption algorithm.

In this example, the HDL source file will be encrypted by the Lattice Public Key.

```
`pragma protect version=1
`pragma protect encoding=(enctype="base64")

// optional information
`pragma protect author="<Your_Name>"
`pragma protect author_info="<Your_Information>"
```

Step 2: Specify the portion of the HDL source file to encrypt.

Annotate the HDL file to specify the encryption. Only the portion defined within these protected pragmas is encrypted.

```
`pragma protect begin
// HDL portion that should be encrypted
`pragma protect end
```

Step 3: Prepare key.

Define the key with which the HDL file should be encrypted. Each key definition must contain the following information:

- key_keyowner: specify the owner of the key
- key_keyname: specify the name of the key. Same owner may provide multiple keys.
- key_keymethod: specify the used cryptographic algorithm. Current version supports RSA algorithm.
- key_public_key: speficy the exact value of the key.

The key definition can be done in two ways:

Defining the key in the key.txt file: The public encryption key or keys can be defined in any .txt file. The key file may contain a single public key or a list of all available public keys. In the Radiant software, all common EDA vendor public keys are located in <*Radiant_install_directory*>/ispfpga/data/key.txt file.

The following is an example of Lattice Public Key defined in key.txt file:

```
`pragma protect key_keyowner= "Lattice Semiconductor"
`pragma protect key_keyname= "LSCC_RADIANT_2"
`pragma protect key_method="rsa"
`pragma protect key_public_key
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0EZKUUhbuB6vSsc7OhQJ
iNAWJR5unW/OWp/LFI71eAl3s9bOYE2OlKdxbai+ndIeo8xFt2btxetUzuR6Srvh
xR2Sj9BbW1QToo2u8JfzD3X7AmRvlwKRX8708DPo4LDHZMA3qh0kfDDWkp2Eausf
LzE2cVxgq7fy/bDhUeN8xKQCSKJ7aguG6kOI6ROoZz21ljzDLUQzhm2qYF8SpU1o
tD8/uw53wLfSuhR3MBOB++xcn2imvSLqdgHWuhX6CtZIx5CD4y8inCbcLy/0Qrf6
sdTN5SAg2OZhjeNdzmqSWqhL2JTDw+Ou2fWzhEd0i/HN0y4NMr6h9fNn8nqxRyE7
IwIDAQAB
```

Defining the key directly in the HDL file: You may define the Public Key directly in the HDL file. You may define one or more keys.

```
`pragma protect key_keyowner= "Lattice Semiconductor"
`pragma protect key_keyname= "LSCC_RADIANT_2"
`pragma protect key_method="rsa"
`pragma protect key_public_key
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0EZKUUhbuB6vSsc7OhQJ
iNAWJR5unW/OWp/LFI71eAl3s9bOYE2OlKdxbai+ndIeo8xFt2btxetUzuR6Srvh
xR2Sj9BbW1QToo2u8JfzD3X7AmRvlwKRX8708DPo4LDHZMA3qh0kfDDWkp2Eausf
LzE2cVxgq7fy/bDhUeN8xKQCSKJ7aguG6kOI6ROoZz21ljzDLUQzhm2qYF8SpU1o
tD8/uw53wLfSuhR3MBOB++xcn2imvSLqdgHWuhX6CtZIx5CD4y8inCbcLy/0Qrf6
sdTN5SAg2OZhjeNdzmqSWqhL2JTDw+Ou2fWzhEd0i/HN0y4NMr6h9fNn8nqxRyE7
IwIDAQAB
```

If the key is defined directly in the HDL file, you don't need to provide the `-k` option in the `encrypt_hdl` command.

Note

The key defined directly in an HDL source file has preference over the key defined in the key.txt file.

Step 4: Select the encryption algorithm for data encryption.

The Radiant software supports both a 128-bit and a 256-bit advanced encryption standard (AES) with CBC mode. Select the type of algorithm by defining one of the two options. The default is set to 256-bit AES with CBC mode.

```
`pragma protect data_method="aes128-cbc"
```

or

```
`pragma protect data_method="aes256-cbc"
```

Step 5: Run the encrypt_hdl Tcl command.

In the Tcl console window, type in the command to encrypt an HDL file. The option `-k` may or may not be used depending the location of the key file. The language selection (`-l`) and creation of new output file (`-o`) are optional. The default is Verilog. If you don't specify the output file, the tool generates a new output file named <*input_file_name*>_enc.v.

If the key was defined in the key.txt file: The command will encrypt the HDL file with all keys defined in the key.txt file.

```
encrypt_hdl -k <keyfile> -l <verilog | vhdl> -o <output_file>
<input_file>
```

If the key was defined directly in the HDL file:

```
encrypt_hdl -l <verilog | vhdl> -o <output_file> <input_file>
```

The encrypted file is located at the path specified in the `encrypt_hdl` command.

Step 6: Activate the encrypted HDL source file in the project file.

In the File List view, add the generated file to the project. Right-click on the encrypted file and choose **Include in Implementation**.

To see an example of a Verilog or VHDL pragma-annotated HDL source file, see "Defining Pragmas" on page 184.

Block-Based Design - Using Macro Blocks

A macro block is a portion of an FPGA design that can be preserved for reuse afterwards. Some of the benefits of implementing macros in a design include the ability to reuse blocks in other designs, and the ability to design more effectively as a team.

Macros contain logic or physical netlist (placed or routed) and is packaged as an IPM format for reuse. After a macro has been created, it is no longer parameterizable.

Macro Types:

There are three types of macro preservation levels:

1. **Logical (Logic Macro)**
   - Logic Macro is defined using Pre-Synthesis Constraint Editor.
   - Placement regions for Logic Macros are not required. However, if you want the macro to be placed at a specific area in both macro creation and reuse, a placement region is required.
   - Placement region is constrained inside Macro block but not fixed.
   - Routing region is constrained inside Macro block or Buffer Zone (if present) but not fixed.
   - Macro logic preserved.
   - Logic Macro can be exported after synthesis.
2. **Logical & Physical with Place Info (Firm Macro)**
   - Firm Macro is defined using Physical Designer and after running Map Design (MAP).
   - Placement regions are constraints for placement. They may not be good for achieving better performance as the macro has connections to I/O ports and other instances placed far away. However, you need to specify the region for the following cases:
     - Create Firm Macro as IP for other designs.
     - Create Firm Macro to be reused in other designs.
   - Routing regions for Firm Macros are not required.
   - Firm Macro can only be exported after Place and Route.
3. **Logic & Physical with Placement & Routing Info (Hard Macro)**
   - Hard Macro is defined using Physical Designer and after running Place & Route Design (PAR).
   - Placement region is required for Hard Macros.
   - You need to create routing regions for the following cases:
     - Create Hard Macro as IP for other designs.
     - Create Hard Macro to be reused in other designs.

  - Create Hard Macro for functional safety applications.
- Hard Macro can only be exported after Place and Route.

Macro Creation:

During macro creation, the macro is implemented as a design block and gets instantiated in the RTL design. The design can be used to contain additional logic to test macro functionality. Unified Design Database (.udb) supports the design block to be a macro.

- **Macro Module** – Module in the design database which implements macro netlist.
- **Macro Instance –** An instance in the design database where a macro module is instantiated.

The **ldc_create_macro** constraint defines the macro instance during the macro creation phase. It is expected that the instance name provided by the constraint can be found by the potentially hierarchical path specified in the constraint. If found, the module of the instance will be synthesized as macro during the synthesis run.

You can also define regions to guide placement and routing for a macro.

- **Placement Region** – Physical boundary of placed macro.
- **Routing Region** – Zone surrounding a macro region for additional resource for routing.

Routing region is always equal to or larger than placement region. It can be defined as extra buffer on left/right/top/bottom sides compared to placement region.

Export Macro:

Modules representing macros will be extracted from the design and exported as a .ipm file. The macro will be the top module in the .ipm file. The name specified in **ldc_create_macro** constraint will be used as the name of the top module.

Macro Reuse:

During macro reuse, you need to add the IPM files for each of the macros to be reused. These provide black box descriptions for the blocks which can be instantiated in the reuse design. A compatibility check is performed between the macro block and the current design target during import.

This section lists the options available in Block-Based Design.

Topics include:

- Creating a Macro Block
- Creating a Macro Region
- Exporting Macro

- Reusing a Macro Block
- Macro Usage Guidelines

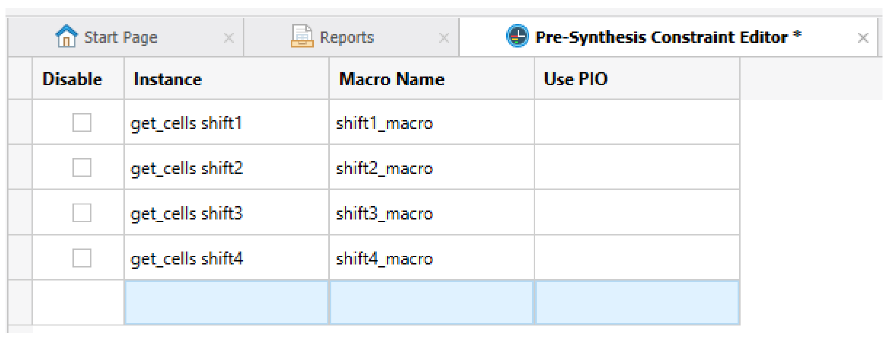
To get some hands-on experience, try the Lattice Radiant Block-Based Design Tutorial.

Creating a Macro Block

Macros must be created prior to synthesis because they are defined at logic instances. For macro creation, you can use a new or an existing project.

To create a macro block:

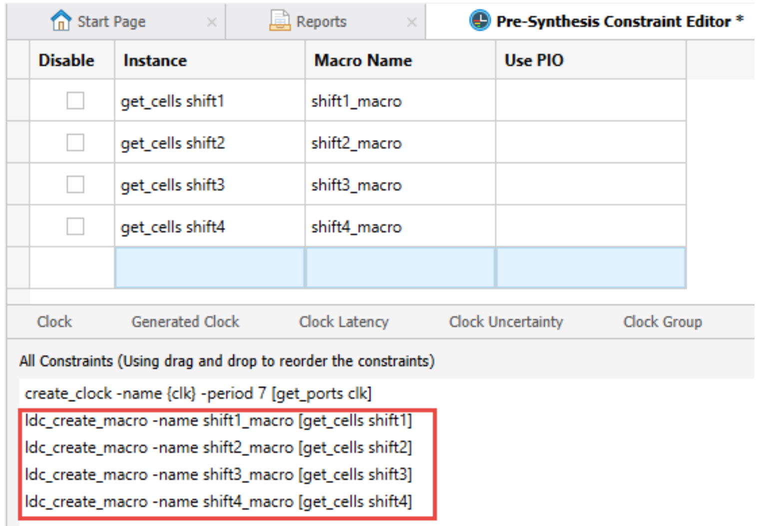
- Click **Tools** > **Pre-Synthesis Constraint Editor** > **Macro** tab.



The Macro tab contains the following columns:

- **Instance** – The Instance column specifies the hierarchical path of the instance which is defined as a macro. It specifies only one instance in a constraint. These constraints define the macro blocks.
- **Macro Name** – The name that is automatically set for the macro instance. The text boxes under the Macro Name column are editable.

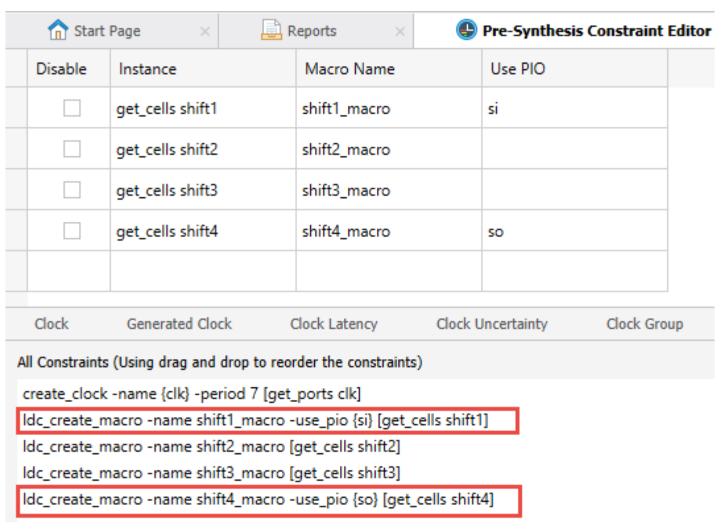
  When **Instance** and **Macro Name** have been added, the command line will be updated as follows:



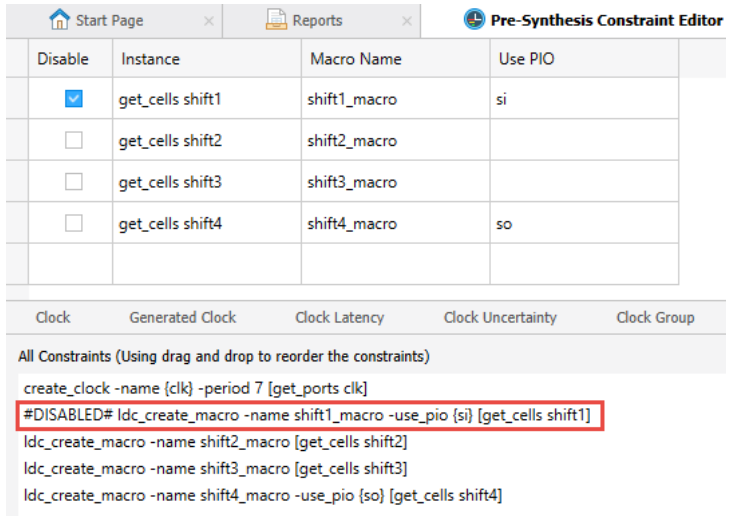
The **ldc_create_macro** constraint command tells the logic synthesis tool which instances are macros.

- **Use PIO** – The Use PIO column specifies the ports of the selected instance to define which ports need to use PIO so that it will be included inside the macro. If you do not manually specify ports, the PIO in the macro module will be automatically inferred during reuse.

  When you add ports to the Use PIO column, the command line will be updated as follows:



- **Disable** – If you do not want to synthesize macro blocks in a design containing macro constraints, the constraints can be disabled by selecting the checkbox in this column. You will see the #DISABLED# appended in front of ldc_create_macro constraint.



After saving the changes in **Pre-Synthesis Constraint Editor**, the defined macro is stored in pre-synthesis constraints (.sdc) file for logic synthesis.

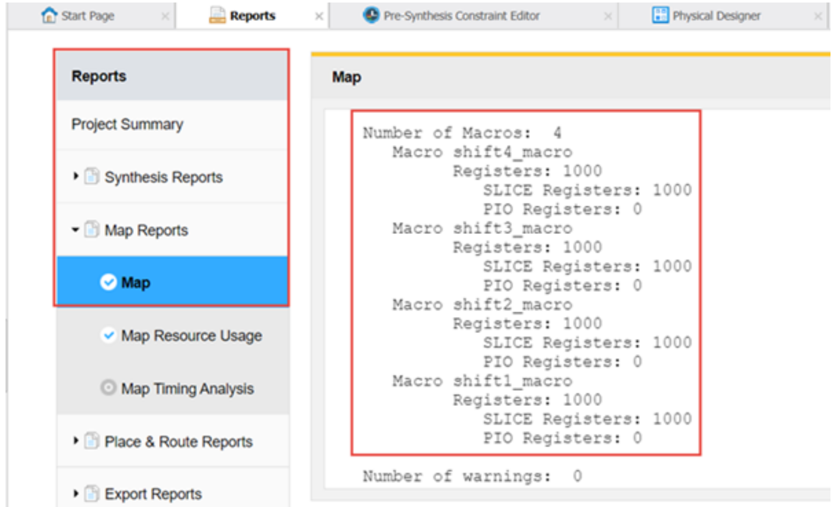
You can also implement macros to a design by adding the **ldc_create_macro** constraint to the active .sdc file.

Note:

By default, Radiant tools produce hierarchical paths with a "/" (forward slash) as separator. However, Synplify Pro uses a "." (period). In order for the flow to work, any hierarchical path needs to be adjusted for Synplify by either setting **set_hierarchy_separator {/}** to the constraint file or by manually changing any hierarchical path created in Radiant to use "." instead of "/."

To check Macro information in Map Report:

- After running MAP, you can view information about the created macros in the **Design Summary** section of the Map Report. The details in this report during the creation stage are based on the **ldc_create_macro** command in the .sdc file.
- This section displays the total number of resources used by the created macro (e.g., LUT, SLICE Register, PIO Register, and Block RAM).



Creating a Macro Region

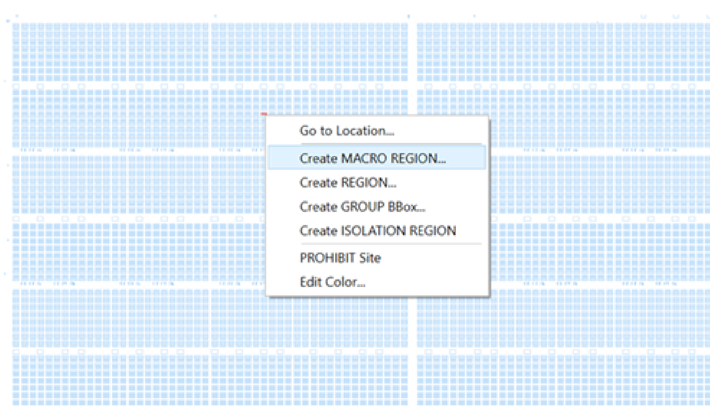
Once macro is created, you can create a macro region in **Physical Designer** (optional).

- If you already know the macro region, you can add the **ldc_create_region** constraint to the post-synthesis constraint (.pdc) file, or start **Physical Designer** to create a macro region before running **MAP**.
- If you want to use the placement result to guide the region creation, you can run **PAR** first. After running **PAR**, open **Physical Designer**, and highlight the macro instance from the netlist.

  Based on the macro placement result, you can create a macro region for the selected macro. The regions will be saved to the .pdc file. As a result, you need to rerun **MAP** and **PAR** to use the specified macro regions.

To create a macro region:

- In **Physical Designer**, right-click on the floorplan layout and select the **Create MACRO REGION…** option.



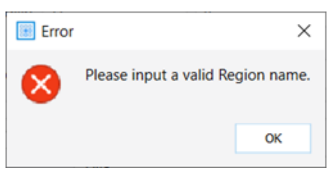
The **Create New MACRO REGION** dialog box opens.

The dialog box contains the following fields:

- **Macro Region Name** – The name that you set for the macro region.

  When specifying the macro region name, you can use any name you like.

  This option is mandatory. You may encounter the following error if you do not set the macro region name.



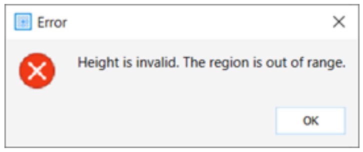
- **Anchor** – Anchor of the placement region used to place the macro.
- **BBox** – Width and height of the placement region.

  When you enter a value in the Anchor and BBox fields, the range of the following items might also change:

  - Macro Resource (LUTs, REGs, EBRs, DSPs)
  - Buffer Size range
  - BBox range

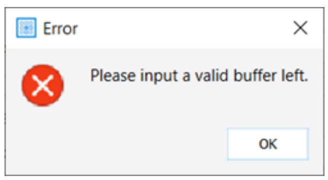
  The range changes depending on the value that you enter in the Anchor and BBox fields.

  If you do not enter a value within range in Anchor and BBox fields, you may encounter the following error.

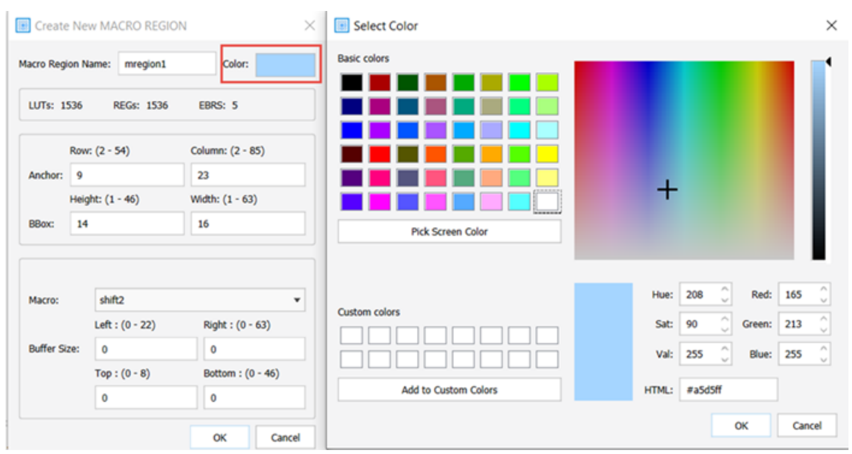


- **Macro dropdown menu** – You can select the predefined macro instance from the pre-synthesis macro creation.
- **Buffer size** – Buffer zone of the routing region. It gives PAR an extra room to do the routing. The routing is confined within the macro region and its surrounding ring.
  - **Buffer left** – Left buffer zone of the routing region.
  - **Buffer right** – Right buffer zone of the routing region.
  - **Buffer top** – Top buffer zone of the routing region.
  - **Buffer bottom** – Bottom buffer zone of the routing region.

  If you do not enter a value within range, you may encounter an error. The following example shows a buffer size error.



- **Color option** – Allows you to change the color of the macro region.



After creating a macro region, click the **Save** icon in **Physical Designer**. The created macro regions are saved into a post-synthesis constraint (.pdc) file.

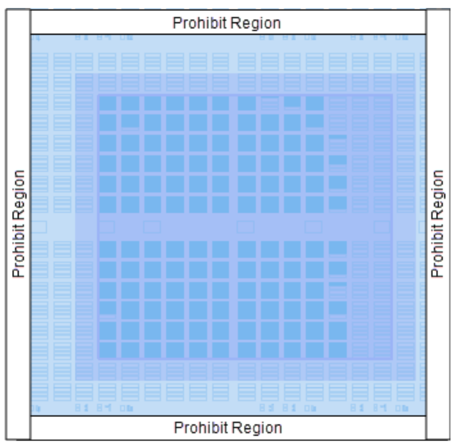
```
ldc_create_region -name mregion1 -site R4C2D -width 13 -height 13 -buffer_left 1 -buffer_right 1 -buffer_top 1 -buffer_bottom 1
ldc_set_location -region mregion1 [get_cells shift1]

ldc_create_region -name mregion2 -site R22C20D -width 13 -height 13 -buffer_left 1 -buffer_right 1 -buffer_top 1 -buffer_bottom 1
ldc_set_location -region mregion2 [get_cells shift2]

ldc_create_region -name mregion3 -site R39C44D -width 13 -height 13 -buffer_left 1 -buffer_right 1 -buffer_top 1 -buffer_bottom 1
ldc_set_location -region mregion3 [get_cells shift3]
```

The **ldc_create_region** constraint is designed to define placement region. It supports additional information of routing region. In the example above, -buffer_left/-buffer_right/-buffer_top/-buffer_bottom (optional) are used to define routing region. Without these options, routing region is identical to placement region.

Once macro regions are defined, you can assign a macro instance to the region by using **ldc_set_location** constraint.

To create an isolated region:

- In **Physical Designer**, right-click on the floorplan layout and select the **Create ISOLATION REGION** option.
  - Draw a rectangle around the sites where you want to place the isolated region.



  - Physical Designer automatically creates prohibited regions on each side. Once prohibited regions are created, the content of the macro can only be accessed by approved signals from the main design.
  - The inner macro locks placement and routing. It also excludes any other logic from using the region for placement or routing.
  - The outer macro is set as empty macro, no logic or routing can be used inside of it.
- If a gap exists between isolation and secure region, some design instances may be placed within it by PAR. The gap will not be considered as an empty/secure region. You need to manually cover the prohibited region by expanding it.
- You can expand the isolated regions by doing the following:
  - Double-clicking the region that you want to expand and changing the BBox field values in the Edit Region Property dialog box.

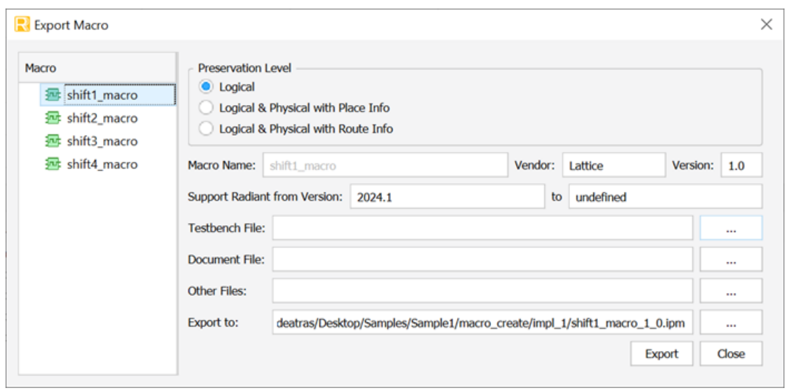
- Selecting the region that you want to expand and dragging it to the desired size.
- Editing the height or width of the isolated region in the ldc_create_region constraint within the .pdc file.

Exporting Macro

Once synthesis is passed, you can launch the Export Macro dialog box from the Tools menu. The dialog box lists all defined macros on the left-hand pane. You can select them and set their preservation levels. When macro is exported, a .ipm file will be generated.

To export macro:

- Click **Tools > Export Macro**.
- In the **Export Macro** dialog box, you can choose the macro preservation levels and specify the default path where the macro package will be exported.



The macro data can be exported according to a specified preservation level. During macro reuse, you can set the preservation levels to a lower value.

Logical (Logic Macro)

- After running **Synthesize Design** and **MAP**, you can only export to **Logical** preservation level.

Logical & Physical with Place Info (Firm Macro)

- For Firm Macro, you can only export to **Logical** and **Logical & Physical with Place Info** preservation levels after running **Place & Route Design**.

Logic & Physical with Placement & Routing Info (Hard Macro)

- After running **PAR**, you can export to **Logical**, **Logical & Physical with Place Info**, and **Logic & Physical with Placement & Routing Info** preservation levels.

The Export Macro dialog box also contains the following fields:

- **Macro Name** – Displays the name of the selected macro from the Macro column in the left-hand pane.
- **Vendor** – Displays the vendor name.
- **Version** – Displays the Macro IP version, the default is always 1.0.
- **Support Radiant from Version to Version** – Displays the supported maximum and minimum Radiant version. The maximum version could be empty, but the minimum is the same as the current Radiant version.
- **Testbench File** – Testbenches that allow you to do simulation or evaluate the macro (optional).
- **Document File** – Contains any document such as help, user guide, or introduction file included in the macro (optional).
- **Other Files** – Any files such as readme files and other documentation (optional).
- **Export to** – Default path where the macro package will be saved.

After exporting, a macro .ipm package will be generated. It contains the following files:

- **<macro_name>.mac** – Contains general information of the macro project.

  The following example shows the information included in the .mac file.

```
"Device": "LIFCL-40",
"Device Package": "CABGA400",
"Family": "LIFCL",
"Files": {
    "Constraints": "",
    "Document": "",
    "LogicDB": "shiftreg1_logic.mdb",
    "OtherFiles": "",
    "PlaceDB": "D:\/ut\/shiftregDemo\/jedi_create\/demo\/demo_create\/impl_1\/demo_create_impl_1.udb",
    "RouteDB": "D:\/ut\/shiftregDemo\/jedi_create\/demo\/demo_create\/impl_1\/demo_create_impl_1.udb",
    "Testbench": "",
    "Interface": "shiftreg1_bb.v",
    "SimulationModel": "shiftreg1_sim.vo"
},
"Name": "shiftreg1",
"Preservation": "logic",
"SupportRadiantMax": "undefined",
"SupportRadiantMin": "1.0",
"Vendor": "Lattice",
"Version": "1.0",
"ResourceUsage": {
    "lut": "0",
    "ff": "1000"
}
```

- **<macro_name>_bb.v** – Synthesis header file of the macro project. It is exported to define module interface for synthesis.
- **<macro_name>_bb_extra.v** – Synthesis header file exported with additional information.

  The bb.v and bb_extra.v files may contain the following information:

  - **TYPE** – Specifies the type of the module. Currently, only the "MACRO" type is supported.
  - **syn_allowed_resources** – Allowed resource in the macro region. It is needed to help create a reasonable-sized macro region.
  - **Locked** – The macro with locked attribute is fixed, not movable.

  These information are to be propagated to top design by synthesis tools when the macro is reused as a black box. Tools can get the information before the macro is integrated to the top design.

The following example shows the information included in the bb.v file.

```
module example_macro (inp, outp1, outp2)
/* synthesis TYPE = "MACRO"

            syn_black_box
            black_box_pad_pin = "outp1,outp2"

            syn_resources = "luts=500,regs=400"
*/;
```

The following example shows the information included in the bb_extra.v file.

```
module example_macro (inp, outp1, outp2)
/* synthesis TYPE = "MACRO"

            syn_black_box
            black_box_pad_pin = "outp1,outp2"

            syn_resources = "luts=500,regs=400"
            syn_allowed_resources = "luts=550,regs=450"

        ARCHITECTURE  = "je5d00"
            ANCHOR        = "R9C25"
            BBOX          = "3,7"
            BUFFER_LEFT   = "1"
            BUFFER_RIGHT  = "1"
            BUFFER_TOP    = "1"
            BUFFER_BOTTOM = "2"

            LOCKED
*/;
```

- **<macro_name>_sim.vo** – Exported for macro simulation. The file is generated based on logic macro.
- **<macro_name>.mdb** – Exported Macro in unified constraints database (.udb) format. Modules representing macros will be extracted from design and exported to .ipm files. The macro will be the top module in .ipm file. The name specified in ldc_create_macro constraint will be used as the name of the top module.

Reusing a Macro Block

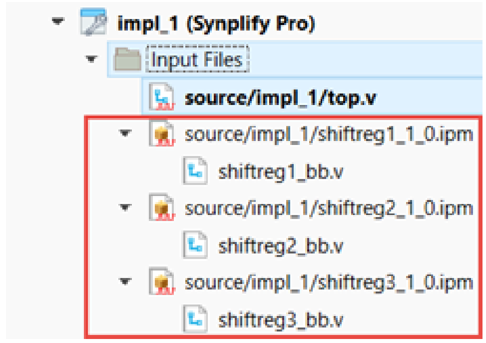
During macro reuse, you need to add the IPM files for each of the macros to be reused. These provide black box descriptions for the blocks which can be instantiated in the reuse design. A compatibility check is performed between the macro block and the current design target during import.

Macros can be reused with different devices of the same family:

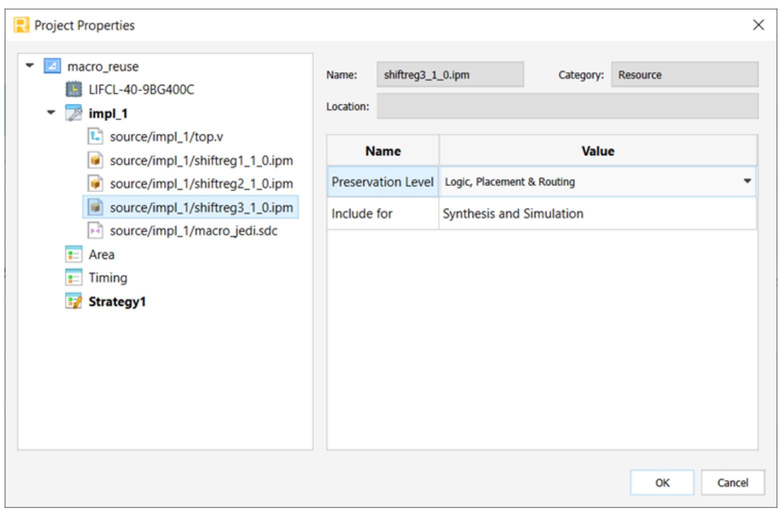
- In general, Logic Macros can be reused in different devices from the same family depending on the preservation level and resources used.
- Firm and Hard Macros can only be reused with the same device.

To reuse a macro within another project:

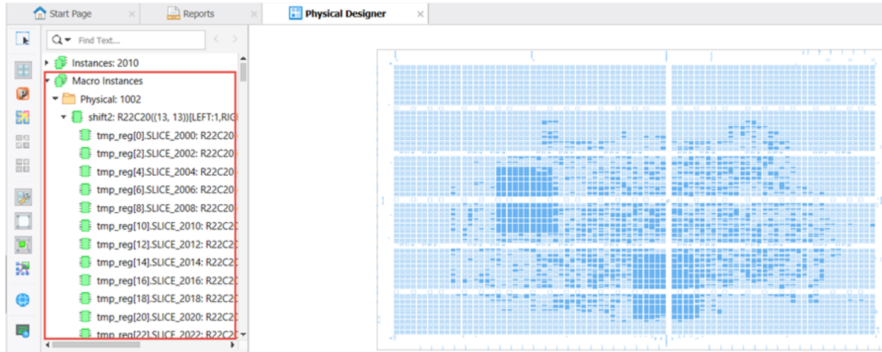
- For macro reuse, the .ipm file for each respective macro that is exported can be used by importing the existing .ipm files into your project. The imported files will be stored under the **Input Files** folder in the Radiant software.



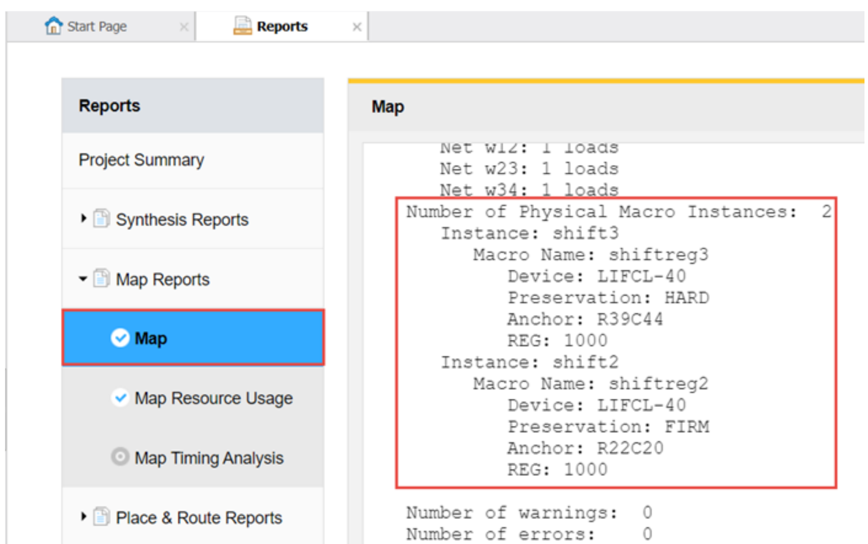
- After running **Synthesize Design**, the post-synthesis Logic Macro is merged into your design.
- You can lower the preservation level of the selected .ipm file in **Project Properties** (optional). Preservation levels must be equal or less than the preservation level when macro is exported.
  - If the imported macro's preservation level is Firm Macro, you can lower it to Logic Macro.
  - If the imported macro's preservation level is Hard Macro, you can lower it to Logic and Firm Macro.



- When you open **Physical Designer**, the netlist shows the reused macro instances. They contain information of the original region from the create stage.

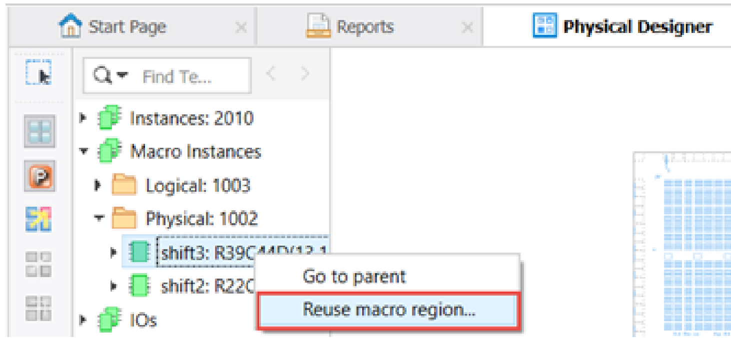


- Run **MAP** with reused macro.
  - Firm Macro – The post-MAP macro design with placement constraint is merged into your design.
  - Hard Macro – The post-MAP macro design with placement and route constraint is merged into your design.
  - After running MAP, you can view information about the reused macros in the **Design Summary** section of the Map Report.
    - This section lists Firm and Hard macro information and attributes such as device family, preservation level, anchor, and main resources.
    - The syn_resources information generated when exporting macro (e.g., LUT, REG, Block RAM, LRAM) determines the main resources listed in the Map Report.

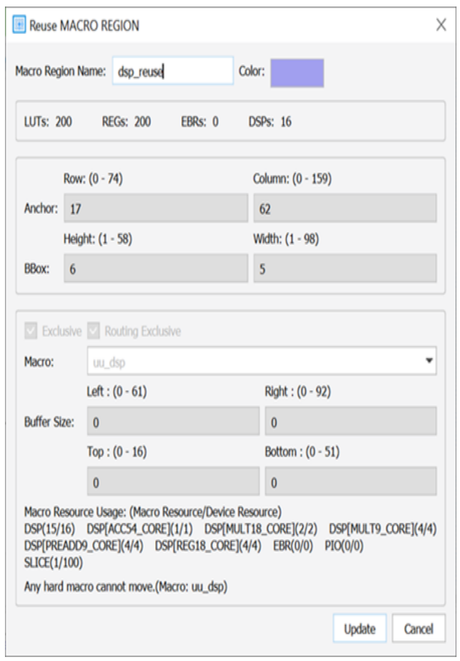


- You can also create a macro region with the **Exclusive** option in Physical Designer. This option excludes other logic from the region for placement and routing when reusing the exported Firm and Hard macro.

  The **Reuse Macro Region** dialog box opens when you right-click on a macro instance from the netlist and select the **Reuse macro region...** option.

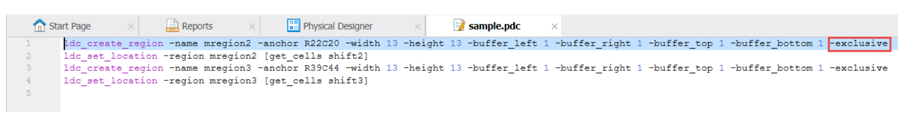


For these macro instances, you can only enable the Exclusive option. The text boxes are grayed out.



- Once done, click the **Save** icon in Physical Designer.

  The created macro regions are saved into a post-synthesis constraint (.pdc) file. The **-exclusive** option will be added to the **ldc_create_region** command line if Exclusive was enabled from the Reuse Macro Region dialog box.

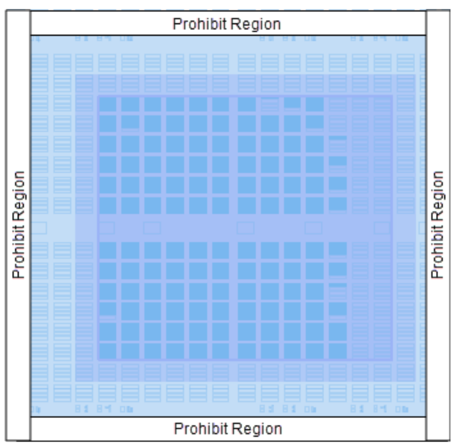


- In **Physical Designer**, you can also right-click on the floorplan layout and select the **Create ISOLATION REGION** option.

  - Draw a rectangle around the sites where you want to place the isolated region.

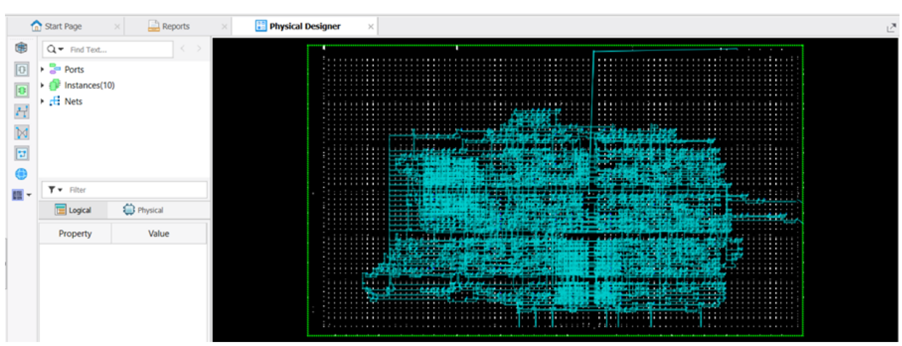
- Physical Designer automatically creates prohibited regions on each side. Once prohibited regions are created, the content of the macro can only be accessed by approved signals from the main design.
- The inner macro locks placement and routing. It also excludes any other logic from using the region for placement or routing.
- The outer macro is set as empty macro, no logic or routing can be used inside of it.

- If a gap exists between isolation and secure region, some design instances may be placed within it by PAR. The gap will not be considered as an empty/secure region. You need to manually cover the prohibited region by expanding it.
- You can expand the isolated regions by doing the following:
  - Double-clicking the region that you want to expand and changing the BBox field values in the Edit Region Property dialog box.
  - Selecting the region that you want to expand and dragging it to the desired size.
  - Editing the height or width of the isolated region in the ldc_create_region constraint within the .pdc file.



- Run **PAR** with reused macro**.**

  After running **PAR**, you can open **Physical Designer** and select **Routing Mode** to see the macro instances in the physical netlist.



Macro Usage Guidelines

Macro creation and reuse have the following guidelines:

- Macro is created from logic instance.
- For multiple macro instances, you need to create a wrapper on top of them.
- Macro creation is instance based. If the whole design is a macro, you need to instantiate it in a top design with a wrapper.
- For macro reuse, macro black box timing model is not supported. Synthesis cannot optimize macro's interface data paths. PAR will need to see full netlist to run timing analysis.
- Macro cannot contain JTAG and PCS hard-IP, which may require merging with others during reuse.
- Place/Route region only supports a single rectangular region.
- A module to be created as a macro block cannot contain black boxes.

Implementation Flow and Tasks

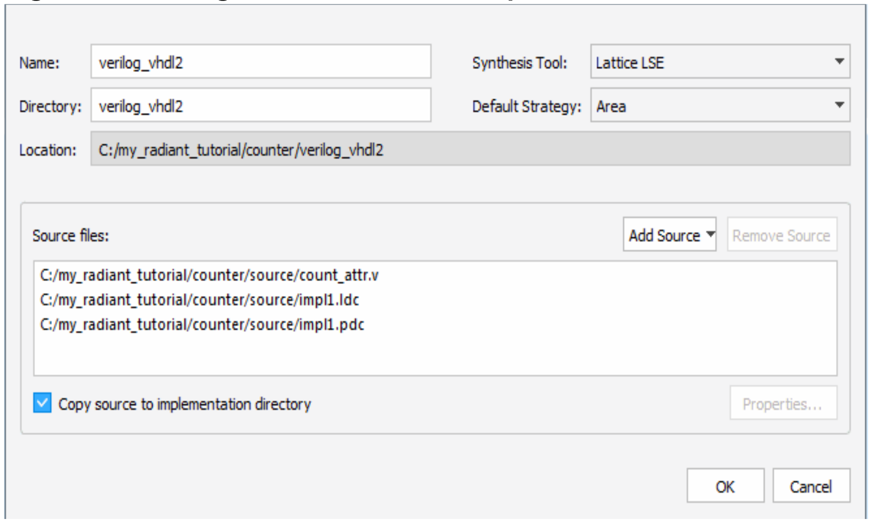
Implementations organize the structure of your design and allow you to try alternate structures and tool settings to determine which ones will give you the best results.

To help determine which scenario best meets your project goals, use a different implementation of a design using the same tool strategy, or run the same implementation with different strategies. Each implementation has an associated active strategy, and when you create a new implementation you must select its active strategy.

To try a new implementation with different strategies, you must create a new implementation/strategy combination.

1. Right-click the project name in File List.
2. Choose **Add > New Implementation**.
3. In the dialog box, choose a source file from an existing implementation using the Add Source drop-down menu.
4. Choose a strategy using the Default Strategy drop-down menu.

Figure 58: Adding a Source to a New Implementation



To use the same source for new and existing implementations, make sure that the "Copy source to implementation directory" option is not selected. This will ensure that your source is kept in sync between the two implementations.

Synthesis Constraint Creation

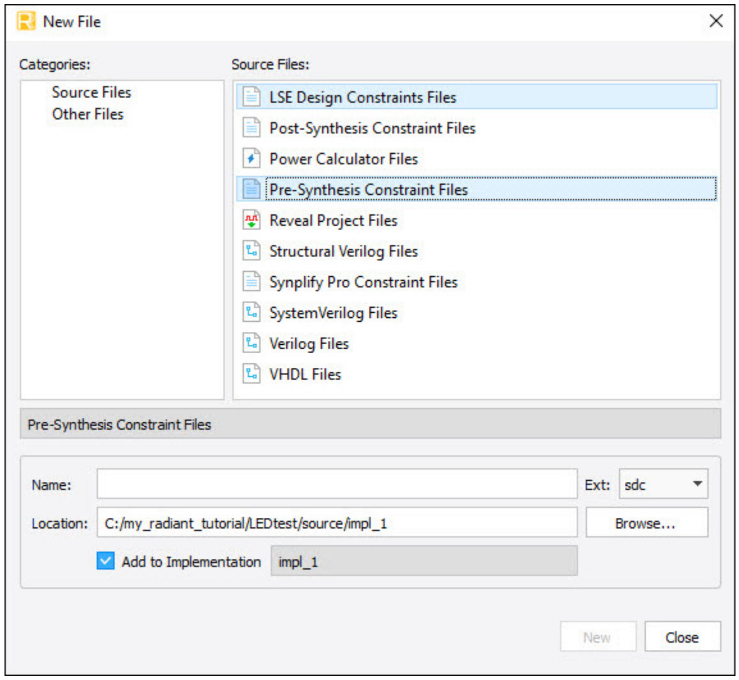
Synthesis constraints can be added to a design implementation in the format of the Synopsys Design Constraint language, while constraints can be added in the Synopsys standard timing constraints format in the form of FPGA Design Constraint (LDC, PDC, or FDC format) files.

If you are using the Lattice Synthesis Engine, the synthesis constraints will be included in an .ldc file. If you are using Synplify Pro for synthesis, the constraints will be included in an .fdc file. The older .sdc file format is also supported for constraints.

To create a new synthesis constraint file, right-click the Synthesis Constraint Files folder in the File List pane and choose **Add > New File**. In the New File dialog box, select one of the following and give the file a name:

- Pre-Synthesis Constraint Files (.ldc)
- Post-Synthesis Constraint Files (.pdc)
- Synplify Pro Constraint Files (.fdc)

Figure 59: New Synthesis Constraint Files



The .ldc, .pdc, or .fdc file will open in the Source Editor to allow you to manually add the constraints. You can use the Pre-Synthesis Constraint Editor tool to add pre-synthesis constraints to the .ldc file and the Post-Synthesis Timing Constraint Editor tool to add logic view level post-synthesis timing constraints to .pdc files. You can also use the Device Constraint Editor and Floorplan View to add physical constraints to .pdc files. For detailed information about setting constraints, see **User Guides > Applying Design Constraints** and the **Reference Guides > Constraints Reference Guide** in the Radiant Software Help.

An alternative way of adding constraint files is through Source Template. To view a constraint template, click on the Source Template tab on the left-hand side of the Project Navigator pane. If not selected, make sure it is enabled in **View > Show Views > Source Template**. The list of constraint templates includes the timing constraints, physical constraints, and user templates. Select a template and copy and paste it into your active design.

Constraint Creation

LDC (pre-synthesis) and PDC (post-synthesis) files are used to input timing and physical constraints. The following steps illustrate how to assign and edit constraints in the Radiant software and implement them at each stage of the design flow.

1. If desired, define some constraints at the HDL level using HDL attributes. These source file attributes are included in the Unified Database (UDB), and will be displayed in the Radiant software after the Map Design process is run. The following is an example of applying the LOC attribute in Verilog source code:

```
module top (
  input clk1,
  input datain /* synthesis loc = B12 */,
  output ff_clk1out
)
```

For more information on HDL Attributes, refer to the Radiant Software Help. See **Reference Guides > Constraints Reference Guide > HDL Attributes**.

2. Open one or more of the following tools to create new constraints or to modify existing constraints from the source files:
   - Device Constraint Editor, which consists of:
     - Spreadsheet View – the primary view for setting constraints.
     - Package View – examines the pin layout of the design, modifies signal assignments, reserves pin sites that should be excluded from placement and routing, and runs PIO design rule check to verify legal placement of signals to pins.
     - Device View – examines FPGA device resources.
     - Netlist View - shows port types (Input, Output) and groups.
   - Timing Constraint Editor. Timing/Physical constraints are entered through:
     - Pre-Synthesis Constraint Editor - used to enter pre-synthesis constraints such as clocks, clock latency/uncertainty/Group, Input/Output delays, timing exceptions, and attributes.
     - Post-Synthesis Timing Constraint Editor - This post-synthesis version of the Timing Constraint Editor is used to enter logic view level timing constraints and physical constraints.
   - Floorplan View examines the device layout of the design, draws bounding boxes for GROUPs, draws REGIONs for the assignment of groups or to reserve an area, and reserves sites and REGIONs that should be excluded from placement.
3. Save the constraints to the pre-synthesis constraint file (.ldc) or post-synthesis constraint file (.pdc).
4. Run the Map Design process (Map).
5. Run the Map Timing Analysis process and examine the timing analysis report. This is an optional step, but it can be a quick and useful way to identify serious timing issues in the design and constraint errors (syntax and semantic). Modify constraints as needed and save them.
6. Run the Place & Route Design process.
7. Open views directly or by cross-probing to examine timing and placement and create new GROUPs. Also examine the Place & Route Timing report.
8. Modify constraints or create new ones using the appropriate constraint tool. Save the constraint changes and rerun the Place & Route Design process.

Simulation Flow

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 and Synthesis (default)
- Simulation
- Synthesis

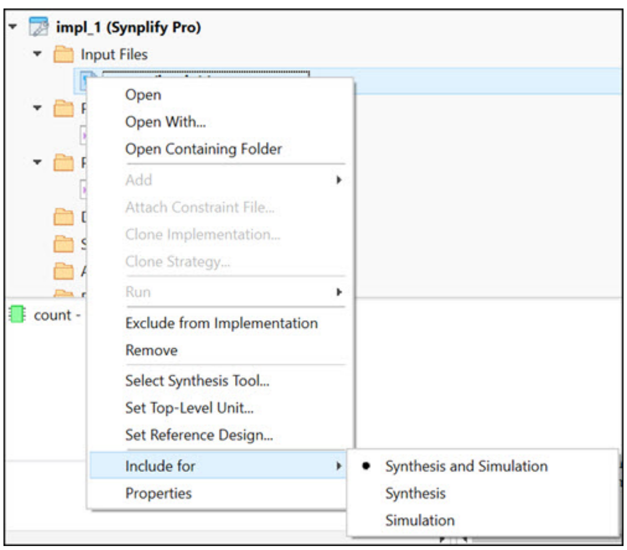
This allows the use of test benches, including multiple file test benches. Additionally, multiple representations of the same module can be supported, such as one for simulation only and one for synthesis only.

You can add top level signals to the waveform display in the simulator and to automatically start the simulator running.

The Simulation Wizard automatically includes any files that have been set for simulation only or for both simulation and synthesis. You can select the top of the design for simulation independent of the implementation design top. This allows easy support for test bench files, which are normally at the top of the design for simulation but not included for implementation. The implementation wizard exports the design top to the simulator, along with source files, and set the correct top for the .spf file if running timing simulation.

After you add a module, use the **Include for** menu to specify how the module file is to be used in the design.

Figure 60: "Include For" Input Files



Simulation Wizard Flow

By generating simulation scripts, Simulation Wizard simplifies the process of simulating projects. It automates the process of setting up a project for simulation.

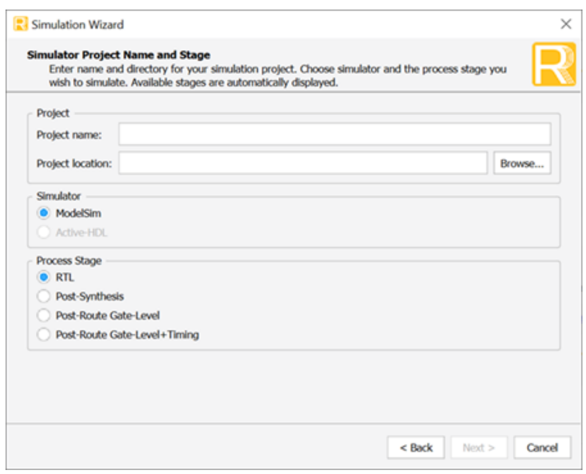
This topic describes how to use the Radiant Simulation Wizard to create a simulation project (.spf) file so you can import it into a standalone simulator.

1. In the Radiant software, click **Tools > Simulation Wizard** or click the button in the toolbar. The Simulation Wizard opens.
2. In the **Preparing the Simulator Interface** page, click **Next**.
3. In the **Simulator Project Name and Stage** page, enter the name of your project in the **Project Name** text box.

   In the **Project Location** box, 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 dialog box that appears.
4. In the Simulation Wizard dialog box, select either the **ModelSim** or **Active-HDL** simulator.



Note:

- If Aldec-HDL is not installed, it will not be selectable.
- If you change simulation tool, project name will not change. Any previous Simulation Wizard results will be deleted, and a new simulation will be started using new simulation files.

5. In the **Process Stage** box, choose the stage where you want to start creating the simulation project.

The following options are valid types:

- **RTL** – The RTL option allows you to run simulation before synthesis. This option is always selectable.
- **Post-Synthesis**
  - To enable this option, you must first generate a post-synthesis simulation file (<file_name>_syn.vo) as follows:

    In the Process Toolbar, click Task Detail View and select **Post-Synthesis Simulation File**.

    Run **Synthesize Design**.
- **Post-Route Gate-Level** and **Post-Route Gate-Level+Timing**
  - To enable these options, you must first generate a post-routed, back-annotated simulation file (<file_name>_vo.vo) and a timing constraint file (<file_name>_vo.sdf) as follows:

    In the Process Toolbar, ensure that **Gate-Level Simulation File** is selected in Task Detail View and click **Export Files**.

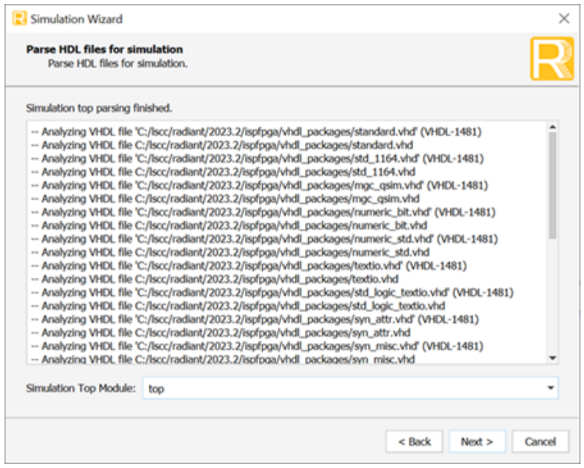
6. Click **Next.**
7. In the **Add and Reorder Source** page, the files included in the simulation are listed in the main **Source Files** window.

   - Files will be compiled in the order they appear here, with the top file being the first to be compiled.
   - You can also manually add and remove files using the buttons at the top of the window.

   By default, the **Automatically set simulation compilation file order** option is checked.

   - If this option is enabled, Radiant automatically detects the order of the simulation files. Otherwise, you have to manually specify the order of the files for your simulation using the buttons.
8. Click **Next**.

   The **Parse HDL files for simulation** page appears.



Radiant automatically infers the **Simulation Top Module**.

If you want to select a different testbench for your project, use the **Simulation Top Module** dropdown menu.

Click **Next** after choosing the correct testbench module.

9. A **Summary** page appears that provides information on the project selections, including the simulation libraries.

   Review all of the information on this page to ensure everything is correct.

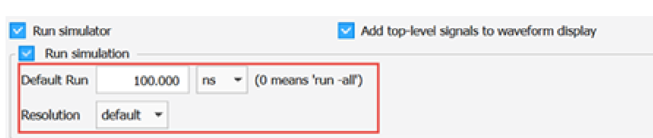
10. By default, the **Run simulator**, **Add top-level signals to waveform display**, and **Run simulation** options are enabled in the Summary page. Their functionality is as follows:

    - The **Run simulator** option runs the script and launches the simulation tool you chose earlier in the Simulation Wizard setup page.
    - The **Add top-level signals to waveform display** option adds top-level signals to the waveform in the simulation tool.

      This option can only be selected if the **Run simulator** option is enabled.
    - The **Run simulation** option initializes the generated Simulation Wizard script by running the Tcl command. By enabling this option, your simulation tool runs a simulation automatically.
    - This option can only be selected if the **Run simulator** and **Add top-level signals to waveform display** options are enabled.

11. If the **Run simulation** option is selected, you can modify the following settings.

    - **Default Run** – The default run length is set to 100, measured in nanoseconds (ns).



- You can manually enter an integer or fractional value in the box. A value of 0 has the same function as the **run -all** argument. This value enables the current simulation to run continuously or until it reaches a breakpoint.
- In the dropdown menu, you can choose a different time unit. Available time units are fs, ps, ns, us, ms, or sec.

- **Resolution** – The Resolution setting supports the **-t** option in the Simulation Wizard. This option specifies the simulator time resolution.

  If you select a different value from the dropdown menu instead of **default**, the Resolution setting adds **-t <value>** to the vsim command line of the qrun argument file.

12. Click **Finish**.

- After completing the Simulation Wizard setup:
  - A **Simulation Wizard Project (.spf) file** and a **qrun argument (.f) file** are generated.

    The qrun argument file contains generated commands and arguments. For example:

    To compile all of the Verilog modules in your design, the Simulation Wizard adds the path to your Verilog files in the argument file, and generates a run command to run the simulation in the duration specified in step 11.

    The generation of this file is triggered by Simulation Wizard or the **sim_generate_script** command.
  - The simulation is launched and displayed in the ModelSim GUI.

Chapter 7

Working with Tools and Views

This chapter covers the tools and views controlled from the Radiant software. Tool descriptions are included and common tasks are described.

Overview

The Radiant design environment streamlines the implementation process for FPGAs by combining the tool and data views into one common location. Two main features of this design environment make it easy to keep track of unsaved changes in your design and examine data objects in different views.

Shared Memory The Radiant software uses shared memory that is accessed by all tools and views. As soon as design data has been changed, an asterisk * appears in the tab of the open views, notifying you that unsaved changes are in memory.

Cross-Probing Shared design data in the Radiant software enables you to select a data object in one view and display it in another. This cross-probing capability is especially useful for displaying the physical location of a component or net after it has been implemented. You can click on a hyperlink icon to cross-probe into the specific tool. The Radiant software supports:

- Post-Synthesis timing report links to Netlist Analyzer
- Map & PAR timing reports link to Physical Designer’s Placement and Routing views

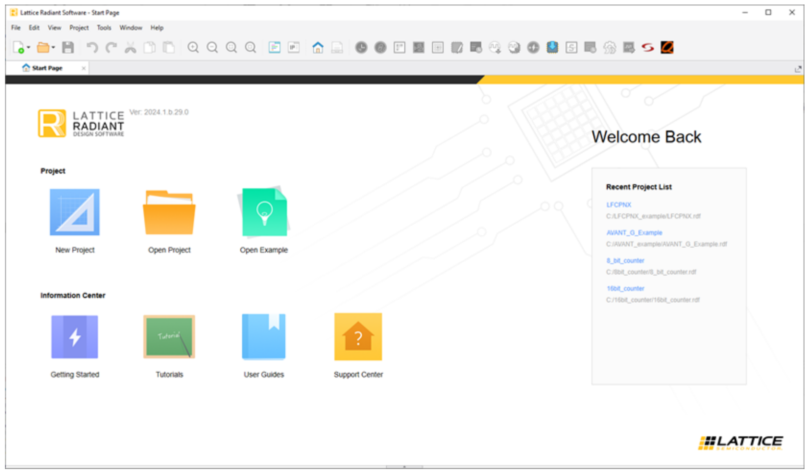
View Menu Highlights

The View menu and toolbar control the display of all toolbars, project views, and display control. Also included in the View menu are the important project-level features: Start Page and Reports.

Start Page

The Start Page is displayed by default when you run the Radiant software. The Start Page enables you to open projects, read product documentation, and view the software version and updates. You can modify startup behavior by choosing **Tools > Options > General > Startup**.

Figure 61: Default Start Page

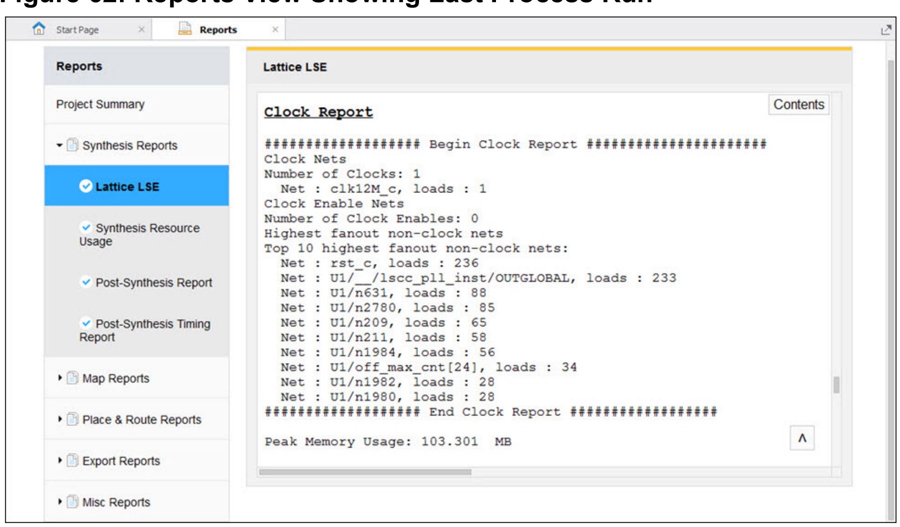


The Start Page gives you quick access to recent projects and to product documentation.

Reports

The Reports view provides one central location for the project summary and design processing reports. It is displayed by default when a project is open. Alternatively, click on the Reports icon in the toolbar.

Figure 62: Reports View Showing Last Process Run



The Reports view is organized into Project Summary, Synthesis Reports, Map Reports, Place & Route Reports, Export Reports, and Misc Reports.

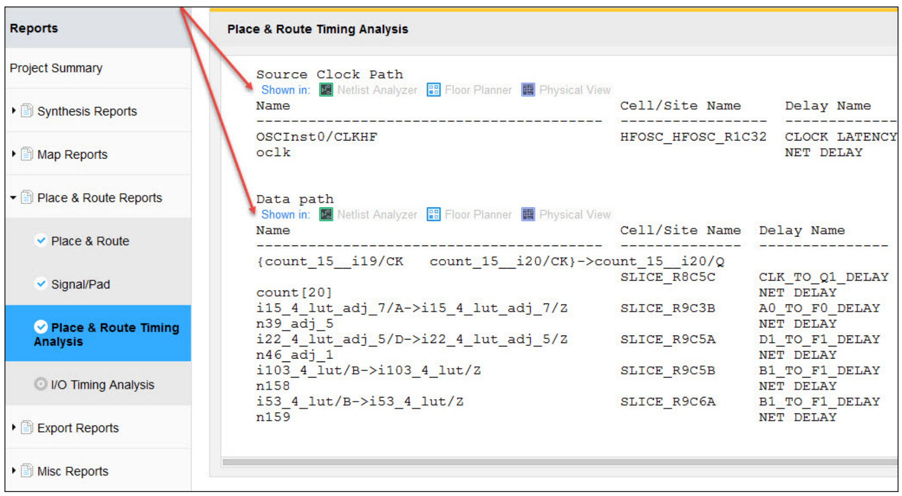
The different file icons indicate:

- A report has been completed (blue check mark).
- A report has never been generated (gray circle).
- A report is out of date (orange question mark).

Select any item to view its report.

The Reports view also supports path cross-probing through the timing or analysis reports. You can view a specific clock or data path in Netlist Analyzer and Physical Designer by clicking the icon next to the path.

Figure 63: Cross-Probing



To learn more about cross-probing, view “Cross-Probing” on page 42.

Tools

The entire FPGA implementation process tool set is contained in the Radiant software. You can run a tool by selecting it from the Tools menu or toolbar.

This section provides an overview of each of these tools. More detailed information is available in their respective user guides, which you can access from the Start Page or from the Radiant Software Help. Detailed descriptions of external tools can be found in their product documentation as well.

If you are viewing an encrypted design, some secured objects may not be visible in the selected tool. To learn more, see **User Guides > Securing the Design > Secure Objects in the Design** in the Help.

Timing Constraint Editor

The Timing Constraint Editor is used to edit pre-synthesis (.ldc) constraints and post-synthesis constraints stored in a .pdc file.The Timing Constraint Editor consists of two tools:

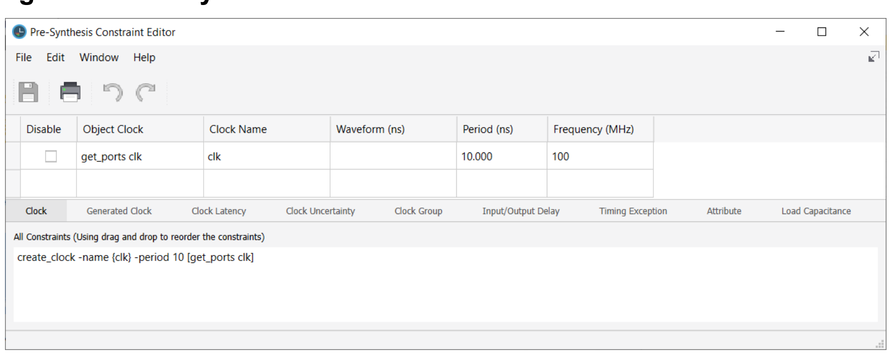
- Pre-Synthesis Constraint Editor
- Post-Synthesis Timing Constraint Editor

Both tools have identical interfaces and the entry mechanisms of the constraints are also the same. The key differences are that pre-synthesis constraints are entered pre-synthesis and are synthesized by the chosen synthesis tool. The post-synthesis constraints are already synthesized and populated in each of the different constraints tabs. You cannot modify the post-synthesis constraints that were populated from pre-synthesis, but can supply new constraints (physical and timing) to either override the existing one already populated or supply new constraints to better constrain the design for improved performance upon analysis.

The different constraint types are entered through these tabs:

- Clock - used to define the clocking scheme of the design.
- Generated Clock - used to define generated clocks such as from PLLs.
- Clock Latency - is the delay between the clock source and clock pin. Used to define the latency in terms of rise and fall times.
- Clock Uncertainty - is the jitter difference of two signals, possibly caused by clocks. Used to define the amount of uncertainty of a clock or during clock domain transfer.
- Clock Group - used to specify which clocks are not related in terms of being logically/physically exclusive and asynchronous.
- Load Capacitance - used to define the load capacitance of ports.
- Input/Output Delay - used for setting input and output delays.
- Timing Exception - used to set min/max, false, and multicycle path constraints.
- Attribute - used to set synthesis attributes.

Figure 64: Pre-Synthesis Constraint Editor



Unified Constraints Flow

The goal of the unified constraints flow is to have one constraint file that can be used by both the synthesis tools (LSE & Synplify Pro). From Radiant 3.0 onwards, pre-synthesis TCE can be launched with Synplify Pro, and constraints can be saved into an .sdc file, which can be used by both LSE and Synplify Pro, along with .ldc (for LSE) and .fdc (Synplify Pro).

.

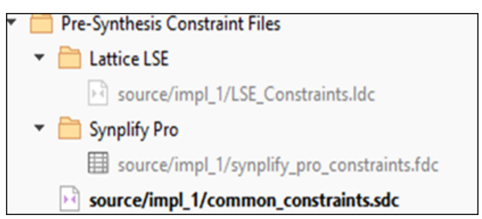
Table 1: Supported Constraints File Type

Constraint Type	Synthesis Tool	TCE Support
.ldc	LSE	Yes
.sdc	LSE/Synplify Pro	Yes
.fdc	Synplify Pro	No

Table 2: Difference between Radiant 3.0 (or later) and previous versions

	Radiant 1.x/2.x	Radiant 3.0 or later
.sdc file	For Synplify Pro only	For both LSE and Synplify Pro

The following figure shows the Radiant folder structure. The .ldc constraints are stored under the Lattice LSE folder, the .fdc constraints are stored under Synplify Pro, and the newly created .sdc constraints are stored outside these two folders, which can be commonly used by both synthesis tools.



Notes:

1. If the previous project includes an .sdc file, Radiant 3.0 or later will use it in the LSE flow, but it can cause unexpected syntax errors.
2. If the previous project includes both .ldc and .sdc files, Radiant 3.0 or later will force one of them to be inactive. Only one constraint file can be active. The Timing Constraint Editor can only be launched with the active constraints file.

Known Limitations

- Constraints on internal objects from Synplify Pro may not be fully compatible. The pre-synthesis TCE uses Verific parser for compilation whereas Synplify Pro uses their own parser. This may cause some name changes on the internal objects when Synplify Pro is used.
- Pre-Synthesis TCE "Attribute" tab is only supported by LSE. The attributes are passed using a Lattice Proprietary constraint "ldc_define_attribute," which is not supported by Synplify Pro.

See Also

- **Reference Guides > Constraints Reference Guide** in the Radiant Help.
- **User Guides > Applying Design Constraints > Using Radiant Software Pre-Synthesis Constraints** in the Radiant Help.
- **User Guides > Applying Design Constraints > Using Radiant Software Tools > Device Constraint Editor** in the Radiant Help

Constraint Propagation

Constraint propagation is not a stand-alone tool, nor do you have to provide any input to take advantage of this feature.

You usually define constraints for the top-level design as well as include other constraint files for any IP that are generated with Radiant tools such as IP Catalog. Constraints defined at the IP level may not contain the correct hierarchical names and so will not be applied correctly when synthesized. To help honor as much of the supplied constraints as possible, this feature runs a design rule check (DRC) on all the input constraints and writes out a new constraint file to support hierarchical constraints such as soft-IP constraints and honor top-level constraints.

For more information about Constraint Propagation, refer to the Radiant Software Help. See **User Guides > Applying Design Constraints > Constraint Propagation**.

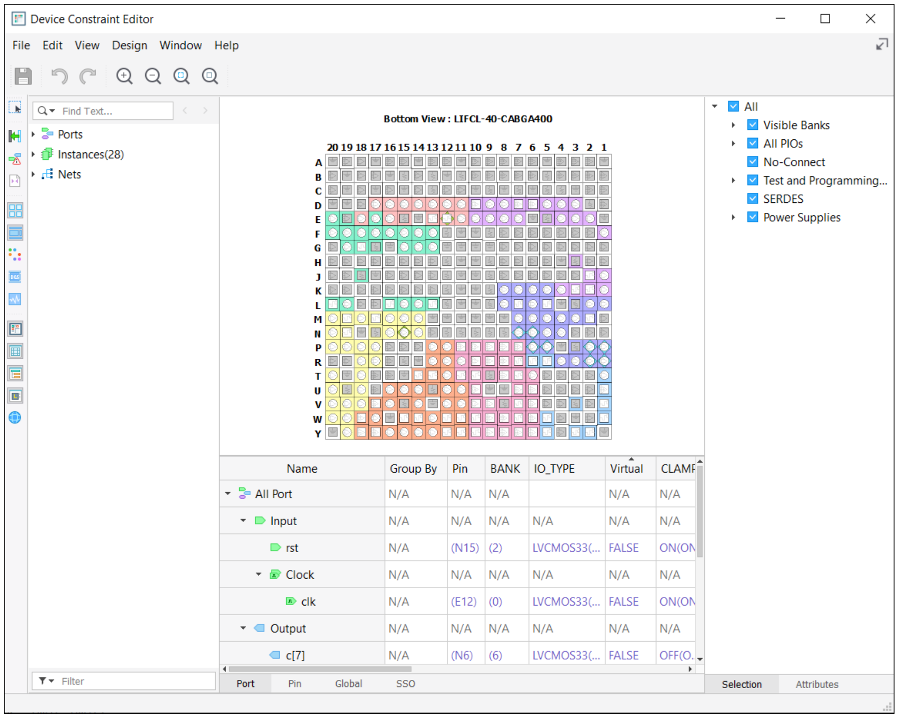
Device Constraint Editor

The Device Constraint Editor is used to edit post-synthesis constraints. These constraint editing views are available from the Radiant toolbar and Tools menu.

All modified constraints are saved to a .pdc file and the flow returned 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 you to edit these assignments, and 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, shown in Figure 65, enable you to develop constraints that will shorten turn-around time and achieve a design that conforms to critical circuit performance requirements.

Figure 65: Device Constraint Editor



Global Settings

In the Device Constraint Editor tool at the bottom of the tool, there are a series of tabs to allow you to set many of the device settings such as Junction Temperature, Voltage, sysCONFIG, User Code, Derating, Bank VCCIO, Global Set/Reset, Use Primary Net, and Vref Locate constraints.

For more information on how to do this, in addition to detailed information about Device Constraint Editor, refer to the Radiant Software Help. See **User Guides > Applying Design Constraints > Applying Radiant Software Physical Constraints > Setting Global Constraints**.

sysCONFIG Settings

Defines system configuration option settings for the sysCONFIG feature. If you do not specify these settings in the .pdc file, using the Global tab in Device Constraint Editor or manually, some default sysCONFIG constraints will automatically be generated based on the device selection.

The SYSCONFIG constraint is available for all Lattice FPGA devices that support the sysCONFIG configuration port. The sysCONFIG port can be a single data line or byte-wide (multiple byte) data port that can support serial and parallel configurations streams. The devices also support daisy chaining.

To set SysConfig parameter values:

1. After opening the design project, choose **Tools > Device Constraint Editor**.
2. Select the Global tab at the bottom of the view.
3. In the SysConfig drop-down setting, for each parameter in the value column, double-click to display the drop-down to select a value or enter the actual value.

SYSCONFIG Keyword Settings

The following table shows the default settings in bold type and the selectable settings for all of the keyword values for the SYSCONFIG constraint. They are ordered as they appear in the Global tab in the Device Constraint Editor.

Device Support

LIFCL, LFCPNX, LFD2NX, LFMXO5, UT24C, UT24CP, LAV-AT

Syntax

ldc_set_sysconfig <keyword>=<value>+

You should set the constraint configuration using the Device Constraint Editor or by editing the .pdc file directly. If you do not set sysCONFIG options, default SYSCONFIG constraints are automatically set.

Examples

The SYCSCONFIG constraint allows you to set a series of keyword values in succession after the ldc_set_sysconfig tcl command.

Figure 66: Example of a SYSCONFIG in the .pdc constraint file

```
ldc_set_sysconfig {JTAG_PORT=ENABLE PROGRAMN_PORT=ENABLE
MCCLK_FREQ=56.2}
```

Table 3: sysCONFIG Values

Keyword	Default and Selectable Values	Supported Devices
SLAVE_SPI_PORT	**DISABLE** [DISABLE, SERIAL, DUAL, QUAD] for Nexus Devices **DISABLE** [DISABLE, SERIAL, DUAL, QUAD, XSPI] for Avant Devices	LAV-AT, LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
MASTER_SPI_PORT	**DISABLE** [DISABLE, SERIAL, DUAL, QUAD] for Nexus Devices **DISABLE** [DISABLE, SERIAL, DUAL, QUAD, XSPI, XSPI_DIFF_CLK] for Avant Devices	LAV-AT, LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
SLAVE_I2C_PORT	**DISABLE** [DISABLE, ENABLE]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
SLAVE_I3C_PORT	**DISABLE** [DISABLE, ENABLE]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
JTAG_PORT	**ENABLE** [ENABLE, DISABLE]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
DONE_PORT	**ENABLE** [ENABLE, DISABLE]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
INITN_PORT	**ENABLE** [ENABLE, DISABLE]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
PROGRAMN_PORT	**ENABLE** [ENABLE, DISABLE]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
BACKGROUND_RECONFIG	**OFF** [OFF, ON, SRAM_EBR, SRAM_ONLY] for Nexus Devices **OFF** [OFF, SRAM_ONLY, ON] for Avant Devices	LAV-AT, LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
DONE_EX	**OFF** [OFF, ON]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
DONE_OD	**ON** [OFF, ON]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
MCCLK_FREQ	See MCCLK section below	LAV-AT, LFCPNX, LFD2NX, LIFCL, UT24C, UT24CP
FLASH_CLK_FREQ	**3.5** [3.5, 7.0, 14.1, 28.1, 56.2, 75, 90, 112.5]	LFMXO5
TRANSFR	**OFF** [OFF, ON] for LAV-AT, LFCPNX, LFMXO5, LIFCL-33U, and UT24CP **OFF** [OFF] for LFD2NX, LIFCL, and UT24C	LAV-AT, LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
CONFIG_IOSLEW	**SLOW** [SLOW, MEDIUM, FAST]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
CONFIGIO_VOLTAGE_BANK0/1	**NOT_SPECIFIED** [NOT_SPECIFIED, 2.5, 1.2, 1.5, 1.8, 3.3] for Nexus devices	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP

Keyword	Default and Selectable Values	Supported Devices
CONFIG_SECURE	**OFF** [OFF, ON]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
WAKE_UP	**ENABLE**[DISABLE]**_DONE_SYNC**	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
COMPRESS_CONFIG	**OFF** [OFF, ON]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
EARLY_IO_RELEASE	**OFF** [OFF, ON]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
BOOTMODE	**DUAL** [DUAL, SINGLE, NONE]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP
MASTER_PREAMBLE_TIMER_CYCLES	**600000** [600000, 400, 2000, 4000, 20000, 40000, 200000, 400000]	LFCPNX, LFD2NX, LFMXO5, LIFCL, UT24C, UT24CP

Table 4: Avant sysCONFIG Values

Keyword	Default and Selectable Values	Supported Devices
BOOT_SEL	**DUAL** [DUAL, SINGLE]	LAV-AT
PROGRAMN_RECOVERY	**DISABLE** [DISABLE, ENABLE]	LAV-AT
MSPI_RESET	**DISABLE** [DISABLE, ENABLE]	LAV-AT
MSPI_RESET_PORT	**DISABLE** [DISABLE, ENABLE]	LAV-AT
MULTI_BOOT_MODE	**DISABLE** [DISABLE, ENABLE]	LAV-AT
DAISY_CHAIN	**DISABLE** [DISABLE, BYPASS, FLOW_THROUGH]	LAV-AT
DAISY_CHAIN_WAIT_DONE	**DISABLE** [DISABLE, ENABLE]	LAV-AT
MSPI_SIGNATURE_TIMER	**200MS** [200MS, 100MS, 50MS, 40MS, 20MS, 1MS, 500US, 100US]	LAV-AT
MSPI_CPOL	**IDLE_CLOCK_LOW** [IDLE_CLOCK_LOW, IDLE_CLOCK_HIGH]	LAV-AT
MSPI_RX_EDGE	**RISING** [RISING, FALLING]	LAV-AT
MSPI_TX_EDGE	**FALLING** [FALLING, RISING]	LAV-AT
MSPI_CPHA	**FIRST_EDGE** [FIRST_EDGE, SECOND_EDGE]	LAV-AT
SSPI_RX_EDGE	**RISING** [RISING, FALLING]	LAV-AT
SSPI_TX_EDGE	**FALLING** [FALLING, RISING]	LAV-AT
SSPI_CPHA	**FIRST_EDGE** [FIRST_EDGE, SECOND_EDGE]	LAV-AT
SSPI_CPOL	**IDLE_CLOCK_LOW** [IDLE_CLOCK_LOW, IDLE_CLOCK_HIGH]	LAV-AT
SSPI_SHIFT_ORDER	**MSB_FIRST** [MSB_FIRST, LSB_FIRST]	LAV-AT
SSPI_DAISY_CHAIN_MODE	**DISABLE** [DISABLE, SCM, AUTO]	LAV-AT

Table 4: Avant sysCONFIG Values

Keyword	Default and Selectable Values	Supported Devices
ERASE_EBR_ON_REFRRESH	**ENABLE** [ENABLE, DISABLE]	LAV-AT
SIGNATURE_CHECK	**ENABLE_LSCC_SIGNATURE** [ENABLE_LSCC_SIGNATURE, DISABLE, ENABLE_SFDP_SIGNATURE]	LAV-AT
MSPI_ADDRESS_32BIT	**DISABLE** [DISABLE, ENABLE]	LAV-AT
MSPI_COMMAND_32BIT	**DISABLE** [DISABLE, ENABLE]	LAV-AT
SSPI_IDLE_TIMER	**DISABLE** [DISABLE, 200S, 100S, 50S, 25S, 10S, 5S, 1S, 750MS, 500MS, 250MS, 100MS, 75MS, 50MS, 25MS, 10MS]	LAV-AT
MSPI_PREAMBLE_DETECTION_TIMER	**200MS** [200MS, 100MS, 50MS, 40MS, 20MS, 1MS, 500US, 100US]	LAV-AT
MULTI_BOOT_SEL	**STATIC** [STATIC, DYNAMIC]	LAV-AT
CONFIGIO_VOLTAGE_BANK1/2	**NOT_SPECIFIED** [NOT_SPECIFIED, 2.5,1.2,1.8, 3.3]	LAV-AT

SLAVE_SPI_PORT

DISABLE (default) | SERIAL | DUAL | QUAD – Nexus Devices

DISABLE (default) | SERIAL | DUAL | QUAD | XSPI – Avant Devices

Specifies if the TARGET_SPI port should be available after configuration. This option will preserve the dual-purpose pins for PAR.

MASTER_SPI_PORT

DISABLE (default) | SERIAL | DUAL | QUAD – Nexus Devices

DISABLE (default) | SERIAL | DUAL | QUAD | XSPI | XSPI_DIFF_CLK – Avant Devices

Specifies if the CSPI port should be available after configuration. Used for background programming of the connected SPI flash. This option will preserve the dual-purpose pins for PAR. When enabled, it allows the use of the external Controller SPI port for configuring the SRAM fuses using the bitstream.

Note

As part of the Lattice inclusive language guidelines, the Avant device's MASTER_SPI_PORT has been changed to CONTROLLER_SPI_PORT and SLAVE_SPI_PORT to TARGET_SPI_PORT. If you want to know more about the Lattice Inclusive Language, please visit Lattice Answer Database.

SLAVE_I2C_PORT

DISABLE (default) | ENABLE

Specifies if the SLAVE I2C port should be available after configuration. This option will preserve the dual-purpose pins for PAR.

SLAVE_I3C_PORT

DISABLE (default) | ENABLE

Specifies if the SLAVE I3C port should be available after configuration. This option will preserve the dual-purpose pins for PAR.

JTAG_PORT

ENABLE (default) | DISABLE

Specifies if the JTAG port should be available after configuration. This option will preserve the dual-purpose pins for PAR. When ENABLED, it is used for configuration settings only, PAR will prohibit them. When DISABLED, it can be used as GPIOs.

DONE_PORT

ENABLE (default) | DISABLE

When set to ENABLE, the pin is not available in user mode.

INITN_PORT

ENABLE (default) | DISABLE

When set to ENABLE, the pin is not available in user mode.

PROGRAMN_PORT

ENABLE (default) | DISABLE

When set to ENABLE, the pin is not available in user mode.

BACKGROUND_RECONFIG

OFF (default) | ON | SRAM_EBR | SRAM_ONLY – Nexus Devices

OFF (default) | SRAM_ONLY | ON – Avant Devices

DONE_EX

OFF (default) | ON

You can select if the device should wake up on its own after the DONE bit is set or wait for an external DONE signal to drive the DONE pin high. The

DONE_EX attribute will determine if the wake up sequence is driven by an external DONE signal.

If set to ON, the user wishes to delay wake up until the DONE pin is driven high by an external signal and synchronous to the clock. You can select OFF if you want to synchronously wake up when the DONE bit is set and ignore any external driving of the DONE pin.

DONE_OD

ON (default) | OFF

Done Pin Open Drain. Enables you to configure the DONE pin as an open drain pin. By default, the pullup on the DONE pin is active.

The DONE pin used in sysCONFIG to indicate that configuration is done. The DONE pin can be linked to other DONE pins and used to initiate the wake up process. The DONE pin can be open collector or active drive. Default is ON and insures the user needs to make a conscious decision to drive the pin.

MCCLK_FREQ

3.5 (default) | 7.0 | 14.1 | 28.1 | 56.2 | 75 | 90 | 112.5 | 150 – Nexus devices.

3.1 (default) | 7.1 | 14.3 | 28.6 | 57.1 | 66.7 | 80.0 | 100.0 | 106.7 | 133.3 | 160.0 – Avant devices.

Controls the Master Clock frequency. When a master configuration mode is used, MCLK will become an output clock with the frequency set by the user. Until the device is configured, the default Master Clock frequency is the lowest frequency support by the device.

For LFCPNX, LFD2NX,LIFCL, UT24C, and UT24CP devices, a 450 Mhz oscillator is divided and made available for configuration.

For LAV-AT devices, a 400 Mhz oscillator is divided and made available for configuration.

FLASH_CLK_FREQ

3.5 (default) | 7.0 | 14.1 | 28.1 | 56.2 | 75| 90 | 112.5

TRANSFR

OFF (default) for LAV-AT, LFD2NX, LIFCL, and UT24C. ON [default] for LFCPNX and UT24CP

For the current supported devices, only OFF is available. When ON is selected for a future supported device,it sets the CR0 TransFR option. This will also enable the Hold IO type.

CONFIG_IOSLEW

SLOW (default) | MEDIUM | FAST

Controls config output pin slew rate. Sets CR0[7:6].

CONFIGIO_VOLTAGE_BANK0/1

NOT_SPECIFIED (default) | 2.5 | 1.2 | 1.5 | 1.8 | 3.3 – Nexus Devices

The CONFIGIO_VOLTAGE_BANK0/1 attribute supports all devices except LAV-AT and iCE40UP.

Setting this attribute informs the software which voltage is required in Bank0/1 to satisfy the user's sysCONFIG requirements. The sysCONFIG pins used for configuration may or may not be used in the design and hence the user shall be able to declare the voltage interface for this bank.

DRC errors can then be generated based on CONFIG_IOVOLTAGE and usage of the dual-purpose sysCONFIG pins. BANK0 and 1 values can be different and a combination of the available values including NOT_SPECIFIED.

CONFIGIO_VOLTAGE_BANK1/2

NOT_SPECIFIED (default) | 2.5 | 1.2 | 1.8 | 3.3

The CONFIGIO_VOLTAGE_BANK1/2 attribute only supports the LAV-AT devices.

CONFIG_SECURE

OFF (default) | ON

Configuration Security. When set to ON, no readback operation is supported through the sysCONFIG port or ispJTAG port of the general contents. The USERCODE area is readable and not considered securable. The OFF setting indicates that readback is enabled through any port. Prevents readback of the configuration memory contents.

WAKE_UP

ENABLE/DISABLE_DONE_SYNC (DONE_EX = OFF) /(DONE_EX = ON)

Wake Up. Wake Up is a controlled event after a part has been configured. External control of the DONE pin can be selected to either delay wake up or be ignored. See DONE_EX.

The Wake Up sequence controls three internal signals, and the DONE Pin will be driven after configuration and prior to user mode. If DONE_EX = ON, the WAKE_UP keyword will take your selected option (1-7), and then the software will set wake-up signal bits accordingly. If you do not select a wake-up sequence, the default wake-up sequence will be 4. If DONE_EX = OFF (default), the WAKE_UP keyword will take your selected option (1-25), and then the software will set wake-up signal bits accordingly. If you do not select a wake-up sequence, the default wake-up sequence will be 21.

COMPRESS_CONFIG

OFF (default)

Bitstream Compression. When set to ON, for those devices that support bitstream compression, the software generates a compressed version of the bitstream file.

EARLY_IO_RELEASE

OFF (default) | ON [once]

Program and wakeup the left/right I/O banks early, before the rest of the device is programmed.

BOOTMODE

DUAL (default) | SINGLE | NONE

This setting enables you to select the booting mode at power up.

- DUAL - Performs the dual boot for the booting event. In case the primary booting image (bitstream) fails, the secondary boot image is invoked to boot the device.
- SINGLE - Performs a single boot only. If it fails, the device will move to unprogrammed mode directly.
- NONE - No controller SPI booting at startup, waits for target configuration port to configure the device.

MASTER_PREAMBLE_TIMER_CYCLES

600000 (default) | 400 | 2000 | 4000 | 20000 | 40000 | 200000 | 400000

This attribute sets the master preamble timer count value.

The preamble_count value below is the number of clock cycles to get a valid preamble before an error is declared.

000 -> 600000

001 -> 400000

010 -> 200000

011 -> 40000

100 -> 20000

101 -> 4000

110 -> 2000

111 -> 400

BOOT_SEL

DUAL (default) | SINGLE

The BOOT_SEL preference allows you to select the booting mode (DUAL or SINGLE) at power up (POR), PROGRAMN pin toggle or REFRESH command execution with the following available options:

- DUAL (default) - Performs the dual boot for the booting event. In case the primary booting image (bitstream) failed, the secondary booting image is automatically invoked to boot up the device.
- SINGLE - Performs the single boot only. If it fails, the device goes directly to unprogrammed mode.

PROGRAMN_RECOVERY

DISABLE (default) | ENABLE

The PROGRAMN_RECOVERY preference allows recovery from Secure Non-Operation (SNOP) state using PROGRAMN with the following available options:

- DISABLE (default) - Disables recovery from Secure Non-Operation (SNOP) state using PROGRAMN.
- ENABLE - Enables recovery from Secure Non-Operation (SNOP) state using PROGRAMN.

MSPI_RESET

DISABLE (default) | ENABLE

Enables the hardware reset signal to the flash device.

The MSPI_RESET attribute has the following options:

- DISABLE - Enables reset port to flash device is disabled.
- ENABLE - Enables reset port to flash device.

MSPI_RESET_PORT

DISABLE (default) | ENABLE

Specifies if the MSPI_RESET port is available to use for configuration purposes during usermode.

The MSPI_RESET_PORT attribute has the following options:

- DISABLE - Not available for configuration during usermode (user IO).
- ENABLE - Available for configuration during usermode (persisted as config IO).

MULTI_BOOT_MODE

DISABLE (default) | ENABLE

The MULTI_BOOT_MODE preference allows you to enable the multi-boot feature. It has the following options:

- DISABLE - Disables multi-boot feature.
- ENABLE - Enables the multi-boot feature.

DAISY_CHAIN

DISABLE (default) | BYPASS | FLOW_THROUGH

You can select the daisy chain mode for upstream device using DAISY_CHAIN preference with the following available options:

- DISABLE - Daisy Chain is disabled (default).
- BYPASS - Enables the transparent SRAM access mode.
- FLOW_THROUGH - Enables the transparent SRAM or/and INIT Registers access mode.

DAISY_CHAIN_WAIT_DONE

DISABLE (default) | ENABLE

The DAISY_CHAIN_WAIT_DONE preference allows you to wait for the done pin before “wakeup” for the daisy chain upstream. It has the following options:

- DISABLE (default) - Does not wait for DONE.
- ENABLE - Waits for DONE.

MSPI_SIGNATURE_TIMER

200MS (default) | 100MS | 50MS | 40MS | 20MS | 1MS | 500US | 100US

MSPI_SIGNATURE_TIMER allows you to specify the time to get a valid LSCC/SFDP signature.

MSPI_CPOL

IDLE_CLOCK_LOW (default) | IDLE_CLOCK_HIGH

MSPI_CPOL specifies if the clock is low or high in idle state.

- IDLE_CLOCK_LOW (default) - Clock is low in idle state.
- IDLE_CLOCK_HIGH - Clock is high in idle state.

MSPI_RX_EDGE

RISING (default) | FALLING

Enables the selection of rising/falling edge for RX.

MSPI_TX_EDGE

FALLING (default) | RISING

Enables the selection of rising/falling edge for TX.

MSPI_CPHA

FIRST_EDGE (default) | SECOND_EDGE

Enables the selection of the clock edge combined with MSPI_CPOL.

SSPI_RX_EDGE

RISING (default) | FALLING

Enables the selection of rising/falling edge for RX.

SSPI_TX_EDGE

FALLING (default) | RISING

Enables the selection of rising/falling edge for TX.

SSPI_CPHA

FIRST_EDGE (default) | SECOND_EDGE

Enables the selection of the clock edge combined with SSPI_CPOL.

SSPI_CPOL

IDLE_CLOCK_LOW (default) | IDLE_CLOCK_HIGH

SSPI_CPOL specifies if the clock is low or high in idle state with the following available options:

- IDLE_CLOCK_LOW (default) - Clock is low in idle state.
- IDLE_CLOCK_HIGH - Clock is high in idle state.

SSPI_SHIFT_ORDER

MSB_FIRST (default) | LSB_FIRST

The SSPI_SHIFT_ORDER bit specifies the SSPI shift order with the following available options:

- MSB_FIRST (default) - Shifts the most significant bit first.
- LSB_FIRST - Shifts the least significant bit first.

SSPI_DAISY_CHAIN_MODE

DISABLE (default) | SCM | AUTO

The SSPI_ DAISY_CHAIN_MODE allows you to enable the SSPI auto function for downstream daisy chain with the following available options:

- DISABLE (default) - Disables SSPI Auto/SCM mode.

- SCM - Persists CLK and MOSI.
- AUTO - SPI Auto mode based on SSPI_PORT selection for SERAIL, DUAL, QUAD or XSPI.

ERASE_EBR_ON_REFRRESH

ENABLE (default) | DISABLE

The ERASE_EBR_ON_REFRRESH allows you to enable or disable “erase” on refresh.

SIGNATURE_CHECK

ENABLE_LSCC_SIGNATURE (default) | DISABLE | ENABLE_SFDP_SIGNATURE

The SIGNATURE_CHECK preference allows you to check the signature with the following available options:

- ENABLE_LSCC_SIGNATURE (default) - Checks the LSCC signature.
- DISABLE - Skips signature checking.
- ENABLE_SFDP_SIGNATURE - Checks SFDP signature.

MSPI_ADDRESS_32BIT

DISABLE (default) | ENABLE

The MSPI_ ADDRESS_32BIT enables you to send the 32-bit command set on MSPI with the following available options:

- DISABLE (default) - 24-bit command set.
- ENABLE - 32-bit command set.

MSPI_COMMAND_32BIT

DISABLE (default) | ENABLE

The MSPI_ ADDRESS_32BIT enables the 32-bit MSPI addressing mode with the following available options:

- DISABLE (default) - 24-bit MSPI addressing mode.
- ENABLE - 32-bit MSPI addressing mode.

SSPI_IDLE_TIMER

DISABLE (default) | 200S | 100S | 50S | 25S | 10S | 5S | 1S | 750MS | 500MS | 250MS | 100MS | 75MS | 50MS | 25MS | 10MS

The SSPI_IDLE_TIMER allows you to specify the time to get a valid preamble.

MSPI_PREAMBLE_DETECTION_TIMER

200MS (default) | 100MS | 50MS | 40MS | 20MS | 1MS | 500US | 100US

The MSPI_PREAMBLE_DETECTION_TIMER allows you to specify the time to get a valid preamble.

MULTI_BOOT_SEL

STATIC (default) | DYNAMIC

The MULTI_BOOT_SEL preference allows you to select the multi-boot address. It has the following options:

- STATIC - Selects the multi-boot address from sysConfig MULTI_BOOT_OFFSET preference.
- DYNAMIC - Selects the multi-boot address from the MULTI_BOOT_OFFSET parameter setting of CONFIG_LMMIC primitive.

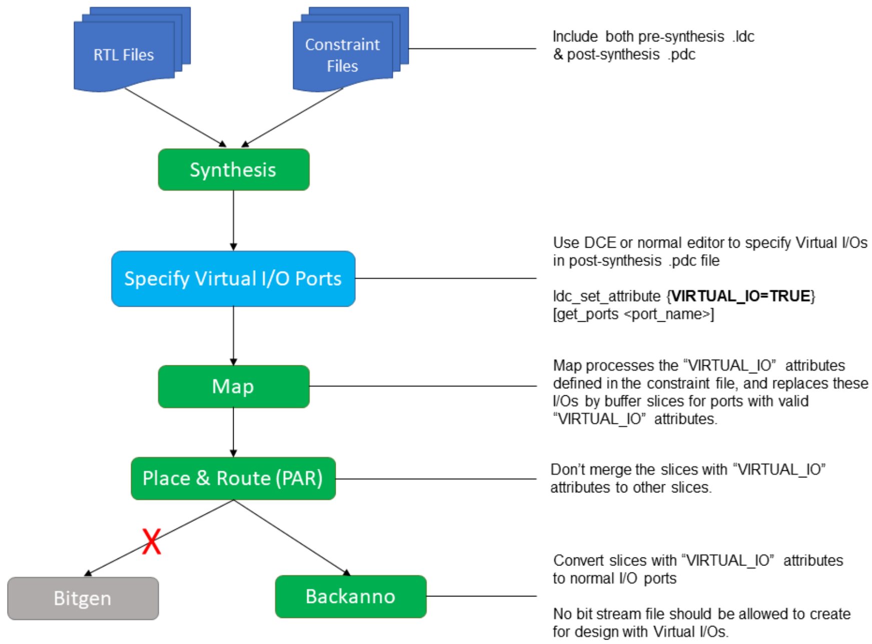
SSO Analysis

The Radiant software enables you to run an analysis of simultaneous switching outputs (SSO). SSO analysis describes the noise on signals caused by a large number of output drivers that are switching at the same time. Analysis of SSO helps ensure that your I/O standards and power integrity meet requirements of the PCB design. This tool can be accessed in the Device Constraint Editor in the SSO tab at the bottom. From there you can set output loads, ground plane PCB noise, SSO Allowance %, and power plane PCB noise values. Once the flow is re-run, an SSO report is generated.

Specifying Virtual I/O Ports

The Radiant Virtual I/O flow is useful in checking the design timing, resource utilization, or performance of a selected device to determine if your modules exceeded the number of physical pins present in the FPGA (e.g., during an IP design). Without the Virtual I/O flow, the implementation will result in an error due to the number of I/Os exceeding the device capacity. You can use the Virtual I/O flow to specify certain ports as "virtual" ports at the top level.

Refer to the following flow chart.



A Virtual I/O is terminated in the LUT instead of a PIO. This is reported as the "Number of feedthrough LUT4s" in the MAP report, and is not included in the number of LUTs in the design implementation.

If the total number of real ports in the design is less than or equal to the total number of I/O ports in the device, you can use the regular Radiant flow for such IP without the Virtual I/O flow.

To specify the Virtual I/O ports in the selected I/Os:

- In the .pdc file using ldc_set_attribute.

Syntax:
```
ldc_set_attribute {VIRTUAL_IO=TRUE} [get_ports
<port_name>]
ldc_set_attribute {VIRTUAL_IO = TRUE} [get_ports ch0_sdrxp]
```

Notes:

Wildcard "*" is supported for the port names:

```
ldc_set_attribute {VIRTUAL_IO = TRUE} [get_ports datain*]
```

- Using the DCE Spreadsheet View Virtual column, set the required ports value to "TRUE." By default, the value is set to "FALSE."

See Also

- "Checking the Virtual I/O Ports" on page 139
- "Verifying Virtual I/O Ports" on page 140

Checking the Virtual I/O Ports

The Virtual I/O attribute for a port is ignored if one of the following conditions is met:

- The I/O port is with tristate signal.

  e.g. tristate output or bi-directional I/O. Tristate support is only in I/O port.

- The I/O port is with LOCATION constraint.

  When user specifies the location (either to pin or bank), it is a real external I/O.

- The I/O port is with IO_TYPE attribute in RTL or constraint.

  When user specifies IO_TYPE, it is a real external I/O. Virtual I/O should not be specified for the ports with IO_TYPE attribute.

- The I/O port is connected with DELAY module.
- The I/O port is not a simple input buffer (IB) or output buffer (OB) but a special I/O buffer, either from an IP or RTL instantiation.

  e.g. MIPI, SEIO18_CORE, DIFFIO18_CORE, etc.

  Its special usage means it is a real external I/O.

- The I/O port drives a clock signal or is driven by a clock signal.
- The I/O port is connected to an external I/O directly in a hardware device, such as:
  - APIO pin of a component (e.g. High speed I/O in PCIE)
  - JTAG ports
  - Some ports of ADC/PMU/I2C etc. components
  - IDDR's D input
  - ODDR's Q output
  - DQS's DQSI input

Notes:

I/O registers are treated differently in the LFCPNX (CertusPro-NX) and UT24CP (CertusPro-NX-RT) device versus the iCE40UP (iCE40 UltraPlus) device. For the LFCPNX and UT24CP device, IOLOGIC is a whitebox cell model. I/O registers are mapped into IOLOGIC. So, when IREG's D input or OREG's Q output is a Virtual I/O port, the I/O REG will be pushed into PLC (SLICE).

However, for the iCE40UP device, I/O register is in IOLOGIC black box. It is packed by the synthesis tool. As a result, the Virtual I/O constraints will be ignored for ports associated with I/O registers.

See Also

- "Specifying Virtual I/O Ports" on page 137
- "Verifying Virtual I/O Ports" on page 140

Verifying Virtual I/O Ports

There are several ways to verify Virtual I/Os.

Viewing Virtual I/Os in DCE If you have specified the Virtual I/O attributes for some I/O ports in the Post-Synthesis constraint file (.pdc), you can see them in the "Virtual" column. As shown in the following figure, the ports "datain[0]" to "datain[7]" are virtual I/Os.

Name	Group By	Pin	BANK	IO_TYPE	Virtual	CLAMP
clk	N/A	(E12)	(0)	LVCMOS33(LVCM...	FALSE	ON(ON)
datain[0]	N/A	N/A	N/A	N/A	TRUE	N/A
datain[1]	N/A	N/A	N/A	N/A	TRUE	N/A
datain[2]	N/A	N/A	N/A	N/A	TRUE	N/A
datain[3]	N/A	N/A	N/A	N/A	TRUE	N/A
datain[4]	N/A	N/A	N/A	N/A	TRUE	N/A
datain[5]	N/A	N/A	N/A	N/A	TRUE	N/A
datain[6]	N/A	N/A	N/A	N/A	TRUE	N/A
datain[7]	N/A	N/A	N/A	N/A	TRUE	N/A

Changing Virtual I/O attributes in DCE

If you save the result, the Radiant software will ask you to rerun the Map Design and Place & Route Design processes, since the .pdc file has been changed.

Viewing Virtual I/Os in the Map Report file

The number of Virtual I/Os are reported in the Map Report (.mrp) file. Here is an example:

```
Number of LUT4s:                      8 out of 32256 (<1%)
 Number used as logic LUT4s:          0
 Number used as distributed RAM:      0 (6 per 16X4 RAM)
 Number used as ripple logic:         0 (2 per CCU2)
 Number of inserted feedthrough LUT4s:8
 Number for virtual IO:               8
Number of Virtual IOs:                8
```

The report shows that the "Number of Virtual I/Os" is 8, and it used 8 LUTs to replace the specified Virtual I/O ports. These eight LUTs are part of "Number of inserted feedthrough LUT4s."

You can find the individual Virtual I/Os in the "I/O (PIO) Attributes" section in the same .mrp file. If a port is a Virtual I/O, the value is marked as "VIRTUAL" in column "Levelmode IO_TYPE."

I/O Name	Direction	Levelmode IO_TYPE	I/O REG	I/O DDR	Special I/O Buffer
clk	INPUT	LVCMOS33			
datain[7]	INPUT	**VIRTUAL**			
datain[6]	INPUT	**VIRTUAL**			
datain[5]	INPUT	**VIRTUAL**			

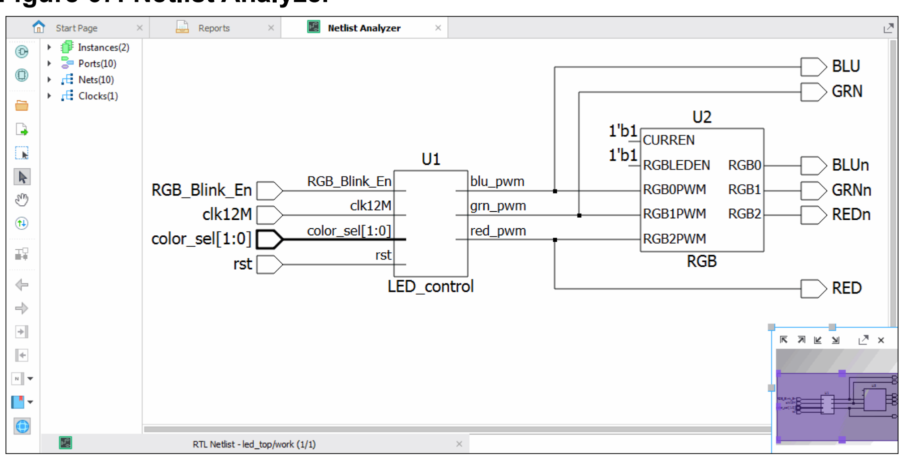
See Also

- "Specifying Virtual I/O Ports" on page 137
- "Checking the Virtual I/O Ports" on page 139

Netlist Analyzer

Netlist Analyzer works with Lattice Synthesis Engine (LSE) and Synplify Pro 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.The Netlist Analyzer window has four parts, as shown in Figure 67.

Figure 67: Netlist Analyzer



- Tool bar provides buttons for various functions.
- Browser provides nested lists of module instances, ports, and nets.
- Schematic view shows a schematic of the design. Depending on the size of the design, the schematic may be made of multiple sheets.

- World View, 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.

There are several ways to adjust the view of a schematic and to navigate through the hierarchy. For more information on how to do this, in addition to detailed information about Netlist Analyzer, refer to the Radiant Software Help. See **User Guides > Managing Projects > Analyzing a Design**.

Physical Designer

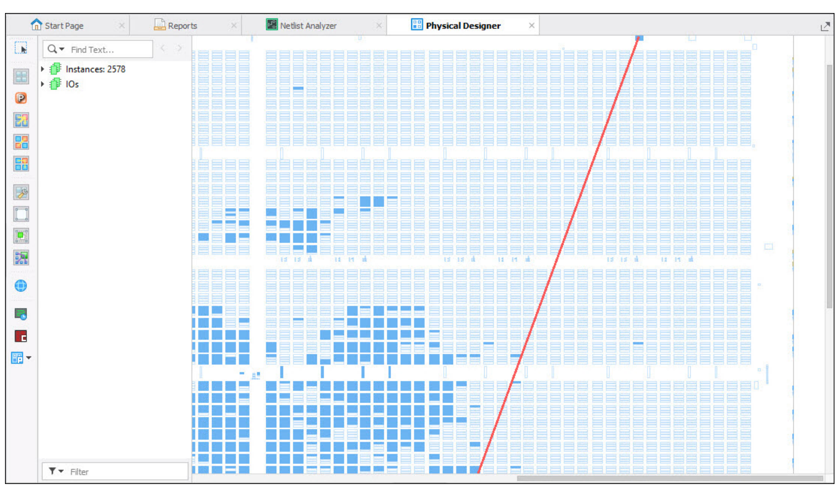
Physical Designer provides one central location where you can do all the floorplanning and be able to view the physical layout of the design. Physical Designer does this with Placement, IO, and Routing modes.

Placement Mode

Placement Mode provides a large-component layout of your design, and is available as soon as the target device has been specified. Placement Mode displays user constraints, and placement and routing information. All connections are displayed as fly-lines. Placement Mode allows you to create REGION and GROUP constraints to keep components together. You can specify the types of components and connections to be displayed. As you move your mouse pointer over the floorplan layout, details are displayed in tool-tips and in the status bar, including:

- Number of resources for each GROUP and REGION
- Number of utilized slices for each PLC component
- Name and location of each component, port, net, and site

Figure 68: Placement Mode

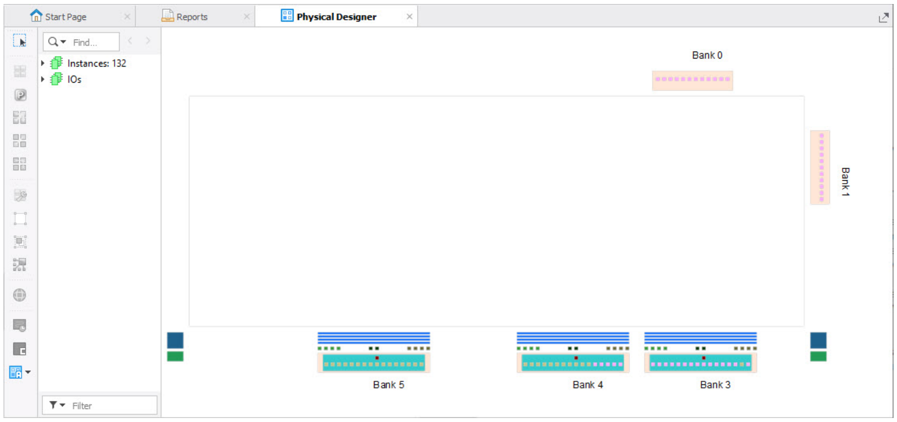


Placement Mode has a Congestion Timing Map view allowing you to debug timing congested areas of your design. This view gives the timing of the most critical paths within the slack threshold input. There is also the Congestion View, which is a read-only view showing the most congested regions based on wire length and pins selected.

IO Mode

IO Mode is used for I/O assignments such as DDR Interface, DQS, and clock assignments. I/O resource utilization can be displayed by hovering and displayed in tool tips.

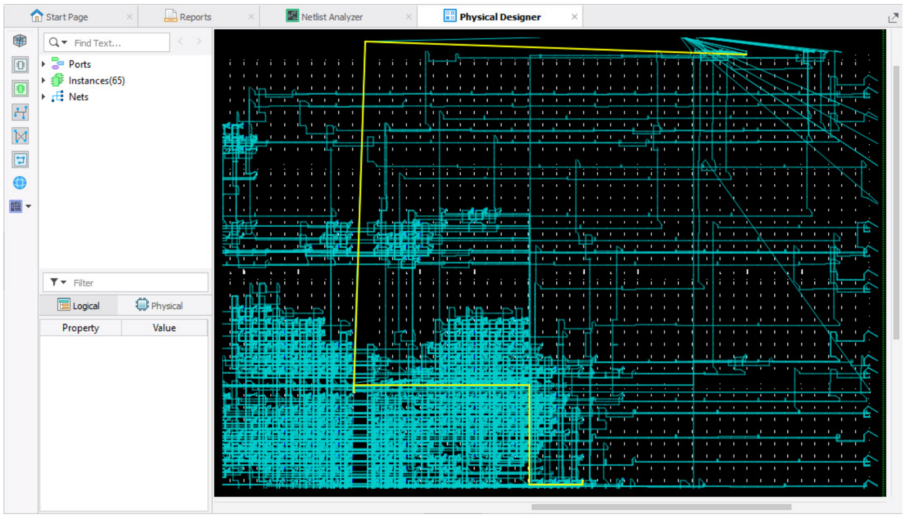
Figure 69: IO Mode



Routing Mode

Routing Mode provides a read-only detailed layout of your design that includes physical wire connections. Routed connections are displayed as Manhattan-style lines, and unrouted connections are displayed as fly-lines.

Figure 70: Routing Mode



As you move your mouse pointer slowly over the layout, the name and location of each REGION, group, component, port, net, and site are displayed as tool tips and also appear in the status bar. The tool tips and status bar also display the group name for components that are members of a group.

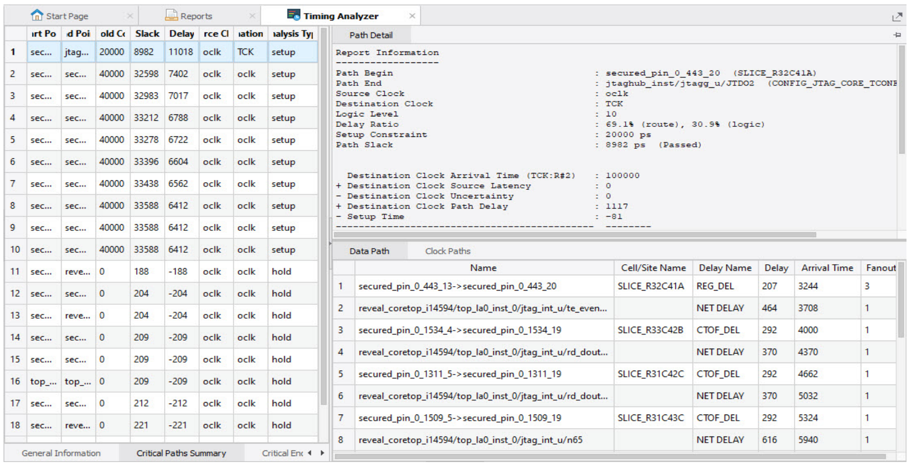
The Routing Mode toolbar allows you to select the types of elements that will be displayed on the layout, including components, empty sites, pin wires, routes, and timing paths. Routing Mode is available after placement and routing.

Timing Analyzer

Timing Analyzer 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 the LSE, Map Timing Analysis process, or the Place & Route Timing Analysis (PAR) process. Place & Routing Timing Analysis results can then be viewed in the Timing Analyzer 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.

Figure 71: Timing Analyzer



Timing Analyzer consists of five tabs of information:

- General Information
- Critical Paths Summary
- Critical Endpoint Summary
- Unconstrained Endpoint Summary
- Query

Each tab can be detached from the main window, rearranged, and resized. When you select a constraint from the Constraint pane, you can view the path table details on one pane, and Timing Analyzer report in the other. For detailed information about Timing Analyzer, see **User Guides > Analyzing Static Timing** in the Radiant Software Help.

Using Standalone Timing Analyzer

Standalone Timing Analyzer can be run on designs using post-synthesis, post-map, post-PAR Unified Design Database (.udb), and associated timing constraints specified in the .pdc file of the design. Using these input files, it provides static timing analysis, path tables, and a report of each timing constraint like the Timing Analyzer in Radiant.

Running Standalone Timing Analyzer from Radiant Timing Analyzer

When you open the Standalone Analyzer tool, it will automatically create the Unified Design Database, which includes logic and physical data (standalone_tav_*.udb). This file is saved to the current project. You can use this database to run tavmain.exe.

To open and run a timing analysis in Standalone Timing Analyzer:

- In the Radiant software main window, click the button in the upper left. You can select a different .udb file.
- If you wish to open the Standalone Timing Analyzer directly, click the button in the upper left or choose **Tools** > **Standalone Timing Analyzer**.

See Also

- "Standalone Timing Analyzer" on page 262

Loading Timing Constraint File and Running Timing Analysis in Standalone Timing Analyzer

To load a timing constraint file and run timing analysis:

1. In the Timing Analyzer main window, choose **File** > **Load Timing Constraint File**. This menu can be used after opening the design.
2. After selecting the constraint file, you can load the timing constraint and do the timing analysis based on the selected constraint file.
3. If you select Load Timing Constraint File (Incremental Flow), do the timing analysis based on the constraints in the .udb file and the selected constraint file.

Exporting Timing Report and Exporting Timing Analysis Reports in Standalone Timing Analyzer

To export timing report and timing analysis report:

- In the Timing Analyzer main window, choose **File** > **Export** > **Timing Report For Setup/Hold**. This menu can export timing analysis reports for setup or hold (.twr).

Exporting Timing Constraints to PDC File in Standalone Timing Analyzer

To export timing constraints to PDC file:

- In the Timing Analyzer main window, choose **File** > **Export** > **Export Timing Constraints To PDC file**.

  This menu can be used after opening the design.
- You can export timing constraints in the current .udb file to a .pdc file.

Saving Design Files in Standalone Timing Analyzer

To save design files in Standalone Timing Analyzer:

- You can save changes to design data. In the Timing Analyzer main window, choose **File** > **Save Design As**.

Setting Options in Standalone Timing Analyzer

Setting options in Standalone Timing Analyzer is the same as setting options in Radiant Timing Analyzer." Refer to the Radiant Help. See **User Guides > Analyzing Static Timing > Using Timing Analyzer > Setting Options in Timing Analyzer**.

To set options in Timing Analyzer:

- In the Timing Analyzer main window, choose Edit > Timing Option Setting, or click the  button in the upper left.

  This menu can be used after opening the design.
- When loading the post-synthesis .udb file, you cannot change the speed in the Timing Option Setting dialog box. This feature is the same as the timing report.
- The default for the Report Format is Lattice Standard. You can change the Report Format in the Timing Option Setting dialog box.

Setting Operating Conditions in Standalone Timing Analyzer

To set operating conditions in Standalone Timing Analyzer:

1. In the Timing Analyzer main window, choose **Edit** > **Operating Conditions Setting**.

   You can change the voltage and temperature based on the current design data.

2. When you click the **OK** button, the timing analysis process starts based on the edited design data.

3. When loading the post-synthesis .udb file, you cannot change the settings in the Operating Conditions Setting dialog box. This feature is the same as timing report.

Running Timing Constraint Editor in Standalone Timing Analyzer

To run timing constraint editor in Standalone Timing Analyzer:

1. In the Timing Analyzer main window, choose **Edit** > T**iming Constraint Editor**. After selecting the Timing Constraint Editor, you can edit the timing constraint design based on the current design data.

2. When you close the Post-Synthesis Timing Constraint Editor window, a dialog box appears. If you click yes, the timing analysis process starts based on the edited design data.

Refreshing Timing in Standalone Timing Analyzer

To refresh timing in Standalone Timing Analyzer:

- In the Timing Analyzer main window, choose **Edit** > **Refresh**. Based on the current timing constraint in the .pdc file, you can do the timing analysis to check if the new constraint works as expected.

To edit the location of the current .pdc file: ▶You can add the new constraint in the current .pdc file.

Click the **Refresh** button to start the timing analysis process.

- To add the new constraint in the .pdc file, you can copy the text from the timing table.

- If the location of the current .pdc file is not specified, you can also choose **File** > **Load Timing Constraint File** to specify the .pdc file and start the timing analysis process.

Reveal Inserter

Reveal Inserter lets you add debug information to your design to allow hardware debugging using Reveal Analyzer. Reveal Inserter enables you to select the design signals to use for tracing, triggering, and clocking. Reveal Inserter will automatically generate the debug core, and insert it into a modified design with the necessary debug connections and signals. Reveal Inserter supports VHDL and Verilog sources. After the design has been modified for debug, it is mapped, placed and routed with the normal design flow in the Radiant software.

For more information about Reveal Inserter, refer to the **Reveal User Guide for Radiant Software**. Also, refer to **User Guides > Testing and Debugging On-Chip** in the Radiant Software Help.

Using the Reveal Controller Simulation Model

The Reveal Controller Simulation Model replicates the functionality of Reveal Controller in simulation.

To set up the Reveal Controller Simulation Model:

1. Choose **Tools > Reveal Inserter** to insert a reveal module into the design.
2. Run **Synthesize Design** to generate the reveal_workspace file.

- If you insert Reveal Controller only, the tool will create the sim_src folder with the pre-synthesis Reveal generated source file under reveal_workspace.
- If you insert Reveal Controller and Analyzer, the Reveal engine will not create the folder and no source files will be provided.

3. Click **Tools > Simulation Wizard** or click the icon in the toolbar.
4. In the Simulator Project Name and Stage dialog box, enter the project name.

   Click **Next**.
5. In the Add and Reorder Source dialog box, select and add the source files from the sim_src folder.

   Click **Next**.

The following compiler directives are all enabled in the source file. You can use the OEM simulator or any stand-alone simulator to compile and simulate the Controller:

- `define en_sw
- `define en_led
- `define en_user_mem
- `define en_user_creg
- `define en_user_sreg

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. The number of triggers is specified in Reveal Analyzer unless you are creating a module to monitor the power-on reset (POR) functions (see below).
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.
3. If you select Multiple Trigger Capture and are creating a POR module, choose the “Number of triggers for POR.” For an explanation of POR modules, see “Power-on Reset (POR) Debug” on page 149.

Power-on Reset (POR) Debug

An automatic "trigger enable" signal must be built into the Reveal module to monitor POR functions. Before Reveal Analyzer starts, these functions happen immediately after the power-on of the test board. When the trigger enables signal transitions to "active," the module will watch for the trigger and collect samples. It is similar to clicking the Run button in Reveal Analyzer. When Reveal Analyzer starts, it loads and displays any data collected by the POR modules.

POR modules are not supported in iCE40UP devices.

To set a POR trigger enable:

1. In the POR Debug section, select **Trigger Enable**.
2. Find the POR trigger signal in the Design Tree view and drag it to the text box in the POR Debug section.
3. Choose whether the signal is **Active High** or **Active Low**.

Configuring User Memory Setup

Setting up the User Memory setting consists of assigning six mandatory ports. (which must be defined in user RTL design) The maximum data width is 32 bits.

The user design must include a memory block (EBR, Distributed Memory or PMI Memory) to be able setup User Memory.

The mandatory signals in the user design to setup User Memory are:

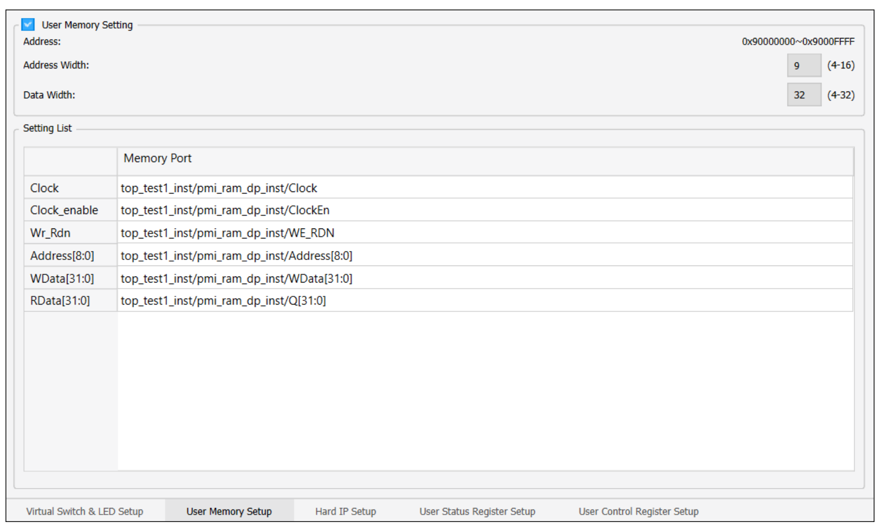
- Clock
- Clock_enable
- Wr_Rdn
- Address
- WData
- RData

To set up User Memory signals:

1. Select the top_Controller core in the Dataset view. (There can only be one Controller core.)

   When you select Add Controller, it displays the control panel. You can enable User Memory by choosing the corresponding memory port names. The Address and Data Width will be updated automatically.

2. Click the **User Memory Setup** tab.



3. Select signals in the Design Tree and drag the signals to the desired position in the Setting List in the User Memory Setup tab. For help finding signals, refer to the Radiant Software Help. See **User Guides > Testing and Debugging On-Chip > Creating Reveal Modules > About Reveal Inserter > Searching for Signals**.

4. Drag the desired signals in to the User Memory data panel to the right. The width of the signals will automatically update between 4-32 entries at the top.

5. Beside the title User Memory Setting, there is a check box in which Controller function for User Memory signals can be enabled/disabled.

Configuring User Control Register Setup

The User Control Register Setup tab displays a table-like panel where you can enable the Control Register. The Control Register address ranges from 0x81000000 up to 0x810000f8.

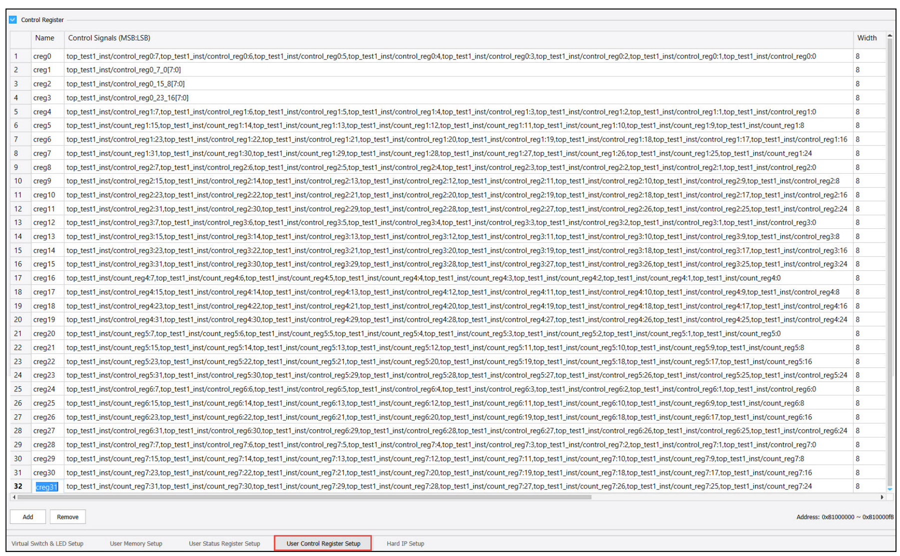
You can add up to 32-address locations with a maximum data-width of 8-bit. If the signal has a data width greater than 8, it splits into 8-bit per address location with a unique label.

The user design must include the control registers as wires, which should not have any other drivers. They should be treated as asynchronous switches as they are changed with the JTAG clock.

These signals are read-writable from the controller.

To set up User Control Register:

1. Click the **User Control Register Setup** tab.



2. Click the **Add** button. By default, a unique label is assigned to the Name column. You can edit this label by double-clicking it.

3. Drag-and-drop the corresponding signals from Design View to the **Control Signals** column.

   Or

4. Select the existing row and double-click the Control Signals column.

   The **Edit Signals** dialog box appears.

5. In the Edit Signals dialog box, do the following:

   a. Choose signals from the **Select Signals** pane.

b. Click the forward > button to move the signals to the right-hand pane.

c. Change the order of the signals as desired.

d. Click **OK**.

6. If you wish to delete a row from the Control Signals column, click the **Remove** button to delete the selected row.

Configuring User Status Register Setup

The User Status Register Setup tab displays a table-like panel where you can enable the Status Register. The Status Register address ranges from 0x81001000 up to 0x810010f8.

You can add up to 32-address locations with a maximum data-width of 8-bit. If the signal has a data width greater than 8, it splits into 8-bit per address location with a unique label.

The user design must include status registers as wires, which should be driven by registers to show the status of the different functional blocks of the design. These signals are read-only.

To set up User Status Register:

1. Click the **User Status Register Setup** tab.

2. Click the **Add** button. By default, a unique label is assigned to the Name column. You can edit this label by double-clicking it.

3. Drag-and-drop the corresponding signals from Design View to the **Status Signals** column.

   Or

4. Select the existing row and double-click the Status Signals column.

   The **Edit Signals** dialog box appears.

5. In the Edit Signals dialog box, do the following:

   a. Choose signals from the **Select Signals** pane.

   b. Click the forward > button to move the signals to the right-hand pane.

   c. Change the order of the signals as desired.

   d. Click **OK**.

6. If you wish to delete a row from the Status Signals column, click the **Remove** button to delete the selected row.

Reveal Analyzer

After you generate the bitstream, you can use Reveal Analyzer to debug your FPGA circuitry. Reveal Analyzer gives you access to internal nodes inside the device so that you can observe their behavior. It enables you to set and change various values and combinations of trigger signals. After the specified trigger condition is reached, the data values of the trace signals are saved in the trace buffer. After the data is captured, it is transferred from the FPGA through the JTAG ports to the PC.

For more information about Reveal Analyzer, refer to the **Reveal User Guide for Radiant Software**. Also, refer to **User Guides > Testing and Debugging On-Chip** in the Radiant Software Help.

User Memory Analysis

The second tab is for User Memory Analysis. The title indicates the range of the User Memory map. Analysis follows no particular steps or order.

- Default Data: A value that will initialize all memory locations.
- Write Address/Data: Address and data in hex to be written.
- Read Address/Data: Enter address. The data is read back when ‘Read’ is selected.
- Memory File: You can dump from/to a range of memory addresses to a .mem file. A user can also **Load MemFile** to load a pre-configured memory file.

Running User Control Register

1. In the **User Control Register** tab, select a label from the **Name** column, then click the **Rd** button.

   User Control Register reads the value of the corresponding address location and executes the following TCL command:

```
rva_run_controller -read_control "label"
```

2. Select a label from the Name column.

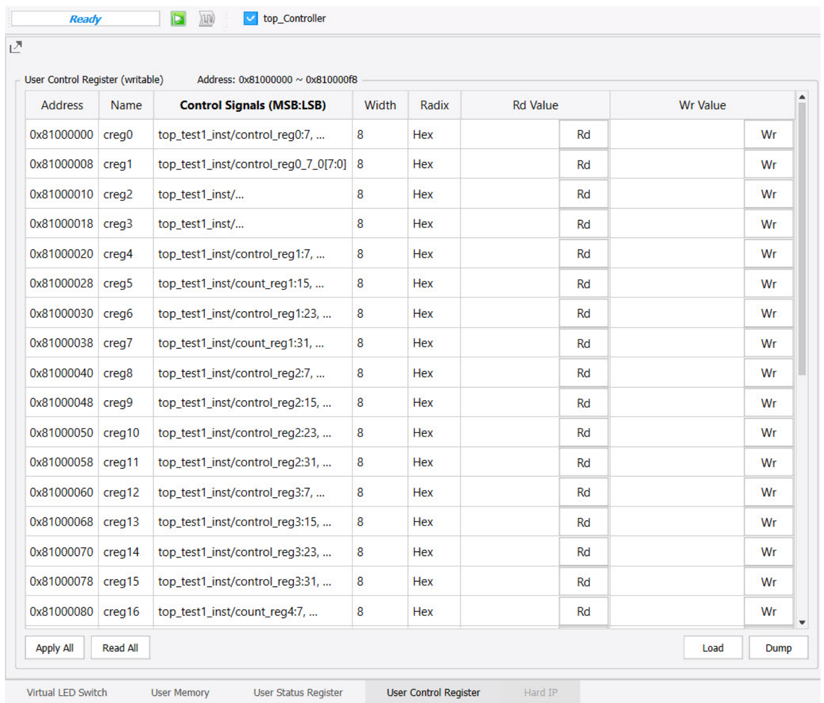
3. In the **WR Value** column, do the following:

   a. Enter desired value.

   b. Click the **Wr** button.

      User Control Register executes the following TCL command:

      ```
      rva_run_controller -write_control "label" -data value
      ```



4. Double-click a row in the **Control Signals** column to display the list of signals with their corresponding value.

5. In the User Control Register tab, click the **Apply All** button to apply the value of all address locations assigned by the Name column and execute multiple TCL commands sequentially.

6. Click the **Read All** button to read the value of all address locations assigned by the Name column and execute multiple TCL commands sequentially.

7. Click the **Load** button to load the value from the register file to all address locations of Control Register.

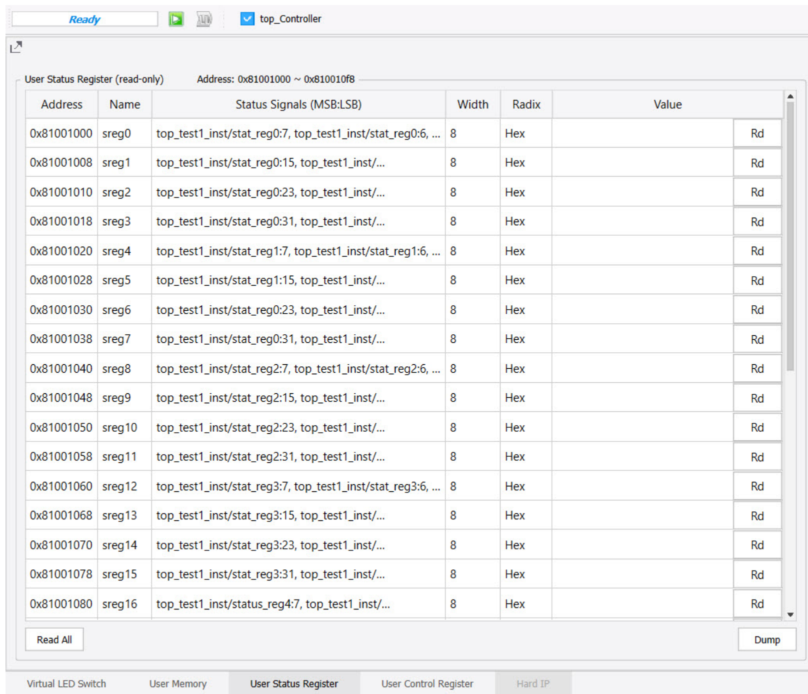
8. Click the **Dump** button to save the value of all address locations to a register file.

Running User Status Register

1. In the **User Status Register** tab, select a label from the **Name** column, then click the **Rd** button.

   User Status Register reads the value of the corresponding address location and executes the following TCL command:

   `rva_run_controller -read_status "`*`label`*`"`



2. On the bottom left corner, click the **Read All** button.

   The Read All button reads the value of all address locations assigned by the Name column and executes multiple TCL commands sequentially.

3. On the bottom right corner, click the **Dump** button.

   The Dump button saves the value of all address locations to a register file.

4. Double-click a row in the **Status Signals** column to display the list of signals with their corresponding value.

Running Reveal Eye-Opening Monitor

The Reveal Eye-Opening Monitor can be launched from the Reveal Analyzer/Controller. You need to be connected to a board to be able to run the Eye-Opening Monitor.

To launch the Eye-Opening Monitor:

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

In the drop-down menu, choose **top_Controller** to display the **Hard IP** tab.

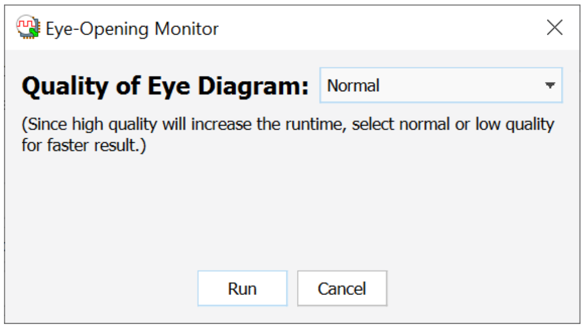
2. Click the **Hard IP** tab.

   If debugging a non-Avant device, scroll down to the PCS channels to locate the **Eye-Opening Monitor** button.

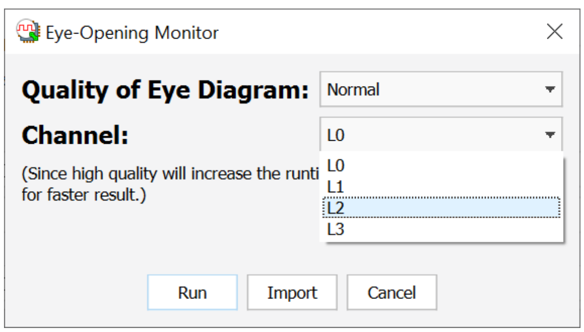
   If debugging an Avant device, the **Eye-Opening Monitor** button is located at the upper-right of the current IP instance.

3. Click the **Eye-Opening Monitor** button. The Eye-Opening Monitor dialog box is displayed.

   The figure below shows the Eye-Opening Monitor dialog box when debugging a non-Avant device.



   The figure below shows the Eye-Opening Monitor dialog box when debugging an Avant device. The Channel and Import options are added.



4. In the Eye-Opening Monitor dialog box, select the quality of the eye diagram from the drop-down menu.

   It is recommended to select normal or low quality for a faster result, and select high quality for the final result. The default is normal quality.

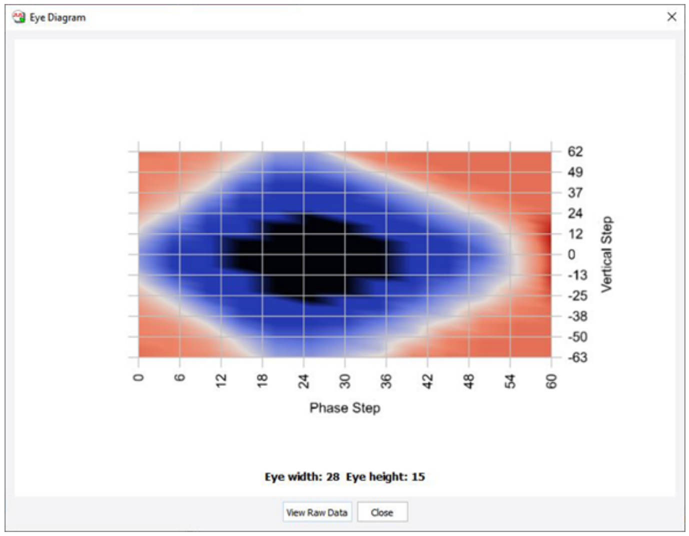
   If you are debugging Avant SerDes multiple channels, select the channel to debug in the Channel drop-down menu.

   By using this function, you can measure the relative performance of a SerDes through comparison, analyze data with regards to the height and width of the eye diagram, and check where the passing eye begins and ends.

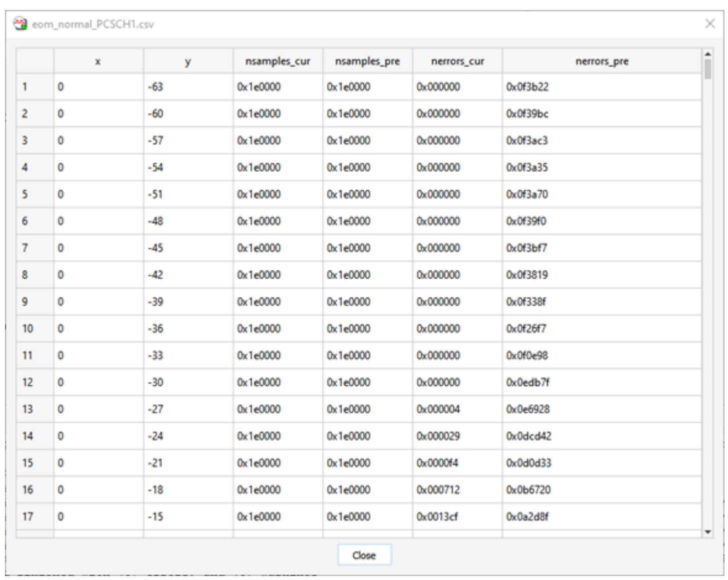
5. Click the **Run** button to start the EOM process. You can stop the process at any time by clicking the **Stop** button.

6. After the EOM has finished running, it will display the eye diagram for the current channel. The figure below shows the Eye Diagram when debugging non- Avant devices.

Figure 72: Example of Eye Diagram for Non-Avant Devices



At the bottom of the eye diagram for non-Avant devices, you can click the **View Raw Data** button to open the raw data file containing the number of samples and errors used in calculating the density of the eye diagram.



eom_normal_PCSCH1.csv

	x	y	nsamples_cur	nsamples_pre	nerrors_cur	nerrors_pre
1	0	-63	0x1e0000	0x1e0000	0x000000	0x0f3b22
2	0	-60	0x1e0000	0x1e0000	0x000000	0x0f39bc
3	0	-57	0x1e0000	0x1e0000	0x000000	0x0f3ac3
4	0	-54	0x1e0000	0x1e0000	0x000000	0x0f3a35
5	0	-51	0x1e0000	0x1e0000	0x000000	0x0f3a70
6	0	-48	0x1e0000	0x1e0000	0x000000	0x0f39f0
7	0	-45	0x1e0000	0x1e0000	0x000000	0x0f3bf7
8	0	-42	0x1e0000	0x1e0000	0x000000	0x0f3819
9	0	-39	0x1e0000	0x1e0000	0x000000	0x0f338f
10	0	-36	0x1e0000	0x1e0000	0x000000	0x0f26f7
11	0	-33	0x1e0000	0x1e0000	0x000000	0x0f0e98
12	0	-30	0x1e0000	0x1e0000	0x000000	0x0edb7f
13	0	-27	0x1e0000	0x1e0000	0x000004	0x0e6928
14	0	-24	0x1e0000	0x1e0000	0x000029	0x0dcd42
15	0	-21	0x1e0000	0x1e0000	0x0000f4	0x0d0d33
16	0	-18	0x1e0000	0x1e0000	0x000712	0x0b6720
17	0	-15	0x1e0000	0x1e0000	0x0013cf	0x0a2d8f

Close

- The raw data comes from the eom register.
- The numbers from nsample_cur, nsample_pre, nerrors_cur, and nerrors_pre columns calculate the result.

- The numbers are collected by reading the eom register of the PCS IP.

If you change the Reveal Controller options, you can re-run the EOM to display the new eye diagram and view the new raw data until the desired result is achieved.

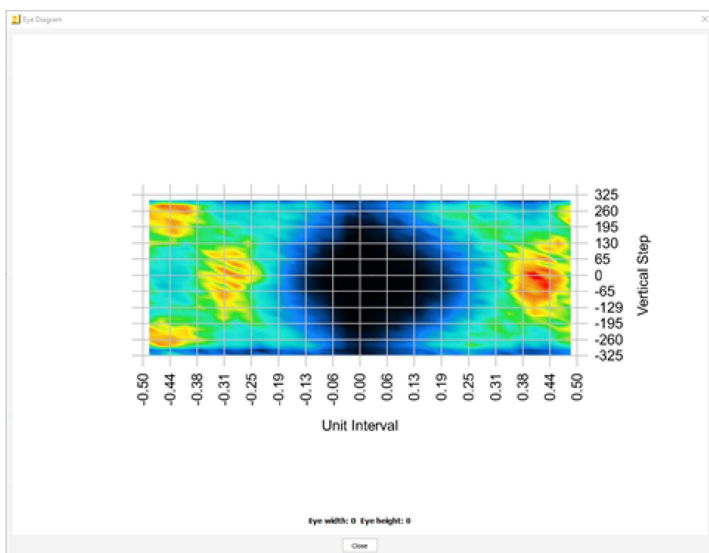
Importing an Existing EOM Result

If you are debugging Avant SerDes, each time you run the EOM, the result is automatically saved as a .txt file in the project folder. The file name provides information on the EOM settings. For example, the file name *eom_normal_MPPHYX4Q1_L2.txt* tells you that the EOM result was generated using normal quality for the MPPHYX4Q1 instance with L2 or lane 2 as the selected channel.

To import an existing EOM result:

1. Click the **Import** button in the Eye-Opening Monitor dialog box.
2. The Select Existing EOM File dialog box is displayed.
   Selecting AVANT EOM (*.txt) displays only the EOM results for debugging the Avant device.
3. Select the .txt file and click **Open.** The Eye Diagram is displayed. The figure below shows the Eye Diagram when debugging Avant devices.

Figure 73: Example of Eye Diagram for Avant Devices



Reveal Controller

Reveal Controller allows you to emulate an otherwise unavailable environment for power debug. For example, your evaluation board would only have a limited number of LEDs or switches but the virtual environment enables up to 32 bits. Register memory mapping and dumping of values is also easily manifested while visibility into Hard IP is also enabled.

For more information about Reveal Controller, refer to the **Reveal User Guide for Radiant Software**. Also, refer to **User Guides > Testing and Debugging On-Chip** in the Radiant Software Help.

Power Calculator

Power Calculator estimates the power dissipation for your design. It 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 power consumption.

To launch Power Calculator from the Radiant software, choose **Tools > Power Calculator** or click the Power Calculator button on the toolbar.

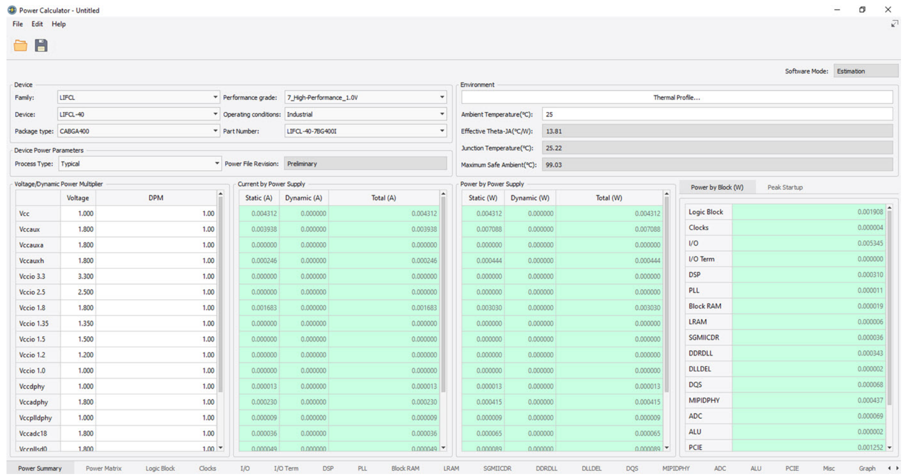
Power Calculation Modes

Power Calculator opens in estimation mode or calculation mode, depending on the status of the selected .pcf file. If it opens in calculation mode, the Bank settings will be from background power database, not from the constraint file. When you make certain design changes in calculation mode, Power Calculator reverts to estimation mode.

Power Calculator Pages

When Power Calculator opens, it displays the Power Summary page, which 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. Tabs for other pages, including Power Matrix, Logic Block, Clocks, I/O, I/O Term, Block RAM, Graph, and Report, are arranged across the bottom. The number and types of these pages depends on the target device.

Figure 74: Power Calculator



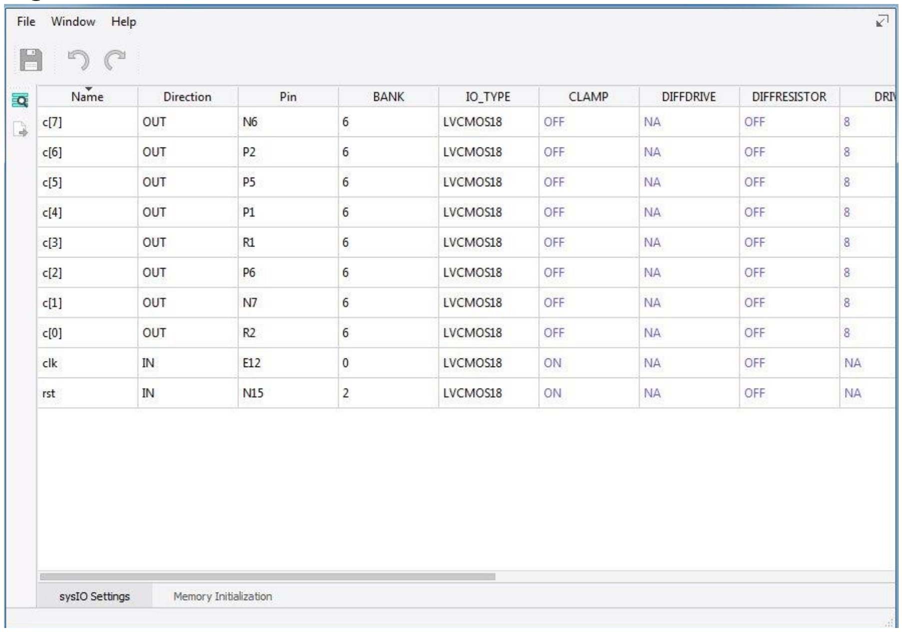
Power Calculator is also available as a non-integrated tool, which you can launch without opening the Radiant software. The non-integrated Power Calculator provides all the same functionality as the integrated version. To open the non-integrated Power Calculator from the Windows Start menu, select **Lattice Radiant Software > Accessories > Power Calculator**. 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).

For more information on Power Calculator, see **User Guides > Analyzing Power Consumption** in the Radiant Software Help.

ECO Editor

The Engineering Change Order (ECO) Editor enables you to safely make changes to an implemented design without having to rerun the entire process flow. Choose **Tools > ECO Editor** or click the ECO Editor button on the toolbar.

Figure 75: ECO Editor



Name	Direction	Pin	BANK	IO_TYPE	CLAMP	DIFFDRIVE	DIFFRESISTOR	DRI
c[7]	OUT	N6	6	LVCMOS18	OFF	NA	OFF	8
c[6]	OUT	P2	6	LVCMOS18	OFF	NA	OFF	8
c[5]	OUT	P5	6	LVCMOS18	OFF	NA	OFF	8
c[4]	OUT	P1	6	LVCMOS18	OFF	NA	OFF	8
c[3]	OUT	R1	6	LVCMOS18	OFF	NA	OFF	8
c[2]	OUT	P6	6	LVCMOS18	OFF	NA	OFF	8
c[1]	OUT	N7	6	LVCMOS18	OFF	NA	OFF	8
c[0]	OUT	R2	6	LVCMOS18	OFF	NA	OFF	8
clk	IN	E12	0	LVCMOS18	ON	NA	OFF	NA
rst	IN	N15	2	LVCMOS18	ON	NA	OFF	NA

ECOs are requests for small changes to be made to your design after it has been placed and routed. The changes are directly written into the Unified Database (.udb) file without requiring that you go through the entire design implementation process.

ECOs are usually intended to correct errors found in the hardware model during debugging. They are also used to facilitate changes that had to be made to the design specification because of problems encountered when other FPGAs or components of the PC board design were integrated.

The ECO Editor includes windows for editing I/O preferences, and memory initialization values. It also provides a Change Log window that enables you to track the changes between the modified .udb file and the post-PAR .udb file.

Note

After you edit your post-PAR UDB file, your functional simulation and timing simulation will no longer match.

For more information, see **Reference Guides > Command Line Reference Guide > Command Line Tool Usage > Running Various Utilities from the Command Line > ECO Editor** in the Radiant Software Help.

Programmer

The Radiant Programmer is a system for programming devices. The software supports serial programming of Lattice devices using PC and Linux environments. The tool also supports embedded microprocessor programming.

For more information about Programmer and related tools, refer to the **Programming Tools User Guide for Radiant Software**. Also, refer to **User Guides > Programming the FPGA** in the Radiant Software or **Stand-Alone Programmer Help**.

Run Manager

Run Manager runs the processes for the different implementation/strategy combinations. Choose **Tools > Run Manager** or click the Run Manager button on the toolbar.

Run Manager takes the design through the entire process flow for each selected implementation. If you are running on a multi-core system, Run Manager will distribute the iterations so that they are executed in parallel. The option “Maximum implementation processes running in Run Manager” is available in the General section of the Options dialog box. Choose **Tools > Options** to access it. This option enables you to set the maximum number of processes to run in parallel. Generally, the maximum number of processes should be the same as the number of cores in your processor; but if the strategy is using the “Multi-Tasking Node List” option for Place & Route Design, this number should be set to one.

You can use the Run Manager list to set an implementation as active. Right-click the implementation/strategy pair and choose **Set as Active**.

For an implementation that uses multiple iterations of place-and-route, you can select the iteration that you want to use as the active netlist for further processes. Expand the implementation list, right-click the desired iteration, and choose **Set as Active**. The active iteration is displayed in italics.

To examine the reports from each process, set an implementation as active, and then select the Reports view.

See the **User Guides > Managing Projects** section of the Radiant Software Help for more information about using implementations, strategies, and Run Manager.

Synplify Pro for Lattice

Synplify Pro for Lattice is an OEM synthesis tool used in the Radiant software design flow. Synplify Pro runs in batch mode when you run the Synthesize Design step in Process View. To examine the output report, select **Synplify Pro** in the Process Reports folder of Reports View.

You can also run Synplify Pro in interactive mode. Choose **Tools > Synplify Pro for Lattice** or click the Synplify Pro button on the toolbar.

For more information, see the **Synplify Pro User Guide**, which is available from the Radiant Start Page or the Synplify Pro Help menu.

Mentor ModelSim

The Mentor® ModelSim® tool is an OEM simulation tool that is closely linked to the Radiant software environment. It is not run as part of the Process implementation flow. To run ModelSim, choose **Tools > ModelSim Lattice-Edition** or click the ModelSim button on the toolbar.

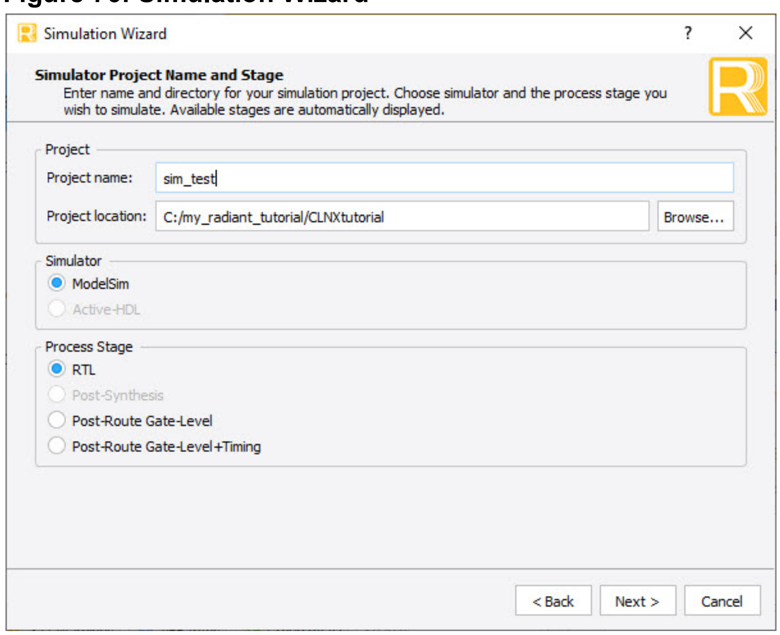
See "Simulation Flow" on page 113 for more information about simulating your design. See "Simulation Wizard Flow" on page 114 for information about creating a simulation project to run in ModelSim.

For complete information about ModelSim, see ModelSim's online documentation, which is available from the ModelSim Help menu.

Simulation Wizard

The Simulation Wizard enables you to create a simulation project for your design. To open Simulation Wizard, choose **Tools > Simulation Wizard** or click on the icon in the Radiant software toolbar. The wizard leads you through a series of steps that include selecting a simulation project name, location, specifying the simulator type, selecting the process stage to use, and selecting the source files. To learn more about the flow, view "Simulation Wizard Flow" on page 114.

Figure 76: Simulation Wizard



Source Template

Source Template provides templates for creating VHDL, Verilog, and constraint files. Templates increase the speed and accuracy of design entry. You can drag and drop a template directly to the source file. You can also create your own templates. For more information, see "Source Template" on page 33.

IP Catalog

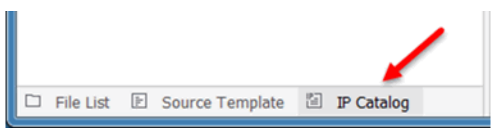
IP Catalog is an easy way to use a collection of functional blocks from Lattice Semiconductor. There are two types of functional blocks available through IP Catalog: modules and IP. IP Catalog enables you to extensively customize these blocks. They can be created as part of a specific project or as a 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.
- **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 in a separate step before they can be accessed from IP Catalog.

Overview of the IP Catalog Process

Below are the basic steps of using IP Catalog modules and IP.

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 may 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 then 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 after generation as you would with any other source file (such as a Verilog or VHDL file).
4. Instantiate the module/IP into the project's design. An HDL instance template is created during generation 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.

For more information on IP Catalog, see **User Guides > Entering the Design > Designing with Soft IP, Modules, and PMI** in the Radiant Software Help.

IP Packager

IP Packager is a tool for you to create and IP package easily. You can edit ports, files, parameters, and memory in the IP Packager, and pack a customized IP directly.

See Also

- "Running IP Packager and Viewing Documentation" on page 165
- "Installing IP Created with IP Packager into IP Catalog" on page 165

Running IP Packager and Viewing Documentation

IP Packager is a stand-alone tool. IP Packager can be run from both Windows and Linux. IP Packager documentation is contained within the IP Packager tool.

To run IP Packager:

- In Windows, go to the Windows Start menu and choose **Programs > Lattice Radiant Software > Accessories > IP Packager**.
- In Linux: from the ./<Radiant Software Install Path>/bin/lin64 directory, enter the following on a command line:

```
./ippack.sh
```

The IP Packager tool opens.

To view IP Packager documentation:

- In the IP Packager main window, choose **Help > Help**.

The IP Packager can also be run from the command line. Refer to "IP Packager" on page 259.

See Also

- "IP Packager" on page 165
- "Installing IP Created with IP Packager into IP Catalog" on page 165

Installing IP Created with IP Packager into IP Catalog

You can download and add IP created with IP Packager into IP Catalog.

To download and add a User IP:

1. In the Radiant IP Catalog, click on the **Install a User IP** button.
2. In the **Select user IP package file to install** dialog box, browse to the 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.

See Also

- "IP Packager" on page 165
- "Running IP Packager and Viewing Documentation" on page 165

SEI Editor

SEI (Soft Error Injection) Editor allows you to generate single-bit errors, insert them into a bitstream, and detect them for analysis, simulating the effect of radiation damage on the device's configuration memory.

SEI Editor is available after you have placed and routed your design and generated a bitstream.

Common Tasks

The Radiant software gathers the many FPGA implementation tools into one central design environment. This gives you common controls for active tools, and it provides shared data between views.

Controlling Tool Views

Tool views are highly configurable in the Radiant software environment. You can detach a tool view to work with it as a separate window, or create tab groups to display two views side-by-side.

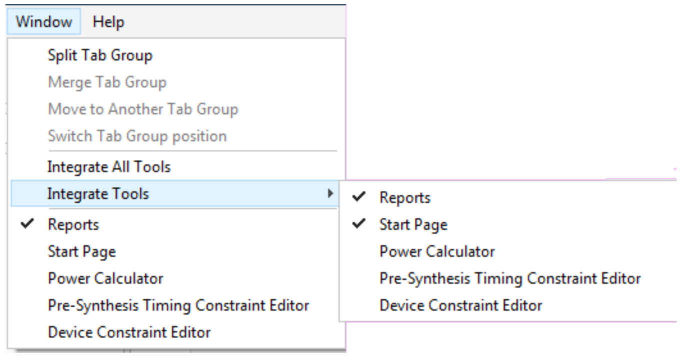
Detaching and Attaching a Tool View

Each integrated tool view contains a Detach button in the upper-right corner that allows you to work with the tool view as a separate window.

After a tool view is detached, the Detach button changes to an Attach button, which reintegrates the view into the Radiant main window.

You can detach as many tool views as desired. The Window menu keeps track of all open tool views and allows you to reintegrate one or all of them with the main window or detach any of them. Those that are already integrated are displayed with a check mark.

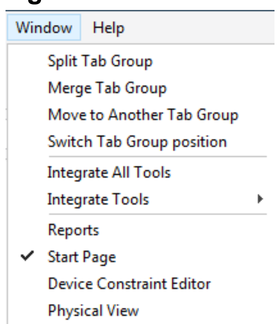
Figure 77: Window Integrate Tools Menu



Tab Grouping

The Radiant software allows you to split one or more active tools into a separate tab group. Use the Window menu or the toolbar buttons to create the tab group and control the display.

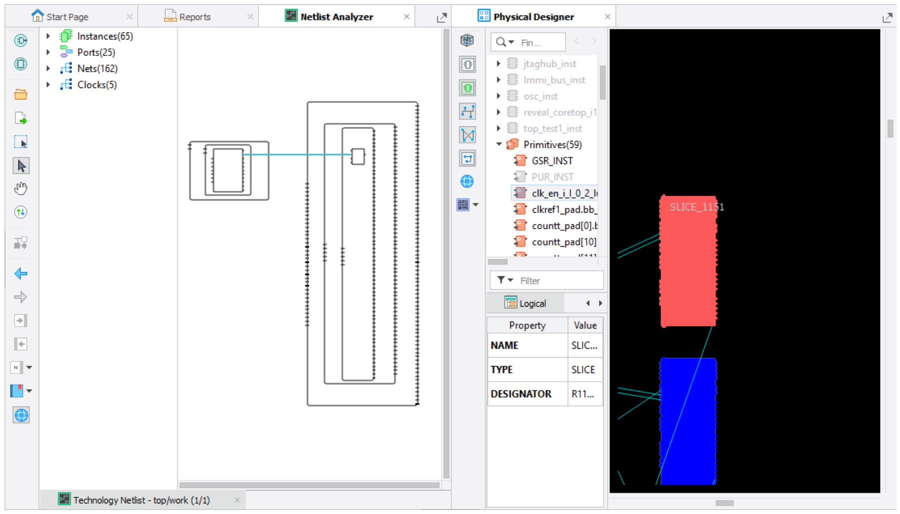
Figure 78: Tab Controls in View Menu



The Split Tab Group command separates the currently active tool into a separate tab group. Having two separate tab groups enables you to work with two tool views side-by-side. This is especially useful for dragging and dropping to make constraint assignments; for example, dragging a port from Netlist View to Package View in order to assign it to a pin.

Having two separate tab groups is also useful for examining the same data element in two different views, such as the Netlist Analyzer and Physical Designer layouts.

Figure 79: Split Tab Group with Side-by-Side Layout Views



Move an active tool view from one tab group to another by dragging and dropping it, or you can use the Move to Another Tab Group command.

To switch the positions of the two tab groups, click the Switch Tab Group Position command.

To merge the split tab group back into the main group, click the Merge Tab Group command.

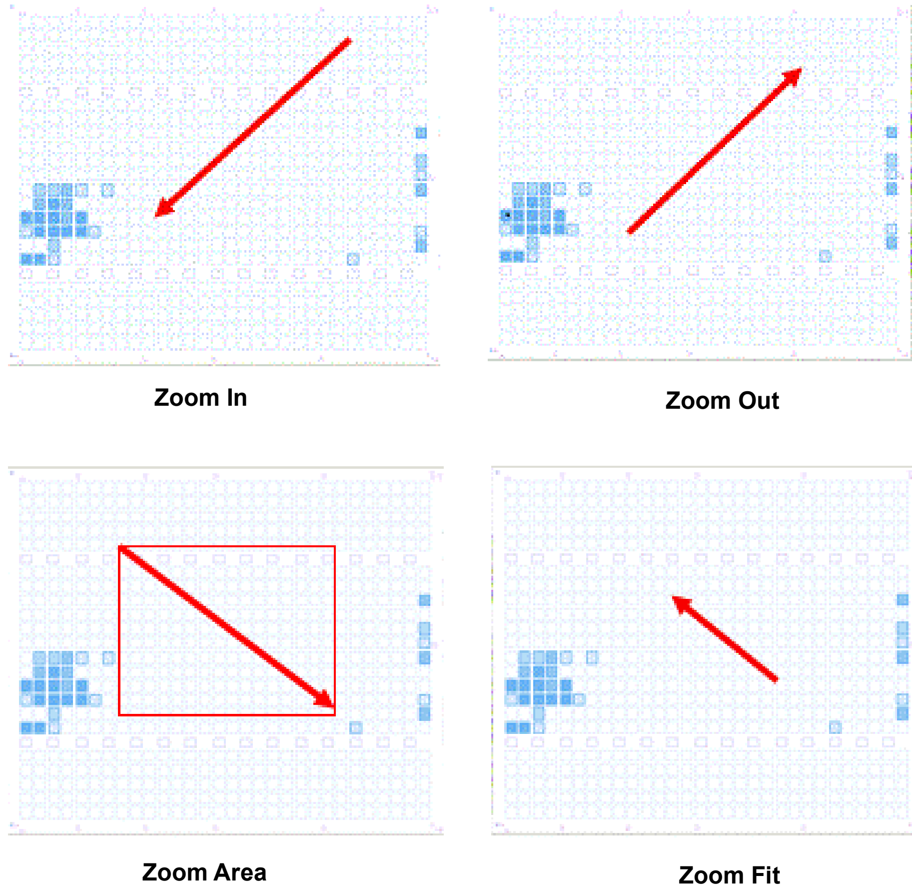
Using Zoom Controls

The Radiant software includes display zoom controls in the View toolbar. There are controls for increasing or reducing the scale of the view, fitting the display contents to the window view area, and fitting a selected area or object to the window view area.

Use the mouse to quickly zoom in, out or pan graphical views, such as Placement Mode and Routing Mode, by doing the following, as shown in Figure 80:

- Zoom In: press and hold the left mouse button while dragging the mouse down from upper right to left to zoom in.
- Zoom Out: press and hold the left mouse button while dragging the mouse up from lower left to right to zoom out.
- Zoom To: press and hold the left mouse button while dragging the mouse down from upper left to right to zoom into the box created.
- Zoom Fit: press and hold the left mouse button while dragging the mouse up from lower right to left to reset the diagram so it fits on screen.
- Pan: Click **Pan** ( ) and drag the mouse in any direction to move the diagram, or press and hold **Ctrl** and drag the mouse.

Figure 80: Zoom with Mouse



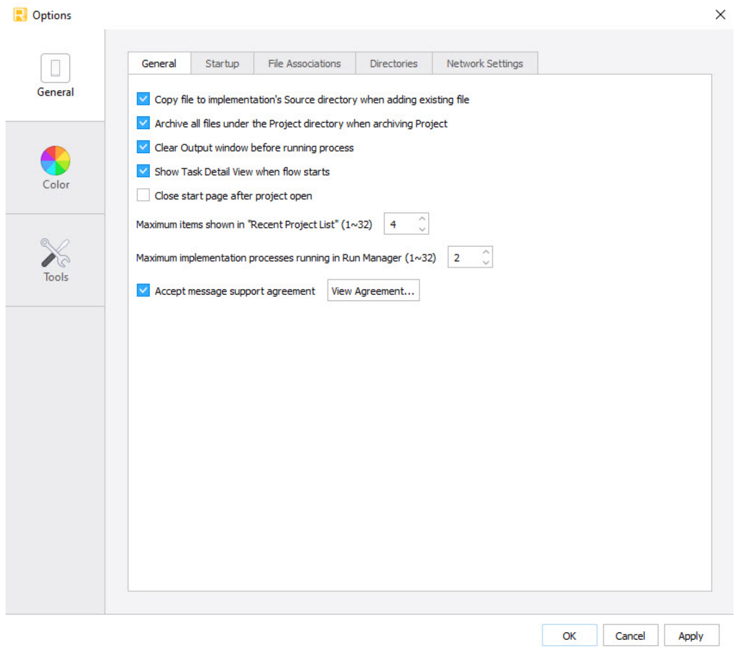
Displaying Tool Tips

When you place the cursor over a graphical element in a tool view, a tool tip appears with information on the element. The same information displayed in the tool tip will also be temporarily shown in the status bar on the lower left of the main window.

Setting Display Options

The Options dialog box, which is available from the Tools menu, enables you to specify general environment options as well as customize the display for the different tools. Tool options include selections for color, font, and other graphic elements.

Tool Options



For more information about Options, refer to "Environment and Tool Options" on page 461.

Chapter 8

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.

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 that often precludes the need for some syntax, making commands less complex. See Command Line General Guidelines and Command Line Syntax Conventions.

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 lists all of the command line tools used for various design functions and their graphical user interface counterparts. The table also provides the functional description of each command line tool.

Table 5: The Radiant Software Core Design and Tool Chart

Design Function	Command Line Tool	Radiant Process	Description
Core Implementation and Auxiliary Tools			
Encryption	Encryption	Encrypt Verilog/VHDL files	Encrypts the input HDL source file with provided encryption key.
Synthesis	SYNTHESIS	Synthesize Design	Runs input source files through synthesis based on Lattice Synthesis Engine options set in Strategies.
Synthesis	Synpwrap	Synthesize Design	Used to manage Synopsys 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	Post-Synthesis Timing Report, 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.

Table 5: The Radiant Software Core Design and Tool Chart

Design Function	Command Line Tool	Radiant Process	Description
Bitstream Generation	BITGEN	Bitstream	Converts a fully routed physical design into configuration bitstream data.
Device Programming	PGRCMD	Device Programming	Downloads data files to an FPGA device.
IP Packager	IP Packager	IP Packager	IP author select files from disks and pack them into one IPK file.

See Also

- Command Line Data Flow
- Command Line General Guidelines
- Command Line Syntax Conventions
- Invoking Core Tool Command Line Programs

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
- Command Line General Guidelines
- Command Line Syntax Conventions
- Setting Up the Environment to Run Command Line
- Invoking Core Tool Command Line Programs
- Invoking Core Tool Command Line Tool Help

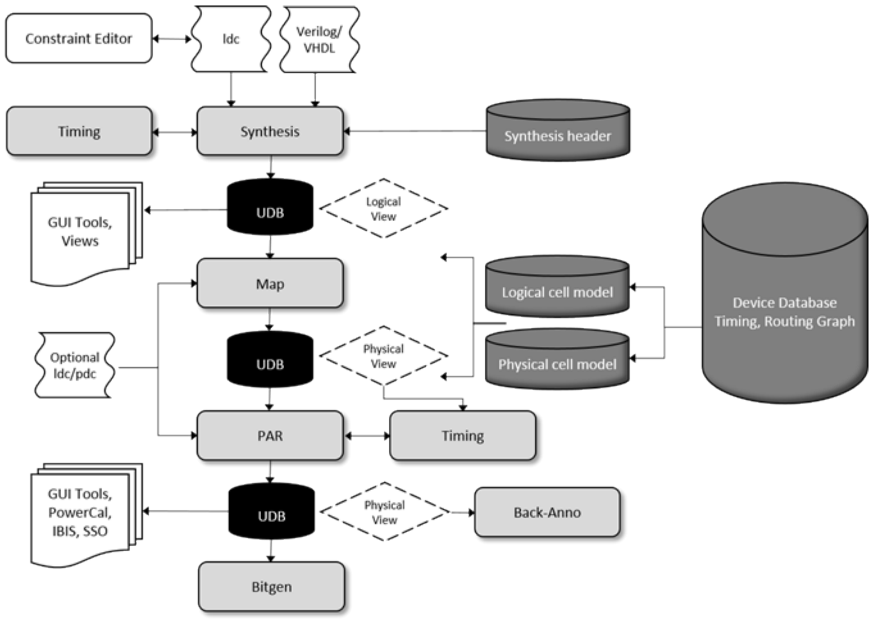
Command Line Data Flow

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
- Command Line General Guidelines



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 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.

See Also

- Command Line Reference Guide
- Invoking Core Tool Command Line Tool Help
- Command Line Syntax Conventions
- Using Command Files
- Command Line Data Flow

Command Line Syntax Conventions

The following conventions are used when commands are described:

Table 6: 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 Linux command, or a DOS command (for example, "The **playback** command is used to execute the macro you created.").

Table 6: Command Line Syntax Conventions

Convention	Meaning
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 Linux 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
- Command Line General Guidelines
- Invoking Core Tool Command Line Tool Help
- Using Command Files

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 (radiant.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 (radiantc) 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, you need to run "bash" to switch to BASH" first, then 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
```

Note:

For Programmer use `set PATH= C:/lscc/radiant/2024.2/programmer/bin/nt64`.

See Also

- Invoking Core Tool Command Line Programs
- Invoking Core Tool Command Line Tool Help

Invoking Core Tool Command Line Programs

This topic provides general guidance for running the Radiant software FPGA flow core tools. Refer to Command Line Program Overview 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 Linux™ or DOS™ command line. You can also run command line scripts or command files.

See Table 6 on page 175 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
- Command Line Syntax Conventions
- Invoking Core Tool Command Line Tool Help
- Setting Up the Environment to Run Command Line

- Using Command Files

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., iCE40 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
- Command Line Data Flow
- Command Line General Guidelines
- Command Line Syntax Conventions
- Setting Up the Environment to Run Command Line
- Using Command Files

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_libs.tcl from the Command Line
- Running HDL Encryption from the Command Line
- Running Synthesis from the Command Line
- Running Postsyn from the Command Line

- Running MAP from the Command Line
- Running PAR from the Command Line
- Running Timing from the Command Line
- Running Backannotation from the Command Line
- Running Bit Generation from the Command Line
- Running Various Utilities from the Command Line
- Using Command Files
- Using Command Line Shell Scripts

Running cmpl_libs.tcl from the Command Line

The cmpl_libs.tcl command allows you to perform simulation library compilation 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 do not have Tcl installed, or you have an older version, perform the following:

- Add <*Radiant_install_path*>/tcltk/windows/BIN to the front of your PATH, and
- For Linux users only, add <*Radiant_install_path*>/tcltk/linux/bin 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 Siemens Questa is already in your PATH and preceding any 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.
- To execute this script error free, **Siemens Questa 2020.4** or a later version should be used for compilation.

Check log files under <*target_path*> (defualt = .) for any errors, as follows:

- For Windows, type:

  ```
  find /i "error" *.log
  ```

- For Linux, type:

  ```
  grep -i error *.log
  ```

Subjects included in this topic:

- Running compl_lib.tcl
- Command Line Syntax
- cmpl_libs.tcl Options
- Example

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>|cadence}] [-device {ice40up|lifcl|lfcpnx|lfd2nx|lfmxo5-25|lfmxo5-15d|lfmxo5-100t|lfmxo5-55t|lavat|ut24c|ut24cp|all<default>}] [-target_path <target_path>]

cmpl_libs.tcl Options

The table below contains all valid options for cmpl_libs.tcl.

Table 7: cmpl_libs.tcl Command Line Options

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>-sim_path <sim_path></td><td>The -sim_path argument specifies the path to the simulation tool executable (binary) folder. This option is mandatory. Currently, only QuestaSim simulators from Mentor and Xcelium simulator from Cadence are supported. NOTE: If <<i>sim_path</i>> has spaces, then it must be surrounded by " ". Do not use { }.</td></tr>
<tr><td>[-sim_vendor {mentor<default>|cadence }]</td><td>The -sim_vendor argument specifies simulation vendor. This argument is optional. If you do not use this argument, Mentor is chosen by default, and the libraries are compiled for QuestaSim. If you are using QuestaSim choose mentor, and Xcelium choose cadence for -sim_vendor.</td></tr>
<tr><td>[-device {ice40up|lifcl|lfcpnx|lfd2nx|lfmxo5-25|lfmxo5-15d|lfmxo5-100t|lfmxo5-55t|lavat|ut24c|ut24cp|all<default>}]</td><td>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.</td></tr>
<tr><td>[-target_path <target_path>]</td><td>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.
Note:
1. This argument is recommended if the current folder is the Radiant software's startup (binary) folder, or if the current folder is write-protected.
2. If <<i>target_path</i>> has spaces, then it must be surrounded by " ". Do not use { }.</td></tr>
</table>

Example

This section illustrates and describes an example of a Simulation Libraries Compilation Tcl command.

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 QuestaSim is specified by -sim_path.

```
tclsh <c:/lscc/radiant/1.0/>/cae_library/simulation/scripts/
cmpl_libs.tcl -sim_path C:/questasim64_10.4e/win64 -target_path
c:/mti_libs
```

See Also

- Command Line Program Overview

Running HDL Encryption from the Command Line

Radiant software allows you to encrypt the individual HDL source files.

The tool supports encryption of Verilog HDL and VHDL files. Per command's execution, single source file is encrypted.

The HDL file can be partially or fully encrypted depending on pragmas' placements within the HDL file. To learn more about pragmas' placements, see Defining Pragmas.

Subjects included in this topic:

- Running HDL Encryption
- Command Line Syntax
- Encryption Options
- Examples

Running HDL Encryption

Before running the utility, you need to annotate the HDL file with the appropriate pragmas. Additionally, you may need to create a key file containing an encryption key. To view the key file's proper formating, see Key File.

Command Line Syntax

```
encrypt_hdl [-k <keyfile>] [-l language]
[-o <output_file>] [-comment <comment>] <input_HDL_file>
```

Encryption Options

The table below contains descriptions of all valid options for HDL encryption.

Table 8: Encryption Command Line Options

Option	Description
-h(elp)	Print command help message.
-k <keyfile>	A key repository file. Depending on the location of the key, this option is required or optional. ▶ If the HDL source file contains no pragma, the key file is required. The tool encrypts the entire HDL file using all key sets declared in the key file. ▶ If the HDL source file contains only `begin` and `end` pragmas and no key pragmas, the key file is required. The tool encrypts the section between `begin` and `end` using all key sets declared in the key file. ▶ If the HDL source file contains the proper key pragma, key_keyowner, key_keyname, but the key file is missing the provided key_public_key, the tool fetches the first public key string matching the key_keyowner and key_keyname requirement in the key file If the HDL source file contains the proper definition of key, this option is not required. NOTE: If the same key name is defined in both, HDL source file and key.txt file, the key defined in HDL source file has a precedence.
-l <language>	Directive language, vhdl or verilog (default).
-o <output_file>	An encrypted HDL file. This is an optional field. If not defined during the encryption, the tool generates a new output file <input_file_name>_enc.v.
<input>	Input Verilog or VHDL file.
-comment <comment>	Additional comment.

Examples

This section illustrates and describes a few examples of HDL encryption using Tcl commands.

Example 1

This example shows a successful encryption of HDL file with default options. It is assumed that key is properly defined in HDL file. Since output file name was not specified, the tool generates an output file <file_name>_enc.v in the same directory as the location of the input file.

```
> encrypt_hdl -k source/impl_1/keys.txt -o top.v top.v
Options:
  Key repository file:     source/impl_1/keys.txt
  Directive language:     <not specified>, use verilog as default
  Output file:  top.v
Processed 2 envelopes.
```

Example 2

This example shows a successful encryption of HDL file by generating a new output file.

```
> encrypt_hdl -k source/impl_1/keys.txt -o remote_files/top_v1_en.v remote_files/sources/top_v1_part.
Options:
  Key repository file:     source/impl_1/keys.txt
  Directive language:     <not specified>, use verilog as default
  Output file:  remote_files/top_v1_en.v
Processed 2 envelopes.
```

Example 3

This example shows unsuccessful HDL encryption due to a missing key file. To correct this issue, the user must either define the appropriate key file key.txt or annotate the HDL file with appropriate pragmas. To correct the issue, define the key either in key.txt file or directly in HDL source file.

```
> encrypt_hdl -o remote_files/sources/top_v1_part_en.v remote_files/sources/top_v1_part.v
Options:
  Key repository file:     <not specified>
  Directive language:     <not specified>, use verilog as default
  Output file:  remote_files/sources/top_v1_part_en.v
ERROR - remote_files/sources/top_v1_part.v at line 88: missing key.
```

Note

A key is always required in the encryption tool while key file is optional. If the complete key: `key_keyowner`, `key_keyname`, `key_method`, and `key_public_key`, is defined within HDL source file, key file is not required.

For specific steps and information on how to encrypt HDL files in the Radiant software, refer to the following section in the Radiant Software Help: **User Guides > Securing the Design**.

Defining Pragmas

Pragmas are used to specify the portion of the HDL source file that must be encrypted. Pragma' definition is compliant with IEEE 1735-2014 V1 standard.

Pragma syntax in Verilog HDL file:

```
'pragma protect <pragma's option>
```

Pragma syntax in VHDL file:

```
'protect <pragma's option>
```

Table 9: List of Available Pragma Options

Name	Available Values	Description
version	1 (default)	Specifies the current Radiant software encryption version.
author	string	Specifies the file creator.
author_info	string	Additional information you would like to include in file.
encoding	base64	The output format of processed data.
begin		The start point for data obfuscation.

Table 9: List of Available Pragma Options

Name	Available Values	Description
end		The end point for data obfuscation.
key_keyowner	string	The key creator.
key_keyname	string	The RSA key name to specify the private key.
key_method	rsa	The cryptographic algorithm used for key obfuscation.
key_public_key		The RSA public key file name.
data_method	aes128-cbc aes256-cbc (default)	The AES encryption data method.

To encrypt HDL source file, encryption version, encoding type, and key specific pragmas must be defined in the HDL source file by HDL designer; only the content within the pragmas is encrypted.

Note

Multiple key sets can be declared in a single key file.

Example of Verilog source file marked with Pragmas:

```
// 3 bit counter with asynchronous reset
module count(c,clk,rst);

input clk,rst;
output [2:0]c;
reg [2:0]c;

`pragma protect version=1
`pragma protect author="<Your Name>"
`pragma protect author_info="<Your info>"
`pragma protect key_keyowner="Lattice Semiconductor"
`pragma protect key_keyname="LSCC_RADIANT_2"
`pragma protect key_method="rsa"
`pragma protect key_public_key
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0EZKUUhbuB6vSsc7OhQJ
iNAWJR5unW/OWp/LFI71eAl3s9bOYE2OlKdxbai+ndIeo8xFt2btxetUzuR6Srvh
xR2Sj9BbW1QToo2u8JfzD3X7AmRvlwKRX8708DPo4LDHZMA3qh0kfDDWkp2Eausf
LzE2cVxgq7fy/bDhUeN8xKQCSKJ7aguG6kOI6ROoZz21ljzDLUQzhm2qYF8SpU1o
tD8/uw53wLfSuhR3MBOB++xcn2imvSLqdgHWuhX6CtZIx5CD4y8inCbcLy/0Qrf6
sdTN5SAg2OZhjeNdzmqSWqhL2JTDw+Ou2fWzhEd0i/HN0y4NMr6h9fNn8nqxRyE7
IwIDAQAB

//put a blank line above
`pragma protect data_method="aes256-cbc"

`pragma protect begin
always @(posedge clk or posedge rst)
     begin
        if (rst)
           c = 3'b000;
          else
           c = c + 1;
      end

`pragma protect end

endmodule
```

The encrypted file may contain multiple encrypted key sets.

Example of encrypted Verilog file:

```
// 3 bit counter with asynchronous reset
module count(c,clk,rst);

input clk,rst;
output [2:0]c;
reg [2:0]c;

//put a blank line above

`pragma protect begin_protected
`pragma protect version=1
`pragma protect author="<Your Name>"
`pragma protect author_info="<Your info>"
`pragma protect encrypt_agent="Radiant encrypt_hdl"
`pragma protect encrypt_agent_info="Radiant encrypt_hdl Version 1.0"

`pragma protect encoding=(enctype="base64", line_length=64, bytes=256)
`pragma protect key_keyowner="Lattice Semiconductor"
`pragma protect key_keyname="LSCC_RADIANT_2"
`pragma protect key_method="rsa"
`pragma protect key_block
gOatWm+1rVPboQIqaGf2gvNdUH/vTXzyRA4C+tdNxpqNWeeTXrTF+uIa21e9Io7S
K6ce3BVAXydtADq5Wy50EjcHzUi3YmleyEdfVn2pxCp+3csuiZSeNgbtYutonZjh
8ReQYzPqKt6fZvhZ1AqHiuuZhFUsZQFXqnT8IW4dQ7HWudREzx6jB7h+7vT+wJvH
N5kZMiHBFGRhiTePZ+yDOQwFVvwITezEoS099I8MoRGWl1U9kb4/Kenk96MIqE3W
lKAiQivIjXeWvLRqmOb0hNGRoOEYwjylY1pjq9GyelHoDC6czdyqOOWrunt//XNu
v/QtpeEe/co7o9arRtHbvw==

`pragma protect data_method="aes256-cbc"
`pragma protect encoding=(enctype="base64", line_length=64, bytes=176)
`pragma protect data_block
z+c6t504d6NkyrL5x6j6/raeaQmg0v9xmOQVaj3oq45SAsUIhGaihFVt+sS2kbNJ
/KBaqdciXAJHW8uRJtE/4nB+gZbVQsVnhRUlH/beksDnhdhWhNw/fcX6v6xptGwh
wQxsY+TjzT+pGC9rOEmAo2tndK63cWjHlg8hIZYnnw7yZAzv2OqylztSWFkdR9T4
mHclplFr96xb59oCRqbpQQgeESGgX9L1Yfd8j0ZXkSM=

`pragma protect end_protected

endmodule
```

Example of VHDL source file marked by Pragmas:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity top_test is

        port(
                cout : out std_logic_vector(7 downto 0);
                reset : in  std_logic;
                clk  : in  std_logic
            );
end top_test;

architecture top_test_arch of top_test is

signal count : std_logic_vector(7 downto 0);

begin

`protect version=1

`protect begin

  P100 : PROCESS(clk, reset)
  BEGIN
     IF (reset = '1') THEN
        count <= (others => '0');
     ELSE
        IF (clk'event and clk = '1') THEN
           count <= count + 1;
           cout <= count;
        END IF;
     END IF;
  END PROCESS;

`protect end


end top_test_arch;
```

Example of encrypted VHDL file:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity top_test is

        port(
                cout : out std_logic_vector(7 downto 0);
                reset : in  std_logic;
                clk  : in  std_logic
            );
end top_test;

architecture top_test_arch of top_test is

signal count : std_logic_vector(7 downto 0);

begin

`protect begin_protected
`protect version=1
`protect author="Lattice Semiconductor Corporation"
`protect author_info="Lattice Semiconductor Corporation"
`protect encrypt_agent="Radiant encrypt_hdl"
`protect encrypt_agent_info="Radiant encrypt_hdl Version 1.0"

`protect encoding=(enctype="base64", line_length=64, bytes=256)
`protect key_keyowner="Lattice Semiconductor"
`protect key_keyname="LSCC_RADIANT_2"
`protect key_method="rsa"
`protect key_block
Vi2YLO+x6rdARfQ9Dy5nkhsQDsmm1w6mdX+BGfDMCLHH9OVHbd1SmeHawCz0jXY8
ZfRK8VF/h1zOBmoosgY1pKd/Dpc5/4xkcEtnLzRbiXr1PhvP+tAFMMYrlFsqzJF/
0yoej8yT6mayYbd5mcEr5rzBmX29HuPBZbYr1ziac5IBpHZUa0mcwwhFnj5kz4OB
bVSqiay7v8ECP41vNzoRKrsTYKBOhiTa6UoG08ut2E4d8wJIXNBgZOuShYYzuOuv
1goVgwaRtFWrpTNXEmrZJPr/ikXRHTFzORpkDM7yNwGVTNJPMJ2aQde2wOi6EZWe
1NnRF4K2HK0Oz1NRbffIjQ==

`protect data_method="aes128-cbc"
`protect encoding=(enctype="base64", line_length=64, bytes=272)
`protect data_block
B7yNcwT9w4purXSxU1ln4f3rs1psxSPlV3pnWYipDj6rQSA7wniBxcC/laFebwKE
fvbKngTIn7N+W/1Den3kjpuznIvLy5cV/GTANFP0cWt9rnRrDCM5CYtNWqaMEZu7
o2QPLifpCvwEgygI0RO6NQS5frKo/jQLgOhf68+VpqFPozfrGAYI/YkEofh0foDH
9ScyO6grmJQCqtkqX2N3p6738N3iCFdKWJ6Udo5t++AT3YYeZZ7bprDN941k/BkU
YZij8fJx+qovJUWQmSUJYNnt2Vyoac4hsevXsFRxm439ssQtP4vHHEnraBSrHdVG
DJn0GqFID+tpC67tE61U2Y/yi18psFhZGs8jmuv9yGk=

`protect end_protected

end top_test_arch;
```

See Also

- Running HDL Encryption from the Command Line
- Key File

Key File

The key repository file defines the cryptographic public key used for RSA encryption. In Radiant software, the key file contains Lattice public key. Additionally, it may contain some of the common EDA vendors public keys.

The Lattice public key file key.txt is located at <*Radiant_installed_directory*>/ ispfpga/data/ folder. Aside from the Lattice Public Key, the current version contains the public key for Synopsys and Cadence.

Note

If using Synplify Pro synthesis tool, both, the Lattice Public Key and the Synplify Pro Public Key must be defined in the key file. The Synplify Pro Public Key is used during the synthesis step to decrypt an encrypted design. The Lattice Public Key is used during the post-synthesis flow to decrypt an encrypted design.

A key file must contain properly declared pragmas such as `key_keyowner`, `key_keyname`, `key_method`, and `key_public_key` for each of the specified keys. The key value follows the `key_public_key` pragma.

The key file typically also contains the `data_method` pragma. It defines the algorithm used in data block encryption of HDL source file.

Example of a Key File:

```
// Use Verilog pragma syntax in this file
`pragma protect version=1
`pragma protect author="<Your Name>"
`pragma protect author_info="<Your info>"
`pragma protect key_keyowner="Lattice Semiconductor"
`pragma protect key_keyname="LSCC_RADIANT_2"
`pragma protect key_method="rsa"
`pragma protect key_public_key
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0EZKUUhbuB6vSsc7OhQJ
iNAWJR5unW/OWp/LFI71eAl3s9bOYE2OlKdxbai+ndIeo8xFt2btxetUzuR6Srvh
xR2Sj9BbW1QToo2u8JfzD3X7AmRvlwKRX8708DPo4LDHZMA3qh0kfDDWkp2Eausf
LzE2cVxgq7fy/bDhUeN8xKQCSKJ7aguG6kOI6ROoZz21ljzDLUQzhm2qYF8SpU1o
tD8/uw53wLfSuhR3MBOB++xcn2imvSLqdgHWuhX6CtZIx5CD4y8inCbcLy/0Qrf6
sdTN5SAg2OZhjeNdzmqSWqhL2JTDw+Ou2fWzhEd0i/HN0y4NMr6h9fNn8nqxRyE7
IwIDAQAB

// Put a blank line above
// Add additonal public keys below this line

`pragma protect data_method="aes256-cbc"

// End of File
```

See Also

- Running HDL Encryption from the Command Line
- Defining Pragmas

Running Synthesis from the Command Line

The Lattice Synthesis command tool 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>}] [**-sver** {systemverilog_file.sv}] [**-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** <no (default) | yes>] [-**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>] [**-partition**] [**-partition_reset**] [**-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>] [**-drc**] [**-drc_off** {<message_id>}] [**-drc_cfg** <rule> <value>]

Synthesis Options

The table below contains descriptions of all valid options for synthesis.

Table 10: 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).
-sp <performance_grade>	Sets the device performance speed grade. If no performance grade specified, default performance value is used.
-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 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).
-sver <systemverilog_file.sv>	Names of SystemVerilog design files (must have, if language is SystemVerilog or mixed language).
-hdl_param <param_name param_value>	Allows you to override HDL parameter pairs in the design file.
-optimization_goal <area | timing (default)>	The synthesis tool allows you to choose among the following optimization options: ▶ **balanced** balances the levels of logic. ▶ **area** optimizes the design for area by reducing the total amount of logic used for design implementation. ▶ **timing** optimizes the design for timing. The default setting depends on the device type. Smaller devices, such as ice40tp default to balanced.
-force_gsr <no(default) | yes>	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 10: Synthesis Command Line Options

Option	Description
-ramstyle <auto(default) | distributed | block_ram(EBR) | 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. 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 (EBR)** 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. ▶ **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 10: 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. Specifying a syn_romstyle attribute globally or on a module or ROM instance with a value of: ▶ **auto** allows the synthesis tool to choose the best implementation to meet the design requirements for performance, size, etc. ▶ **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.
-output_edif <filename.edf>	Specifies the name of the output EDIF netlist file.
-output_hdl <filename.v>	Specifies the name of the output Verilog netlist file.
-sdc <sdc_file.ldc>	Specifies a Lattice 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. 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. 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 10: Synthesis Command Line Options

Option	Description
-max_fanout <max_fanout (default 1000)>	Specifies maximum global fanout limit to the entire design at the top level. Default value is 1000 fanouts.
-bram_utilization <bram_utilization (default 100%)>	Specifies block RAM utilization target setting in percent of total vacant sites. Default is 100 percent.
-use_dsp <0|1(default)>	Specifies how DSP modules are implemented and allows DSP inference.
-dsp_utilization <dsp_utilization (default 100%)>	Specifies the percentage of DSP that is allowed to be used.
-mux_style <auto(default) | pfu_mux | L6Mux_single | L6Mux_multiple>	Specifies the MUX style setting, which controls the way the macrogenerator implements the multiplexer macros. Valid options are *auto, L6Mux Multiple, L6Mux Single, and PFU Mux*. The default value is auto, meaning that the tool looks for the best implementation.
-fsm_encoding_style <auto(default) | 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. 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(default)>	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(default)>	Specifies the use of I/O insertion. The 1 or true setting is the default.
-use_io_reg <0|1|auto(default)>	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. 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. The syn_useioff attribute can be set on a: ▶ top-level port ▶ register driving the top-level port ▶ lower-level port, only if the register is specified as part of the port declaration

Table 10: Synthesis Command Line Options

Option	Description
-resource_sharing <0|1(default)>	Specifies the resource sharing option. The 1 or true setting is the default.
-propagate_constants <0|1(default)>	Prevents sequential optimization such as constant propagation, inverter push-through, and FSM extraction. The 1 or true setting is the default.
-remove_duplicate_regs <0|1(default)>	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. (This option is for internal use only.)
-comp	Indicates HDL compilation only without synthesis. (This option is for internal use only.)
-syn	Allows you to synthesize without HDL compilation. Reads the .db file. (This option is for internal use only.)
-udb <udb_file.udb>	Writes UDB output.
-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.
-drc	Run LSE RTL design rule checking.
-drc_cfg <rule>	Configure a DRC rule
-drc_off <message_id>	Disable one or more DRC rules.
-partition	Enables the partition flow.
-partition_reset	Resets the partition flow (auto enables -partition).
-resolve_mixed_drivers <0 default | 1>	Connects the net to the VCC or GND driver if it is driven by them.

Examples

The following are examples of synthesis command lines and their respective descriptions:

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"

See Also

- Command Line Program Overview

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.

Subjects included in this topic:

- Command Line Syntax
- Postsyn Options

Command Line Syntax

```
postsyn [-w] [-a <architecture>] [-p <device>] [-t <package>]
[-sp <performance>] [-oc <operating_condition>] [-ldc
<ldc_file>] [-iplist <iplist_file>] [-macro <macro_list_file>]
[-gui] [-msgset <msgtypefile>] [-verbose] [-o <output.udb>] [-
keeprtl] [-top] [-cont_on_sdc_err] <input.vm>
```

Postsyn Options

Table 11: Postsyn Command Line Options

Option	Description
-h(elp)	Print command help message.
-w	Overwrite output file.
-a	Target architecture name.
-p	Target device name.
-t	Target package name.

Table 11: Postsyn Command Line Options

Option	Description
-sp	Target performance grade.
-oc	Target operating condition: commercial | industrial | automotive.
-ldc	Load LDC file.
-iplist	Load IP list file.
-macro	Load list file if any macro is used in the design.
-o	Output UDB file.
-keeprtl	Keep RTL view if it exists in UDB file.
-cont_on_sdc_err	Continue on SDC error instead of exit.
-verbose	Print more detailed messages.
-top	Indicate that the input is for the top design.
<input.vm>	Input structural Verilog file.

See Also

- Command Line Program Overview

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
- Example

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 performance 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 12: 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.
-help	Print usage.
-inferGSR	GSR inferencing if applicable.
-o <name[.udb]>	Optional output design file (.udb).
-f	Reads map command line arguments and switches from file.
-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.
-hierrpt	Print out hierarchical resource usage in .hrr file.
-cont_on_sdc_err	Ignore Constraint errors.
-u	Unclip unused instances.

Example

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
- Command Line Program Overview

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 Linux is explained in detail here because the option is not available for PCs.

Subjects included in this topic:

- Running PAR
- Command Line Syntax
- General PAR Options
- Examples
- PAR Multi-Tasking (-m) Option

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>] [-cores <number of cores>] [-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 PAR Options

Table 13: 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.
-cores	Run multiple threads on the local machine. You can specify "<number of cores>" to run the jobs. For cases when the user specifies both -cores and -m with a valid node list file, PAR should apply both settings (merge). If the user repeats the host machine in the node list file, the settings in the node list file take precedence over the setting in -cores (for backwards compatibility).
-p	Do not run placement.
-r	Do not run router.
-k	Keep existing routing in input UDB file. Note: only meaningful when used with -p.
-x	Ignore timing constraints.

Table 13: General PAR Command Line Options

Option	Description
-pack	Set the packing density parameter. Default: auto.
-sp	Change performance grade for setup optimization. Default: Keep current performance grade.
-hsp	Change performance grade for hold optimization. Default: M.
-dh	Disable hold timing correction.
-hos	Prioritize hold timing correction over setup performance.
-sort	Set the sorting method for ranking multiple iterations. <method> "c" sorts by cumulative slack, "w" sorts by worst slack. Default: c.
<infile>	Name of input UDB file.
<outfile>	Name of output UDB file.

Table 14: PAR Placement Command Line Option

Option	Description
<pdcfile>	Name of optional constraint file. 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

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

- **User Guides > Implementing the Design** in the Radiant Software Help
- Command Line Data Flow
- Command Line Program Overview

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 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.

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 15: Compulsory Timing Command Line Options

Compulsory Option	Description
-db-file arg	Design database file name.

Table 16: Optional Timing Command Line Options

Optional Option	Description
-endpoints arg (=10)	Number of end points.
-logical-db-file arg	Logical design database file name.
-logical-sdc-file arg	Alternate sdc file name.
-u arg (=10)	Number of unconstrained end points printed in the table.
-ports arg (=10)	Number of top ports printed in the table.
-help	Print the usage and exit.
-hld	Hold report only.
-sp arg (=None)	Setup speed grade.
-hsp arg (=M)	Hold speed grade.
-rpt-file arg	Timing report file name.
-o arg	Timing report file name.
-alt_report	Diamond-like report.
-report_sdc	Parsed file appears in report file.
-sdc-file arg	sdc file name.
-sethld	Both setup and hold report.
-v arg (=10)	Number of paths per constraint.
-time_through_async	Timer will time through async resets.
-iotime	Compute the input setup/hold and clock to output delays of the FPGA.
-io_allspeed	Get worst IO results for all speed grades.
-pwrprd	Output clock information for PowerCalculator.
-nperend arg (=1)	Number of paths per end point.
-html	HTML format report.

Table 16: Optional Timing Command Line Options

Optional Option	Description
-gui	Call from GUI.
-msg arg	Message log file.
-msgset arg	Message setting.
-io_allspeed	Get worst IO results for all speed grades.
-dump_cdc	Dump CDC registers.
-dump_uncovered	Dump load pins of connections that are not covered.

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 embedded in design1.udb and in additional timing constraint file (e.g. test.ldc). The report is called design1.twr.

timing -e 3 -sdc-file test.ldc design1.udb

Example 4

The following command analyzes the file design1.udb and produces a verbose report to check on hold times on any timing constraints embedded in design1.udb. 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 timing constraints embedded in design1.udb. 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
- Command Line Data Flow

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 **backanno** 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 (Verilog)
- 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 17: 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.
-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.

Table 17: Backanno Options

Option	Description
<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 peformance 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

- Command Line Data Flow

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
- iCE40UP Command Line Syntax
- All devices except iCE40UP 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.

iCE40UP Command Line Syntax

bitgen [**-d**] [**-b**] [**-a**] [**-w**] [**-noebrinitq1**] [**-noebrinitq2**] [**-noebrinitq3**] [**-noheader**] [**-simbitmap**] [**-nvcm**] [**-freq** <frequency_bit_setting>] [**-spilowpower**] [**-warmboot**] [**-nvcmsecurity**] {**-g** <setting_value>} <infile> [<outfile>]

All devices except iCE40UP Command Line Syntax

bitgen [**-d**] [**-w**] [**-m** <format>] [**-g** <opt:val>] [**-f** <*.t2b>] [-s <*.secproj>] {**-site** <seirule>} {**-site** <seitype>} <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 18: iCE40UP 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.
-**freq** <frequency_bit_setting>	Can setup different frequency: 0 = slow, 1 = medium, 2 = fast. Depending on the speed of external PROM, this options adjusts the frequency of the internal oscillator used by the iCE40UP device during configuration. This is only applicable when the iCE40UP device is used in SPI Controller Mode for configuration.
-**nvcm**	Produce NVCM file.
-**nvcmsecurity**	Set security. Ensures that the contents of the Non-Volatile Configuration Memory (NVCM) are secure and the configuration data cannot be read out of the device.
-**spilowpower**	SPI flash low power mode. Places the PROM in low-power mode after configuration. This option is applicable only when the iCE40UP device is used as SPI Controller Mode for configuration.
-warmboot	Enable warm boot. Enables the Warm Boot functionality, provided the design contains an instance of the WARMBOOT primitive.
-noheader	Do not include the bitstream header.
-noebrinitq0	Do not include EBR initialization for quadrant 0.
-noebrinitq1	Do not include EBR initialization for quadrant 1.
-noebrinitq2	Do not include EBR initialization for quadrant 2.
-noebrinitq3	Do not include EBR initialization for quadrant 3.
-g NOPULLUP:ENABLED	No IO pullup. Removes the pullup on the unused I/Os, except Bank 3 I/Os which do not have pullup.

Table 18: iCE40UP 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. A report (.bgn) file containing all of BITGEN's output is automatically created under the same directory as the output file.

Table 19: BITGEN Command Line Options (all devices except iCE40UP)

Option	Description
-d	Disable DRC.
-**w**	Overwrite an existing output file.
-**m** <format>	Create "mask" and "readback" files. Valid formats are: 0: Output files in ASCII 1: Output files in binary.
-**g** <opt:val>	Set option to value, options are (First is default): CfgMode Disable, Flowthrough, Bypass RamCfg Reset, NoReset DONEPHASE T3, T2, T1, T0 GOEPHASE T1, T3, T2 GSRPHASE T2, T3, T1 GWEPHASE T2, T3, T1 ES Yes, No.
-**f** <*.t2b>	Load additional command line arguments from a .t2b file.
-**s** <*.secproj>	Load a bitstream security settings file.
-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.

Table 19: BITGEN Command Line Options (all devices except iCE40UP)

Option	Description
<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. A report (.bgn) file containing all of BITGEN's output is automatically created under the same directory as the output file.

Examples

Example 1

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**, and specifies a Raw Bit File (.rbt) output with **-b**. Notice how these three options can be combined with the **-wdb** syntax.

```
bitgen -wdb <design.udb>
```

Example 2

There are two command line options to set the SEI Editor tool with the following examples:

- 1-bit Error Injection

```
bitgen -w  <other options>  -ebit 1 -sei unused -site
<unused site type> <des>.udb  <output file>
    Avant: -where <unused site type> = [PFU  | DSP |ANY]
    Nexus: -where <unused site type> = [PFU  | DSP | EBR
    |ANY]
```

  - Unused

```
bitgen -w -ebit 1  -sei unused -site PFU  des.udb
des_partial.bit
```

  - Random

```
bitgen -w -ebit 1 -sei random - des.udb  des_partial.bit
```

- 2-bit Error Injection

```
bitgen -w  <other options> -ebit 2 -sei unused -site
<unused site type>  <des>.udb  <output file>
    Avant: -where <unused site type> = [PFU  | DSP |ANY]
    Nexus: -where <unused site type> = [PFU  | DSP | EBR
    |ANY]
```

- Unused

```
bitgen -w -ebit 2 -sei unused -site DSP des.udb
des_partial.bit

bitgen -b -ebit 2 -sei unused -site DSP,DSP < udb design
>
```

- Random

```
bitgen -w -ebit 2 -sei  random des.udb  des_partial.bit
```

Note

The <other options> option does not support compressed or encrypted partial bitstream, and if it includes “-b”, it generates an ASCII bitstream output file.

See Also

- Command Line Program Overview
- Command Line Data Flow

Running Programmer from the Command Line

You can run Programmer from the command line. The **PGRCMD** command uses a keyword preceded by a hyphen for each command line option.

Subjects included in this topic:

- Running PGRCMD
- Command Line Syntax
- PGRCMD Options
- Example

Running PGRCMD

PGRCMD allows you to download data files to an FPGA device.

- To run PGRCMD, type **pgrcmd** on the command line with, at minimum, the **pgrcmd** command.

There are several command line options that give you control over the way PGRCMD programs devices. Please refer to the rest of the subjects in this topic for more details.

Command Line Syntax

The following describes the PGRCMD command line syntax:

pgrcmd [-**help**] [**-infile** <input_file_path>] [**-logfile** <log_file_path>] [**-cabletype** <cable>]

-cabletype

lattice [**-portaddress** < 0x0378 | 0x0278 | 0x03bc | 0x<*custom address*> >]

usb [**-portaddress** < EZUSB-0 | EZUSB-1 | ... | EZUSB-15 >]

usb2 [**-portaddress** < FTUSB-0 | FTUSB-1 | ... | FTUSB-15 >]

TCK [**0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10**]

PGRCMD Options

The following are PGRCMD options.

Help (Optional)

Option	Description
-help or **-h**	Displays the Programmer command line options.

Input File (required)

Option	Description
-infile *filename*.xcf	Specifies the chain configuration file (.xcf). If the file path includes spaces, enclose the path in quotes.

Log File (optional)

Option	Description
-logfile *logfilename*.log	Specifies the location of the Programmer log file.

Cable Type (optional)

Option	Description
-cabletype lattice	Lattice HW-DLN-3C parallel port programming cable (default).
-cabletype usb	Lattice HW-USBN-2A USB port programming cable.
-cabletype usb2	Lattice FHW-USBN-2B (FTDI) USB programming cable and any FTDI based demo boards.

Parallel Port Address (optional)

Option	Description
-portaddress 0x0378	LPT1 parallel port (default)
-portaddress 0x0278	LPT2 parallel port
-portaddress 0x03BC	LPT3 parallel port
-portaddress 0x*<custom address>*	Custom parallel port address

This option is only valid with parallel port cables.

USB Port Address (optional)

Option	Description
-portaddress EZUSB-0 ... EZUSB-15	HW-USBN-2A USB cable number 0 through 15
-portaddress FTUSB-0 ... FTUSB-15	FTDI based demo board or FTDI USB2 cable number 0 through 15

Default is EZUSB-0 and FTUSB-0. Only valid with the USB port cables.

FTDI Based Demo Board or Cable Frequency Control (optional)

Option	Description
-TCK 0, 1, 2, 3, 4, ,5 ,6 7, 8, 9, 10	0 = 30 MHz 1 = 15 MHz (default) 2 = 10 MHz 3 = 7.5 MHz 4 = 6 MHz 5 = 5 MHz 6 = 4 MHz 7 = 3 MHz 8 = 2 MHz 9 = 1 MHz 10 = 900 kHz

Calculation formula for USB-2B (2232H FTDI USB host chip): Frequency = 60 MHz / (1 + ClockDivider) *2

Calculation formula for USB-2B (2232D FTDI USB host chip): Frequency = 12 MHz / (1 + ClockDivider) *2

Only applicable for FTDI based demo boards or programming cable.

Return Codes

Code	Definition
0	Success
-1	Log file error
-2	Check configuration setup error
-3	Out of memory error
-4	NT driver error
-5	Cable not detected error
-6	Power detection error
-7	Device not valid error
-8	File not found error
-9	File not valid error
-10	Output file error
-11	Verification error
-12	Unsupported operation error
-13	File name error
-14	File read error
-17	Build SVF file error
-18	Build VME file error
-19	Command line syntax error

Example

The following is a PGRCMD example.

```
pgrcmd -infile c:\test.xcf
```

See Also

- Command Line Data Flow
- Command Line Program Overview

Running the Deployment Tool from the Command Line

You can run the Deployment Tool from the command line. The **DDTCMD** command uses a keyword preceded by a hyphen for each command line option.

Note

If you use the Deployment Tool GUI, the correct command line for the operation you are performing will be displayed in the GUI in the “Step 4 of 4 - Generate Deployment” dialog box. The command line is also recorded in the Deployment Project File (*.ddt). You can view the Deployment Project File with a text editor and view or copy the command line.

Subjects included in this topic:

- Running DDTCMD
- Command Line Syntax
- DDTCMD Options
- Examples

Running DDTCMD

DDTCMD allows you to convert data files to other formats for one device at a time and devices in a chain. DDTCMD can also use the data files it produces to generate other data file formats.

- To run DDTCMD, type **ddtcmd** on the command line with, at minimum, the **ddtcmd** command.

There are several command line options that give you control over the way **DDTCMD** converts data files. Please refer to the rest of the subjects in this topic for more details.

Command Line Syntax

The following describes the DDTCMD command line syntax:

<*install_path*>**ddtcmd**

[**-h**] |

-oft <**-bit | rbt** >

[**-dev** <Device Name>]

-if "<Data File Path>"

[-**of** "<Data File Path>"]

[**-compress** <on | off>]

[**-verifyid** <on | off>]

[**-freq**]

[**-crc <**frame | global>]

[**-mirror**]

[**-header**]

[**-secure** <on | off>]

[**-overwriteues on** <Hex Value>]

[**-encryption -key hex** <Valid Key Value>

-config_mode <slave_scm | slave_pcm | jtag | spi | spim | master_pcm | slave_spi>]

|

-oft <**-int | mot | xtek**>

[**-dev** <Device Name>]

-if "<Data File Path>"

[-**of** "<Data File Path>"]

[**-compress** <on | off>]

[**-verifyid** <on | off>]

[**-freq**]

[**-crc <**frame | global>]

[**-mirror**]

[**-header**]

[**-secure** <on | off>]

[**-overwriteues on** <Hex Value>]

[**-encryption -key hex** <Valid Key Value>

-config_mode <slave_scm | slave_pcm | jtag | spi | spim | master_pcm | slave_spi>]

[**-address** <hex address>]

[**-fast**]

|

-oft -svf

[**-dev** <Device Name>]

-if "<Data File Path>"

[**-of** "<Data File Path>"]

-op <Operation>

[**-nocomments**]

[**-revd**]

[**-runtest**]

[**-skipverify**]

[**-reset**]

[**-maxdata** <8 | 16 | 32 | 64 | 128 | 256>]

|

-oft -svfchain

-if "<Data File Path>"

[**-of** "<Data File Path>"]

[**-nocomments**]

[**-revd**]

[**-runtest**]

[**-skipverify**]

[**-reset**]

[**-maxdata** <8 | 16 | 32 | 64 | 128 | 256>]

|

-oft -stp

[**-dev** <Device Name>]

-if "<Data File Path>"

[**-of** "<Data File Path>"]

-op <Operation>

[**-nocompress**]

[**-print**]

[**-skipverify**]

|

-oft -stpchain

-if "<Data File Path>"

[**-of** "<Data File Path>"]

[**-nocompress**]

[**-print**]

[**-skipverify**]

|

oft -ate

-if "<Data File Path>"

[**-of** "<Data File Path>"]

-type <tst | gr | hp3070 | hp3065 | t1800 | t200>

[**-ispdcd**]

[**-skipverify**]

[**-headerfile** <file path>]

[**-splitfile** [**-init**] [**-vectors** <# vectors/file>] [**-splitatlow**]]

|

-oft -**fullvme**

-if "<Data File Path>"

[**-of** "<Data File Path>"]

[**-maxdata** <8 | 16 | 32 | 64 | 128 | 256>]

[**-hex**]

[**-nocompress**]

[**-compact**]

[**-fixedpulse**]

[-**verifyues**]

[**-noheader**]

[**-comment**]

|

-oft -slimvme

-if "<Data File Path>"

[**-ofa** <Algorithm File Path>]

[**-ofd** "<Data File Path>"]

[**-nocompress**]

[**-hex**]

|

-oft -sspi

-if "<Data File Path>"

[**-ofa** <Algorithm File Path>]

[**-ofd** "<Data File Path>"]

[**-nocompress**]

[**-hex**]

[**-op** <Operation>]

|

-oft -i2c

-if "<Data File Path>"

[**-ofa** <Algorithm File Path>]

[**-ofd** "<Data File Path>"]

[**-nocompress**]

[**-hex**]

[**-op** <Operation>]

-address <80 | (User Specified*)*>

[**-comment**]

[**-fixedpulse**]

|

-oft -cpu

-if "<Data File Path>"

[**-of** "<Data File Path>"]

- type <cpu **|** c **|** hex | txt>

[**-verify**]

[**-comment**]

[**-nocompress**]

[**-mirror**]

|

-oft -boot

[**-dev** <Device Name>]

-golden "<Data File Path>"

-primary "<Data File Path>"

-format <int | mot | xtek>

[**-of** "<*D*ata File Path>"]

-flashsize <1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256>

[**-protect**]

[**-mirror**]

[**-header**]

[**-preamble**]

[**-optimize**] [**-fast** | **-dualio** | **-quad 1**]

[**-asc** <1 | 2 | 3 | 4 | 5 | 6 | 7 | 8>

-ascfile “<Data File Path>”

[**-ascfile** “<Data File Path>”] … [**-ascfile** “<Data File Path>”]]

|

-oft -jed

-if “<Data File Path>”

[**-dev** <Device Name>]

[**-of** “<Data File Path>”]

[**-overwriteues** <**on** "<hex value>" **| checksum | disable**>]

[**-comment** “<Header File Path>”]

[**-encryption -key hex** “<valid key value>”]

|

-oft -advanced

[**-dev** <Device Name>]

-if “<Data File Path>”

-format <int | mot | xtek>

[**-of** “<Data File Path>”]

-flashsize <512 | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 513>

[**-mirror**]

[**-header**]

[**-optimize**]

[**-preamble**]

[**-userdata** <1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16>

-usermirror

-userfile “<Data File Path>”

-address <hex address value>

[-**userfile** “<Data File Path>”

-address <hex address value>] ...

[**-userfile** “<Data File Path>”

-address <hex address value>]]

[**-ice** [**-warm** <1 | 2 | 3 | 4>]

-patterns <2,3,4>

-bootfile “<Data File Path>”

-address <hex address value>

[**-bootfile** “<Data File Path>”

-address <hex address value>] ...

[**-bootfile** “<Data File Path>”

-address <*hex address value*>]]

[**-fast | -dualio | -quad 1**>]

[**-multi** <1 | 2 | 3 | 4>

-golden "<Data File Path>"

-altfile “<Data File Path>”

-address <hex address value>

-next <primary | alt1 | alt2 | alt3 | alt4> [

-altfile “<Data File Path>”

-address <hex address value>

-next <primary | alt1 | alt2 | alt3 | alt4>] ...

[**-altfile** “<Data File Path>”

-address <hex address value>

-next <primary|alt1|alt2|alt3|alt4>]]

|

-oft -merge

-ifdev1 “<Data File Path>”

-ifdev2 “<Data File Path>”

-format <int | mot| xtek>

[**-of** “<Data File Path>”]

[**-mergeformat** <intelligent | combine>

[**-frequency** <freq>]

[**-scoutput** <dout | qout>]]

[**-mirror**]

[-**header**]

|

-oft -bsm

-ifd “<Data File Path>”

-ifb “<BSDL File Path>”

[**-dev** <Device Name>]

[**-of** “<Data File Path>”]

[**-convertbidi**]

DDTCMD Options

The following are DDTCMD options.

Option	Description
-h	Displays the Deployment Tool command line usage in the terminal or DOS window.
-oft <File Type>	Indicates which file type(s) will be generated.

Option	Description
-bit	Generate binary bitstream file. **-dev** <Device Name>: Indicates the name of the device (optional). The device name must be the full device name with the performance grade replaced with two X's, and minus the industry rating or the code name. Example: `LFE3-17EA-XXFN484` (with performance grade replaced with two X's). The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-of** "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default file name is the same name as the input file. binary bitstream files should end with the *.bit extension. **-compress** <on | off>: Specifies compression (optional). Default is off. **-verifyid**: On: Include the Verify ID frame. Off: Replaces Verify ID frame with NOOP (optional). Default is off. **-freq**: Specifies the frequency of the master bitstream. Possible frequencies valid for LatticeEC/P, LatticeECP2/M, LatticeECP3, and LatticeXP/2 devices only are: 2.5, 4.3, 5.4, 6.9, 8.1, 9.2, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60, and 130 MHz. Possible frequencies valid for for MachXO2 and MachXO3L devices only are: 2.1, 2.5, 3.2, 4.3, 5.5, 7, 8.3, 9.2, 10, 13, 15, 20, 27, 30, 33, 38, 44, 53, 66, 89, and 133 MHz. Possible frequencies valid for for ECP5 devices only are: 2.4, 3.2, 4.1, 4.8, 6.5, 8.2, 9.7, 12.9, 15.5, 16.3, 19.4, 20.7, 25.8, 31, 34.4, 38.8, 44.3, 51.7, 62, 77.5, 103.3, and 155 MHz. **-crc** <frame | global>: The "frame" option includes the CRC checking for each data frame. The "global option disables the frames CRC but still calculates the global CRC at the end of the configuration data. Default uses the settings of the input file. Valid for ECP5. LatticeECP2/M, LatticeECP2S/SM, LatticeECP3, MachXO3L, MachXO2, and LatticeXP/2 devices only. **-secure** <on | off>: Specifies program security (optional). On: Programs the Security Fuse (blocks read back). Off: Does not program the Security Fuse (allows read back).Default uses the settings of the input file. **-overwriteues on**: Overwrites (or replaces) the USERCODE value contained in the input file with the hex value passed in. **Note**: The encryption software patch must be installed before using the following **-encryption -key** and **-config_mode** commands. **-encryption -key hex** <Valid Key Value>: Encrypts the input file using the -key 128-bit encryption key. **-config_mode** <slave_scm | slave_pcm | jtag | spi | spim | master_pcm | slave_spi>]: Selects the appropriate configuration mode for the bitstream encryption. Valid for LatticeECP2S/MS and LatticeECP3 devices only.

Option	Description
-rbt	Generate ASCII Bitstream file. **-dev** <Device Name>: Indicates the name of the device (optional). The device name must be the full device name with the performance grade replaced with two X's, and minus the industry rating or the code name. Example: `LFE3-17EA-XXFN484` (with performance grade replaced with two X's). The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-of** "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes. Optional. Default file name is the same name as the input file. ASCII bit files should end with the *.rbt extension. **-compress** <on | off>: Specifies compression (optional). Default is off. **-verifyid**: On: Include the Verify ID frame. Off: Replaces Verify ID frame with NOOP (optional). Default is off. **-freq**: Specifies the frequency of the master bitstream. Possible frequencies valid for LatticeEC/P, LatticeECP2/M, LatticeECP3, and LatticeXP/2 devices only are: 2.5, 4.3, 5.4, 6.9, 8.1, 9.2, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60, and 130 MHz. Possible frequencies valid for for MachXO2 and MachXO3L devices only are: 2.1, 2.5, 3.2, 4.3, 5.5, 7, 8.3, 9.2, 10, 13, 15, 20, 27, 30, 33, 38, 44, 53, 66, 89, and 133 MHz. Possible frequencies valid for for ECP5 devices only are: 2.4, 3.2, 4.1, 4.8, 6.5, 8.2, 9.7, 12.9, 15.5, 16.3, 19.4, 20.7, 25.8, 31, 34.4, 38.8, 44.3, 51.7, 62, 77.5, 103.3, and 155 MHz. **-crc** <frame | global>: The "frame" option includes the CRC checking for each data frame. The "global option disables the frames CRC but still calculates the global CRC at the end of the configuration data. Default uses the settings of the input file. Valid for LatticeECP2/M, LatticeECP2S/SM, LatticeECP3, ECP5, MachXO3L, MachXO2, and LatticeXP/2 devices only. **-secure** <on | off>: Specifies program security (optional). On: Programs the Security Fuse (blocks read back). Off: Does not program the Security Fuse (allows read back).Default uses the settings of the input file. **-overwriteues on**: Overwrites (or replaces) the USERCODE value contained in the input file with the hex value passed in. **Note**: The encryption software patch must be installed before using the following **-encryption -key** and **-config_mode** commands. **-encryption -key hex** <Valid Key Value>: Encrypts the input file using the -key 128-bit encryption key. **-config_mode** <slave_scm | slave_pcm | jtag | spi | spim | master_pcm | slave_spi>]: Selects the appropriate configuration mode for the bitstream encryption. Valid for LatticeECP2S/MS and LatticeECP3 devices only.

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>**-int**</td><td>Generate Intel 32-bit hex file.

-dev <Device Name>: Indicates the name of the device. Optional. The device name must be the full device name with the performance grade replaced with two X’s, and minus the industry rating or the code name.

Example:

`LFE3-17EA-XXFN484` (with performance grade replaced with two X's).

The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option.

-if <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes.

-of "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes. Optional. Default file name is the same name as the input file. Intel files should end with the *.mcs extension.

-compress <on | off>: Specifies compression (optional). Default is off.

-verifyid: On: Include the Verify ID frame. Off: Replaces Verify ID frame with NOOP (optional). Default is off.

-freq: Specifies the frequency of the master bitstream.

Possible frequencies valid for LatticeEC/P, LatticeECP2/M, LatticeECP3, and LatticeXP/2 devices only are: 2.5, 4.3, 5.4, 6.9, 8.1, 9.2, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60, and 130 MHz.

Possible frequencies valid for for MachXO2 and MachXO3L devices only are: 2.1, 2.5, 3.2, 4.3, 5.5, 7, 8.3, 9.2, 10, 13, 15, 20, 27, 30, 33, 38, 44, 53, 66, 89, and 133 MHz.

Possible frequencies valid for for ECP5 devices only are: 2.4, 3.2, 4.1, 4.8, 6.5, 8.2, 9.7, 12.9, 15.5, 16.3, 19.4, 20.7, 25.8, 31, 34.4, 38.8, 44.3, 51.7, 62, 77.5, 103.3, and 155 MHz.

-crc <frame | global>: The “frame” option includes the CRC checking for each data frame. The "global option disables the frames CRC but still calculates the global CRC at the end of the configuration data. Default uses the settings of the input file. Valid for ECP5, LatticeECP2/M, LatticeECP2S/SM, LatticeECP3, MachXO3L, MachXO2, and LatticeXP/2 devices only.

-mirror: Flips each byte. Default: bytes are not flipped. Optional.

-header: Retains the input file header in the output file. Default: Replaces header with all ones (0xFF). Optional.

-secure <on | off>: Specifies program security (optional). On: Programs the Security Fuse (blocks read back). Off: Does not program the Security Fuse (allows read back). Default uses the settings of the input file.

-overwriteues on: Overwrites (or replaces) the USERCODE value contained in the input file with the hex value passed in.

Note: The encryption software patch must be installed before using the following **-encryption -key** and **-config_mode** commands.

-encryption -key hex <Valid Key Value>: Encrypts the input file using the -key 128-bit encryption key.</td></tr>
</table>

Option	Description
-int (cont.)	**-config_mode** <slave_scm | slave_pcm | jtag | spi | spim | master_pcm | slave_spi>]: Selects the appropriate configuration mode for the bitstream encryption. Valid for LatticeECP2S/MS and LatticeECP3 devices only. **-address** <hex address> Specifies the starting address of the memory device to store the input file. **-fast**: Using this option, the FPGA will issue the Fast Read command (0x0B) to the SPI Flash, instead of the Slow Read command (0x03). Valid for ECP5.
-mot	Generate Motorola 32-bit hex file. **-dev** <Device Name>: Indicates the name of the device. Optional. The device name must be the full device name with the performance grade replaced with two X's, and minus the industry rating or the code name. Example: `LFE3-17EA-XXFN484` (with performance grade replaced with two X's). The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-of** "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes. Optional. Default file name is the same name as the input file. Motorola files should end with the *.exo extension. **-compress** <on | off>: Specifies compression (optional). Default is off. **-verifyid**: On: Include the Verify ID frame. Off: Replaces Verify ID frame with NOOP (optional). Default is off. **-freq**: Specifies the frequency of the master bitstream. Possible frequencies valid for LatticeEC/P, LatticeECP2/M, LatticeECP3, and LatticeXP/2 devices only are: 2.5, 4.3, 5.4, 6.9, 8.1, 9.2, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60, and 130 MHz. Possible frequencies valid for for MachXO2 and MachXO3L devices only are: 2.1, 2.5, 3.2, 4.3, 5.5, 7, 8.3, 9.2, 10, 13, 15, 20, 27, 30, 33, 38, 44, 53, 66, 89, and 133 MHz. Possible frequencies valid for for ECP5 devices only are: 2.4, 3.2, 4.1, 4.8, 6.5, 8.2, 9.7, 12.9, 15.5, 16.3, 19.4, 20.7, 25.8, 31, 34.4, 38.8, 44.3, 51.7, 62, 77.5, 103.3, and 155 MHz. **-crc** <frame | global>: The "frame" option includes the CRC checking for each data frame. The "global option disables the frames CRC but still calculates the global CRC at the end of the configuration data. Default uses the settings of the input file. Valid for ECP5, LatticeECP2/M, LatticeECP2S/SM, LatticeECP3, MachXO3L, MachXO2, and LatticeXP/2 devices only. **-mirror**: Flips each byte. Default: bytes are not flipped. Optional. **-header**: Retains the input file header in the output file. Default: Replaces header with all ones (0xFF). Optional. **-secure** <on | off>: Specifies program security (optional). On: Programs the Security Fuse (blocks read back). Off: Does not program the Security Fuse (allows read back). Default uses the settings of the input file.

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>-mot (cont.)</td><td>-overwriteues on: Overwrites (or replaces) the USERCODE value contained in the input file with the hex value passed in.

Note: The encryption software patch must be installed before using the following -encryption -key and -config_mode commands.

-encryption -key hex <Valid Key Value>: Encrypts the input file using the -key 128-bit encryption key.

-config_mode <slave_scm | slave_pcm | jtag | spi | spim | master_pcm | slave_spi>]: Selects the appropriate configuration mode for the bitstream encryption. Valid for LatticeECP2S/MS and LatticeECP3 devices only.

-address <hex address> Specifies the starting address of the memory device to store the input file.

-fast: Using this option, the FPGA will issue the Fast Read command (0x0B) to the SPI Flash, instead of the Slow Read command (0x03). Valid for ECP5.</td></tr>
<tr><td>-xtek</td><td>Generate Tektronix Extended hex file

-dev <Device Name>: Indicates the name of the device (optional). The device name must be the full device name with the performance grade replaced with two X’s, and minus the industry rating or the code name.

Example:

<code>LFE3-17EA-XXFN484</code> (with performance grade replaced with two X's).

The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option.

-if <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes.

-of "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes. Optional. Extended Tektronix files should end with the *.xtek extension.

-compress <on | off>: Specifies compression (optional). Default is off.

-verifyid: On: Include the Verify ID frame. Off: Replaces Verify ID frame with NOOP (optional). Default is off.

-freq: Specifies the frequency of the master bitstream.

Possible frequencies valid for LatticeEC/P, LatticeECP2/M, LatticeECP3, and LatticeXP/2 devices only are: 2.5, 4.3, 5.4, 6.9, 8.1, 9.2, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60, and 130 MHz.

Possible frequencies valid for MachXO2 and MachXO3L devices only are: 2.1, 2.5, 3.2, 4.3, 5.5, 7, 8.3, 9.2, 10, 13, 15, 20, 27, 30, 33, 38, 44, 53, 66, 89, and 133 MHz.

Possible frequencies valid for ECP5 devices only are: 2.4, 3.2, 4.1, 4.8, 6.5, 8.2, 9.7, 12.9, 15.5, 16.3, 19.4, 20.7, 25.8, 31, 34.4, 38.8, 44.3, 51.7, 62, 77.5, 103.3, and 155 MHz.

-crc <frame | global>: The “frame” option includes the CRC checking for each data frame. The "global option disables the frames CRC but still calculates the global CRC at the end of the configuration data. Default uses the settings of the input file. Valid for ECP5, LatticeECP2/M, LatticeECP2S/SM, LatticeECP3, MachXO3L, MachXO2, and LatticeXP/2 devices only.</td></tr>
</table>

Option	Description
-xtek (cont.)	**-mirror**: Flips each byte. Default: bytes are not flipped (optional). **-header**: Retains the input file header in the output file. Default: Replaces header with all ones (0xFF) (optional). **-secure** <on | off>: Specifies program security (optional). On: Programs the Security Fuse (blocks read back). Off: Does not program the Security Fuse (allows read back).Default uses the settings of the input file. **overwriteues on**: Overwrites (or replaces) the USERCODE value contained in the input file with the hex value passed in. **Note**: The encryption software patch must be installed before using the following **-encryption -key** and **-config_mode** commands. **-encryption -key hex** <Valid Key Value>: Encrypts the input file using the -key 128-bit encryption key. **-config_mode** <slave_scm | slave_pcm | jtag | spi | spim | master_pcm | slave_spi>]: Selects the appropriate configuration mode for the bitstream encryption. Valid for LatticeECP2S/MS and LatticeECP3 devices only. **-address** <hex address> Specifies the starting address of the memory device to store the input file. **-fast**: Using this option, the FPGA will issue the Fast Read command (0x0B) to the SPI Flash, instead of the Slow Read command (0x03). Valid for ECP5.
-svf	Generate SVF file for a single device. **-dev** <Device Name>: Indicates the name of the device. Optional. The device name must be the full device name with the performance grade replaced with two X's, and minus the industry rating or the code name. Example: `LCMXO2-256ZE-XXMG64` (with performance grade replaced with two X's). The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-of** "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default file name is the same name as the input file. SVF files should end with the *.svf extension. -**op** Indicates the operation. The operation list varies depending on the device family. Use the Deployment Tool GUI to create the command line. **-nocomments**: Omit Header and Comments from SVF file (optional). Default includes Header and Comments. **-revd**: Generate Rev D standard SVF file (optiona)l. Default generates Lattice Extended SVF file (not supported by all third party tools). **-runtest**: Uses RUNTEST format from SVF revision C (optional). Default is revision D SVF file. **-skipverify:** Generates an SVF without the verify portion (optional). **-reset**: Writes RESET in the SVF file (optional). **-maxdata** [<8 | 16 | 32 | 64 | 128 | 256>]: Specifies the maximum data size per row (in Kbits) (optional).

Option	Description
-svfchain	Generate SVF file for chain of devices. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-of** "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default file name is the same name as the input file. SVF files should end with the *.svf extension. **-nocomments**: Omit Header and Comments from SVF file (optional). Default includes Header and Comments. **-revd**: Generate Rev D standard SVF file (optional). Default generates Lattice Extended SVF file (not supported by all third party tools). **-runtest**: Uses RUNTEST format from SVF revision C (optional). Default is revision D SVF file. **-skipverify:** Generates an SVF without the verify portion (optional). **-reset**: Writes RESET in the SVF file (optional). **-maxdata** [<8 | 16 | 32 | 64 | 128 | 256>]: Specifies the maximum data size per row (in Kbits) (optional).
-stp	Generate Standard Test and Programming Language (STAPLE) file. **-dev** <Device Name>: Indicates the name of the device (optional). The device name must be the full device name with the performance grade replaced with two X's, and minus the industry rating or the code name. Example: `LCMXO2-256ZE-XXMG64` (with performance grade replaced with two X's). The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-of** "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default uses the input file name with *.stp extension. -**op** Indicates the operation. The operation list varies depending on the device family. Use the Deployment Tool GUI to create the command line. **-nocompress:** < ON | OFF >: Generates a compressed or uncompressed STAPL file. Default is ON. **-print** < ON | OFF >: Generates a STAPL file with all the comment statements. Default is OFF. **-skipverify**: < ON | OFF >: Generates a STAPL file without the Verify portion (optional).

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>-stpchain</td><td>Generate Standard Test and Programming Language (STAPL) file for a chain of devices.
-if <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes.
-of "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes. Optional. Default uses the input file name with *.stp extension.
-nocompress: < ON | OFF >: Generates a compressed or uncompressed STAPL file. Default is ON.
-print < ON | OFF >: Generates a STAPL file with all the comment statements. Default is OFF.
-skipverify: < ON | OFF >: Generates a STAPL file without the Verify portion (optional).</td></tr>
<tr><td>-ate</td><td>Generate ATE file.
-if <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes.
-of "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default uses the input file name with *.tst, *.gr, *.pcf, or *.asc extension, descending type.
-type: <tst | gr | hp3070 | hp3065 | t1800 | t200>
-ispdcd: Generates vectors in ispDCD Format.
-skipverify: For Erase, Program, and Verify Operations, Skip Verify.
-headerfile <file path>: Specifies a customer header file.
-splitfile [-init] [-vectors <# vectors/file>] [-splitatlow]: Split each file at active Low. Default: Split each file at active high.</td></tr>
<tr><td>-fullvme</td><td>Generate JTAG Full VME Embedded file.
-if <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes.
-of "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default uses the input file name with *.vme extension.
-maxdata <8 | 16 | 32 | 64 | 128 | 256>: Maximum Memory Allocation Size Per Row of Data (Kbytes) (optional).
-hex: Generate a Hex (.c) file along with the VME file (optiona)l.
-nocompress: Do not compress data (optional).
-compact: Compact VME file. Optional.
-fixedpulse: Generate a fixed pulse width VME file (optional).
-verifyues: Test the USERCODE. Program the device if USERCODE fails verify (optional).
-noheader: Omit header from VME file (optional).
-comment: Include comments in VME file (optional).</td></tr>
</table>

Option	Description
-slimvme	Generate JTAG Slim VME Embedded file. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-ofa** "<Algorithm File Path>": Programming Algorithm file (optional). Default use input with *.algo.vme. **-ofd** "<Data File Path>": Programming Data file (optional). Default uses input with *.data.vme. **-nocompress**: Do not compress data (optional). **-hex**: Generate a Hex (.c) file along with the Slim VME file (optional).
-sspi	Generate Target SPI embedded file. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-ofa** "<Algorithm File Path>": Programming Algorithm file (optional). Default use input with *.algo.vme. **-ofd** "<Data File Path>": Programming Data file (optional). Default uses input with *.data.vme. **-nocompress**: Do not compress data (optional). **-hex**: Generate a Hex (.c) file along with the Target SPI Embedded file (optional). -**op** Indicates the operation. This is required when the input file is JEDEC. When the input file is an XCF, the XCF contains the operation.
-i2c	Generate I2C embedded file. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-ofa** "<Algorithm File Path>": Programming Algorithm file (optional). Default use input with *.algo.vme. **-ofd** "<Data File Path>": Programming Data file (optional). Default uses input with *.data.vme. **-nocompress**: Do not compress data (optional). **-hex**: Generate a Hex (.c) file along with the VME file (optional). -**op** Indicates the operation. This is required when the input file is JEDEC. When the input file is an XCF, the XCF contains the operation. **-address** <80 | (User Specified)> Specifies the slave I2C address. **-comment:** Include comments in I2C embedded file (optional). **-fixedpulse**: Generate a fixed pulse width I2C embedded file (optional).

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>-cpu</td><td>Generates sysCONFIG CPU Embedded file.

-if <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes.

-of "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default uses the input file name with the extension of the selected -type.

-type <cpu | c | hex | txt>: Indicates whether file is in binary embedded (cpu), C-code (c), Intel Hex (hex), or Text debug only (txt) format.

-verify: Generates a sysCONFIG Embedded file that includes a verify operation.

-comment: Include comments in sysCONFIG Embedded file (optional).

-nocompress: Do not compress data (optional).

-mirror: Flips each byte. Default: bytes are not flipped (optional).</td></tr>
<tr><td>-boot</td><td>Generate Dual Boot Hex file.

-dev <Device Name>: Indicates the name of the device (optional). The device name must be the full device name with the performance grade replaced with two X’s, and minus the industry rating or the code name.

Example:

<code>LFE3-17EA-XXFN484</code> (with performance grade replaced with two X's).

The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option.

-golden "<Golden File Path>" -primary "<Primary File Path>": Merges the Golden, Primary, and a JUMP command into one Hex file. The address location of the Golden, Primary, and JUMP command depends on the files sizes and the target device. The Deployment Toll will record the address location of each in the Status window and log file.

-format < intel | motorola | xtek >: Specifies the output format as Intel, Motorola, or Tektronix Extended Hex. The default format is Intel Hex (optional). Default uses the input file name with *.mcs, *exo, or *.xtek extension, depending on format type selected.

-of "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default uses the input file name with *.mcs, *exo, or *.xtek extension, depending on format type selected.

-flashsize: Specifies size in Mb.

Possible PROM sizes are 1, 2, 4, 8, 16, 32, 64, 128, 256, and 513 (512Mb).

Default: 1.</td></tr>
</table>

Option	Description
-boot (cont.)	**-protect:** Inserts the Golden File at the beginning of the upper half of the SPI flash (optional). This allows the upper half of the SPI flash to be write-protected, which protects the Golden File from accidental erase or reprogram. **-mirror**: Flips each byte. Default: bytes are not flipped (optional). **-header**: Retains the input file header in the output file. Default: Replaces header with all ones (0xFF) (optional). **-preamble**: Replaces the header Preamble of the primary bitstream with 0xFFs (for Fail Safe Upgrade feature) (optional). **-optimize**: Uses the files size of the input files to determine SPI flash sector addresses for each input file. Use this option to minimize wasted memory space. The default is to use worst case file size fo the target device, which is an uncompressed file with all EBR and PCS, depending on the family. Using the worst case will ensure that all files will always be located at the same address even if any of the files/designs are updated at a later date. **-asc** <1 | 2 | 3 | 4 | 5 | 6 | 7 | 8> Indicates the number of ASC devices connected to the MachXO2 or Platform Manager 2 device. The maximum number that can be used with the MachXO2 device is 8. The maximum number that can be used with the Platform Manager 2 device is 7. The input JEDEC file and the ASC Hex files will be merged into one Hex file that can be programmed into an external SPI Flash device for Dual Boot support. Optional. **-ascfile** "<Data File Path>": Specifies the ASC Hex file path and name. If the file path includes spaces, enclose the path in quotes. There must be one "-ascfile" and path for each ASC specified by the "-asc" Optional. <**-fast** | **-dualio** | **-quad 1**>: With the -fast option, the FPGA will issue the Fast Read command (0x0B) to the SPI Flash. The -dualio option must be used if a Dual I/O SPI Flash is used. The -quad 1 option must be used if a Quad I/O SPI Flash is used. Only one of these options (-fast, -dualio, -quad 1) can be used. Without any of these options, the FPGA uses the Slow Read command (0x03). Valid for ECP5. **-encryption -key hex** <Valid Key Value>: Encrypts the input file using the -key 128-bit encryption key.

Option	Description
-jed	Generates JEDEC file. **-if** <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes. **-dev** <Device Name>: Indicates the name of the device (optional). The device name must be the full device name with the performance grade replaced with two X's, and minus the industry rating or the code name. Example: `LCMXO2-256ZE-XXMG64` (with performance grade replaced with two X's). The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option. **-of** "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default uses the input file name with *.jed extension. **-overwriteues** <**on**>: Overwrites (or replaces) the USERCODE value contained in the input file with the hex value passed in. The Checksum option overwrites the USERCODE with the JEDEC file fuse checksum value. The Disable option omits the USERCODE from the JEDEC file. **Note**: The encryption software patch must be installed before using the following **-encryption -key** commands. **-encryption -key hex** <Valid Key Value>: Encrypts the input file using the -key 128-bit encryption key.

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>-advanced</td><td>Generates Advanced SPI Flash Hex file.

-dev <Device Name>: Indicates the name of the device (optional). The device name must be the full device name with the performance grade replaced with two X’s, and minus the industry rating or the code name.

Example:

<code>LFE3-17EA-XXFN484</code> (with performance grade replaced with two X's).

The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option.

-if <Data File Path>: Indicates the input data file full path and file name. If the file path includes spaces, enclose the path in quotes.

-format < intel | motorola | xtek >: Specifies the output format as Intel, Motorola, or Tektronix Extended Hex. The default format is Intel Hex (optional).

-of "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default uses the input file name with *.mcs. *.exo, or *.xtek extension, depending on the format selected.

-flashsize: Specifies size in Mb. Possible PROM sizes are 512 (512Kb), 1, 2, 4, 8, 16, 32, 64, 128 , 256, and 513 (512Mb). Default: 1.

-mirror: Flips each byte. Default: bytes are not flipped (optional).

-header: Retains the input file header in the output file. Default: Replaces header with all ones (0xFF) (optional).

-preamble: Replaces the header Preamble of the bitstream with 0xFFs (for Fail Safe Upgrade feature) (optional).

-userdata <<i>1</i> | <i>2</i> | <i>3</i> | <i>4</i> | <i>5</i> | <i>6</i> | <i>7</i> | <i>8</i> | <i>9</i> | <i>10</i> | <i>11</i> | <i>12</i> | <i>13</i> | <i>14</i> | <i>15</i> | <i>16</i>> Number of data files.

-userdata: Flips each byte of the user data files. Default: bytes are not flipped (optional).

-userfile “<Data File Path>” Indicates the input user data file path and file name. Must be specified for each user data file (-userdata).

-address <hex address value>: SPI flash address location for the user data file. Must be specified for each user data file (-userdata).

-optimize: Uses the files size of the input files to determine SPI flash sector addresses for each input file. Use this option to minimize wasted memory space. The default is to use worst case file size fo the target device, which is an uncompressed file with all EBR and PCS, depending on the family. Using the worst case will ensure that all files will always be located at the same address even if any of the files/designs are updated at a later date.

-ice: iCE40 warm boot cold boot.

-warm <1 | 2 | 3 | 4>: Optional. Default is cold boot. The -warm selects warm boot. The number selects the first boot configuration file.

-patterns <2,3,4>: Specifies the number of iCE40 configuration files.

-bootfile “<Data File Path>”: Indicates the input configuration file path and name. Must be specified for each configuration file (-patterns).

-address <hex address value>: SPI flash address location for the configuration file. Must be specified for each configuration file (-patterns).

<-fast | -dualio | -quad 1>: With the -fast option, the FPGA will issue the Fast Read command (0x0B) to the SPI Flash. The -dualio option must be used if a Dual I/O SPI Flash is used. The -quad option must be used if a Quad I/O SPI Flash is used. Only one of these options (-fast, -dualio, -quad 1) can be used. Without any of these options, the FPGA uses the Slow Read command (0x03). Valid for ECP5.</td></tr>
</table>

<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>-advanced (cont.)</td><td>-multi <1 | 2 | 3 | 4> Optional. The -multi option selects Multiple Boot. The number selects the number of alternate files. Must be used in conjunction with Dual Boot, since dual boot is a subset of Multiple Boot feature. The Golden file must also be selected. The data file passed in using the -if option is considered the Primary file.

-golden "<Data File Path>" Indicates the Golden file path and file name.

-altfile “<Data File Path>” Indicates the alternate file path and name. Must specify the alternate file for each number following the -multi option.

-address <hex address value> SPI Flash address location for the alternate file. Must be specified for each alternate file (-altfile).

-next <primary | alt1 | alt2 | alt3 |alt4> Indicates which will be the next pattern to boot from when the PROGRAMN pin is toggle, or the Refresh command is issued. Must be specified for each alternate file (-altfile).</td></tr>
<tr><td>-merge</td><td>-ifdev1 “<Data File Path>” Indicates the input dta file full path and file name for the first device in the chain. If the file path includes spaces.

-ifdev2 “<Data File Path>” “Second device in the chain.”

-format < intel | motorola | xtek >: Specifies the output format as Intel, Motorola, or Tektronix Extended Hex. The default format is Intel Hex (optional).

-of "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default uses the input file name with *.mcs. *.exo, or *.xtek extension, depending on the format selected.

-mergeformat <intelligent | combine> Intelligent merge will set the appropriate bits in the bitstream to support sysCONFIG Daily Chaining. Combine will merge the two files without changing any bits in the bitstreams.

-freq: Specifies the frequency of the master bitstream.

Possible frequencies valid for LatticeEC/P, LatticeECP2/M, LatticeECP3, and LatticeXP/2 devices only are: 2.5, 4.3, 5.4, 6.9, 8.1, 9.2, 10, 13, 15, 20, 26, 30, 34, 41, 45, 51, 55, 60, and 130 MHz.

Possible frequencies valid for for MachXO2 and MachXO3L devices only are: 2.1, 2.5, 3.2, 4.3, 5.5, 7, 8.3, 9.2, 10, 13, 15, 20, 27, 30, 33, 38, 44, 53, 66, 89, and 133 MHz.

Possible frequencies valid for for ECP5 devices only are: 2.4, 3.2, 4.1, 4.8, 6.5, 8.2, 9.7, 12.9, 15.5, 16.3, 19.4, 20.7, 25.8, 31, 34.4, 38.8, 44.3, 51.7, 62, 77.5, 103.3, and 155 MHz.

-scoutput <dout | qout>: For LatticeSC/M devices, indicates if the output pin is Dout or Qout.

-mirror: Flips each byte. Default: bytes are not flipped (optional).

-header: Retains the input file header in the output file. Default: Replaces header with all ones (0xFF). Optional.</td></tr>
</table>

Option	Description
-bsm	Generates Application Specific BSDL file. **-ifd** “<Data File Path>”: Indicates the input data file full path and file name.If the file name includes spaces ... **-ifb** “<BSDL File Path>”: Indicates the input BSDL file full path and file name. If the file path includes spaces ... **-dev** <Device Name>: Indicates the name of the device (optional). The device name must be the full device name with the performance grade replaced with two X’s, and minus the industry rating or the code name. Example: `LCMXO2-256ZE-XXMG64` (with performance grade replaced with two X's). The device name will default to the device name in the data file. If a device name cannot be found in the file, it will return an error unless the name has been given with the -dev option. **-of** "<Data File Path>": Indicates the output file path. If the file path includes spaces, enclose the path in quotes (optional). Default uses the input file name with *.??? extension. **-convertbidi**: Converts bidirectional I/Os to input or output only, based on the user’s design. Default is to keep al bidirectional I/Os as bidirectional (optional).
-jed2hex	Generates an ASCII Raw Hex file.
-jed2bin	Generates a Binary Raw Hex file.

Examples The following are DDTCMD examples.

Tester Examples:

SVF - Single Device:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -svfsingle -if
"C:/example/lfxp2_05ef256.jed" -dev LFXP2-5E -op "FLASH
Erase,Program,Verify" -of  "C:/ example/
lfxp2_05ef256.svf"
```

SVF - JTAG Chain:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -svfchain -if
"C:/example/demo.xcf" -of  "C:/ example/demo.svf"
```

STAPL - Single Device:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -stpsingle -if
"C:/example/ isppac_powr1220at8.jed" -dev ispPAC-
POWR1220AT8 -op "Erase,Program,Verify" -of  "C:/ example/
isppac_powr1220at8.svf"
```

STAPL - JTAG Chain:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -stpchain -if
"C:/example/demo.xcf" -of  "C:/example/demo.svf"
```

ATE - Generic Vector Format:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -ate -if " C:/
example/demo.xcf " -type tst -of " C:/example/demo.tst"
```

ATE - GenRad:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -ate -if " C:/
example/demo.xcf " -type gr -of " C:/example/demo.gr"
```

ATE - HP3070:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -ate -if " C:/
example/demo.xcf " -type hp3070 -of " C:/example/
demo.pcf"
```

ATE - HP3065:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -ate -if " C:/
example/demo.xcf " -type hp3065 -of " C:/example/
demo.pcf"
```

ATE - Teradyne 1800:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -ate -if " C:/
example/demo.xcf " -type t1800 -of " C:/example/demo.asc"
```

ATE - Teradyne L200/300:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -ate -if " C:/
example/demo.xcf " -type t200 -of " C:/example/demo.asc"
```

Embedded System Examples:

JTAG Full VME Embedded:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -fullvme -if
"C:/example/demo.xcf" -of "C:/example/demo.vme"
```

JTAG Slim VME Embedded:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -slimvme -if
"C:/example/demo.xcf" -ofa "C:/ example/demo_algo.vme" -
ofd "C:/ example/demo_data.vme"
```

Slave SPI Embedded:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -sspi -if "C:/
example/demo.xcf" -ofa "C:/example/demo_algo.sea" -ofd
"C:/example/demo_data.sed"
```

I2C Embedded:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -i2c -if "C:/
example/demo.xcf" -ofa "C:/example/demo_algo.iea" -ofd
"C:/example/demo_data.iea"
```

sysCONFIG (CPU) Embedded - Binary:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -cpu -if "C:/
example/demo.xcf" -type cpu -of "C:/example/
demo_algo.cpu"
```

sysCONFIG (CPU) Embedded - C-Code:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -cpu -if "C:/
example/demo.xcf" -type c -of "C:/example/demo_algo.c"
```

sysCONFIG (CPU) Embedded - Intel Hex:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -cpu -if "C:/
example/demo.xcf" -type hex -of "C:/example/
demo_algo.hex"
```

sysCONFIG (CPU) Embedded - Text (Debug Only):

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -cpu -if "C:/
example/demo.xcf" -type txt -of "C:/example/
demo_algo.txt"
```

External Memory Examples:

Hex Conversion - Intel Hex:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -int -dev LFE2-
6E -if "C:/example/lfec2_06ef256.bit" -of "C:/example/
lfec2_06ef256.mcs"
```

Hex Conversion - Motorola Hex:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -mot -dev LFE2-
6E -if "C:/example/lfec2_06ef256.bit" -of "C:/example/
lfec2_06ef256.exo"
```

Hex Conversion - Extended Tektronix Hex:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -xtek -dev LFE2-
6E -if "C:/example/lfec2_06ef256.bit" -of "C:/example/
lfec2_06ef256.xtek"
```

Dual Boot - Intel Hex:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -boot -dev LFE3-
95E -golden "C:/example/lfec3_095_golden.bit" -primary
"C:/example/lfec3_095.bit" -format int -flashsize 64 -of
"C:/example/lfec3_095_dual_boot.mcs"
```

Dual Boot - Motorola Hex:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -boot -dev LFE3-
95E -golden "C:/example/lfec3_095_golden.bit" -primary
"C:/example/lfec3_095.bit" -format mot -flashsize 64 -of
"C:/example/lfec3_095_dual_boot.exo"
```

Dual Boot - Extended Tektronix Hex:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -boot -dev LFE3-
95E -golden "C:/example/lfec3_095_golden.bit" -primary
"C:/example/lfec3_095.bit" -format xtek -flashsize 64 -of
"C:/example/lfec3_095_dual_boot.xtek"
```

File Conversion Examples:

Bitstream - Binary Bitstream:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -bit -dev LFXP2-
5E -if "C:/example/lfxp2_05ef256.jed" -of "C:/example/
lfxp2_05ef256.bit"
```

Bitstream - ASCII Bitstream:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd" -oft -rbt -dev LFE2-
6E -if "C:/example/lfec2_06ef256.bit" -of "C:/example/
lfec2_06ef256.rbt"
```

JEDEC - Hex and JEDEC - Binary:

```
"C:/lscc/radiant/3.5/bin/nt/ddtcmd -oft <jed2hex|jed2bin>
[-dev <Device Name>] -if "<Data File Path>" [-of "<Data
File Path>"]
```

Return Codes

Code	Definition
0	CMD_SUCCESS
-1	ERROR_LOG_FILE
-2	OUT_MEMORY
-3	CANNOT_WRITE_DIRECTORY
-4	ERROR_CONFIGURATION_SETUP
-5	DEVICE_NAME_NOT_FOUND
-6	FILE_NOT_FOUND
-7	FILE_NOT_VALID
-8	UNKNOWN_OPERATION
-9	TOO_MANY_CHIPS
-10	FILE_NOT_JEDEC
-11	ERROR_OUTPUT_FILE
-12	OPERATION_NOT_SUPPORT
-13	FILE_NAME_ERROR
-14	FILE_READ_ERROR
-15	ERROR_BUILD_SVP_FILE
-16	ERROR_BUILD_BJD_FILE
-17	ERROR_BUILD_SVF_FILE
-18	CANNOT_WRITE_LOG_FILE
-19	MISSING_START_FILE
-20	UNKNOWN_DEVICE
-21	ERRROR_BUILD_BIT_FILE
-22	MULTIPLE_DEVICES_FOUND
-23	DEVICE_NOT_MATCH_FILE
-24	ERROR_COMMAND_LINE_SYNTAX
-25	ERROR_BUILD_ISC_FILE
-26	ERROR_BUILD_SPLIT_BITSTREAM_FILE
-27	ERROR_BUILD_MERGE_BITSTREAM_FILE
-28	ERROR_INPUT_FILE
-29	ERROR_BUILD_BITSTREAM_FILE
-30	ERROR_BUILD_JED_FILE

Code	Definition
-31	ERROR_BUILD_PROM_FILE
-32	DEVICE_NOT_SUPPORTED

See Also

- Command Line Data Flow
- Command Line Program Overview

Running SSPI from the Command Line

You can run sspi_cmd to execute SSPI commands for Nexus and Avant devices from the command line. This tool interacts with the configuration engine via SPI protocol, ideal for testing when the SPI host is unavailable, offering enhanced flexibility and automation.

Subjects included in this topic:

- sspi_cmd Features
- Usage
- sspi_cmd Options
- Example Command Outputs

sspi_cmd Features

- **Detection of Programming Cables** – Lists all detected programming cables connected to the machine.
- **TCK Frequency Adjustment** – Allows you to change the TCK frequency.
- **Target Cable Specification** – Enables you to specify the target cable for running SSPI commands.
- **Command Input** – Accepts SSPI commands as input.
- **Hex String Data Input** – Takes hex string data as input.
- **Output Definition** – Defines the expected number of bytes for the output.
- **Data Output Capture** – Prints the captured data output.
- **Bit File Configuration** – Supports SSPI configuration using bit files.

Usage

sspi_cmd is a standalone executable and can be found in:
<*installer_path*>\programmer\radiant*version_number*\bin\<platform>\sspi_cmd

sspi_cmd Options

The following are sspi_cmd options.

Option	Description
-h	**--help**: Print this help message and exit **--listcables**: Lists all detected programming cables **--list-supported-devices**: Lists all supported devices
-n	**--cable INT [0]**: Programming cable index
-t	**--tck_delay INT [0]**: TCK delay value **--device TEXT**: Device name
-x	**--command TEXT**: SSPI command in hex **--opcode TEXT**: Opcode in hex **--data TEXT**: Input data in hex **--bytes-out INT**: Expected number of output bytes **--nCS_control TEXT:{normal,hold}**: nCS control mode (normal/hold) Control nCS behaviour: 'normal' - Assert before command, deassert after command (default) 'hold' - Assert before command and hold after command
-p	**--program**: Program bitstream mode **--file TEXT**: Path to bit file
-v	**--verbose BOOLEAN**: Enables Verbose Message Print **--print-examples**: Prints usage examples **--version**: Print tool version

Example Command Outputs

- **Lists all detected programming cables**

```
sspi_cmd --listcables
```

- **List all supported devices along with their JTAG ID**

```
sspi_cmd --list-supported-devices
```

- **Specify cable index using --cable, use --listcables to know the indexes**

```
sspi_cmd --cable 0 --device LFMXO5-25 --command 0xE0 --
bytes-out 4
```

- **Send an SSPI command in hex –**

```
sspi_cmd --device LFMXO5-25 --command 0xE0 --bytes-out 4  //
[ Default cable index 0]
sspi_cmd --cable 1 --device LFMXO5-25 --command 0xE0 --
bytes-out 4
```

- **Specify input data for the SSPI command**

```
sspi_cmd --cable 0 --device LFMXO5-25 --command 0xB4 --data
0x12345678
```

- **Specify opcode along with input data for the SSPI command**

```
sspi_cmd --cable 0 --device LFMXO5-25 --command 0xB4 --
opcode 0x00000000 --data 0x12345678
```

- **Program Bitstream file**

```
sspi_cmd --device LFMXO5-25 --program --file "c:/my_design/
cram.bit"
```

- **Use --verbose to print detailed mode**

```
sspi_cmd --device LFMXO5-25 --command 0xE0 --bytes-out 4 --
verbose 1
```

- **Use '--nCS_control hold' to assert chip select before command and do not de-assert after command**

```
sspi_cmd --nCS_control hold --device LFMXO5-25 --command
0xE0 --bytes-out 4
```

TCK_DELAY Details

This option allows you to slow down the TCK clock. It is the same as the **TCK Divider** setting in Programmer. Clock slow down is done by extending the low period of the clock. Valid values for TCK_DELAY are between 0 to 30.

Refer to 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 cables.

Table 20: 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 21: 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 22: USB-2A

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 is fixed at 1.5 MHz, and the Parallel Port frequency is fixed at 500 kHz.

Running stpplayer from the Command Line

stpplayer is a standalone command-line utility for executing STAPL (.stp) files. While tools such as Download Debugger offer a graphical interface for hands-on debugging, stpplayer provides greater flexibility and automation, making it ideal for use in automated test scripts and production environments.

Subjects included in this topic:

- stpplayer Usage
- Key features

- stpplayer Options
- Example Command Outputs

stpplayer Usage

The **stpplayer** executable is located at:

<installer_path>\programmer\radiant\<version_number>\bin\<platform>\stpplayer.exe

Key features

- **Standard File Format Support** – Executes a STAPL (.stp) file in a single device or executes STAPL (.stp) files in multiple devices connected in a single JTAG chain.
- **Automatic Action Detection** – Detects multiple available actions within an input file when only the --infile argument is provided.
- **Selective Action Execution** – Allows you to specify and execute a single action (for example, FC, RMV, RUI, PROGRAM) from a file containing multiple operations.
- **Detection of Programming Cables** – Lists all detected programming cables connected to the machine.
- **TCK Frequency Adjustment** – Allows you to change the TCK divider value for hardware communication.
- **Debug Output** – Enables detailed debug messaging for troubleshooting execution failures.

stpplayer Options

The following table describes the available stpplayer options.

Table 23: stpplayer Options

Option	Description	Usage Type
-i, --if, --infile TEXT ...	Input STAPL file(s) (.stp)	Mandatory
-a, --action TEXT	Action to perform (auto-detected from file if not specified)	Requires --if
-p, --pa, --portaddress TEXT [FTUSB-0]	USB port address (e.g. FTUSB-0)	Requires --if
-c, --ct, --cabletype TEXT [0]	Cable type (USB1 or USB2)	Requires --if
-t, --tck_divider INT:INT in [0-30] [4]	TCK divider value (0-30); higher = slower TCK, more reliable on multi-device chains. Default 4 (~6 MHz on FT2232H)	Requires --if
-d, --debug	Enable debug output	Requires --if
-h, --help	Print help message and exit.	Standalone

Table 23: (Continued)stpplayer Options

Option	Description	Usage Type
-v, --version	Print tool version	Standalone
-e, --print-examples	Print usage examples	Standalone
-l, --list_cables	List all detected programming cables	Standalone

Note:

Standalone options (such as --help, --version, --print-examples, and --list_cables) execute immediately and ignore other options. Any option marked “**Requires --if**” must be used together with --if (--infile). If such an option is specified without --if, the tool exits without performing an operation.

Example Command Outputs

- **Display help topic**

```
stpplayer -h or stpplayer --help
```

- **Display tool version**

```
stpplayer --version
```

- **Display usage examples**

```
stpplayer --print-examples
```

- **List all detected programming cables**

```
stpplayer --list_cables
```

- **Program a single device**

```
stpplayer --if design.stp --action program
```

- **Program with specific Cable type, USB port and TCK divider**

```
stpplayer --if design.stp --action program --cabletype USB2
--portaddress FTUSB-0 --tck_divider 2
```

- **Auto-detect action (first action in STP file is used)**

```
stpplayer --if design.stp
```

- **Enable debug output during programming**

```
stpplayer --if design.stp --action program --debug
```

- **Multiple devices in a JTAG chain**

```
stpplayer --if design1.stp -a program --if device2.stp -a
verify --tck_divider 4
```

The **Embedded STAPL Player Implementation Guide**, which provides detailed instructions for integrating and adapting the player to embedded and custom hardware platforms, is included with the Radiant installation. It is located at:

<installer_path>\programmer\embedded_source\staplembedded\Implementation_Guide.pdf

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 the links below for details.

- Synpwrap
- IP Generation
- IP Packager
- ECO Editor
- Standalone Timing Analyzer

Note

For information on commonly-used FPGA command line tools, see Command Line Basics.

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 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 [**-nolog**] [**-int** <*command_file*>] [**-gui**] [**-int** <*prj_file*> **| -prj** <*project_file*>] [**-notoem**] [**-notpro**] [**-options** <*arguments*>]

Table 24: SYNPWRAP Command Line Options

Option	Description
-options <*arguments*>	Passes all arguments to Synplify/Synplify Pro. Ignores all other options except -notOEM and -notPro. It must be at the end of other options.
-gui	Invokes the Synplify GUI instead of batch running.
-notOEM	Does not use the Lattice OEM version of Synplify/Synplify Pro.
-notPro	Does not use Synplify Pro. It will automatically use Synplify if SynplifyPro does not exist.
-int <*prj_file*>	Same as the -gui and -prj <file> options. Invokes Synplify GUI with project per command file.
-nolog	Does not print out the log file after the process is finished if there is no error (batch mode only).
-prj <*project_file*>	Uses Synplify .prj file instead of the command file (recommended).
-fg	Forces to run in foreground mode, and waits for Synplify to quit. Only available with the -options command.
-h|-help	Prints this usage. Same as running the process without any arguments.

Example

Below shows a synpwrap command line example.

```
synpwrap -gui -prj C:/lscc/radiant/2023.2/sample/sample.prj
```

See Also

- Command Line Program Overview
- Command Line Data Flow

IP Generation

The IP generation command allows you to generate or regenerate any IP in the design.

To see all argument/syntax for the “ipgen” command and their description, use the **--help/-h** syntax in the TCL console.

Command Line Syntax

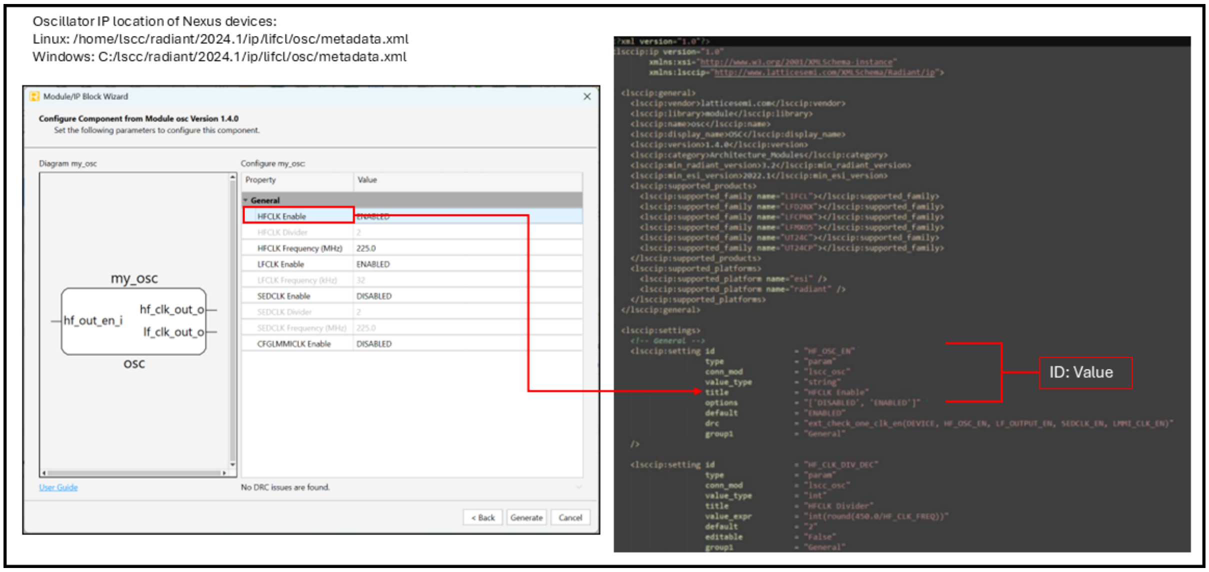
ipgen.EXE [**-h**] [**-o** OUTPUT_DIR] [**-proj_dir** PROJ_DIR] [**-ip** IP_MODULE_DIR] [**-cfg** IP_CONFIG_FILE] [**-skip_uniquify** SKIP_UNIQUIFY_HDL] [**-cfg_value** IP_CONFIG_VALUE] **-name** IP_INSTANCE_NAME [**-platform** IPGEN_PLATFORM] [**-vlnv** IPGEN_VLNV] [**-rtl_library_path** RTL_LIBRARY_PATH] [**-a** ARCHITECTURE] [**-p** DEVICE] [**-t** PACKAGE] [**-sp** PERFORMANCE] [**-f** FAMILY] [**-op** OPERATION]

Table 25: IP Generation Command Line Options

Option	Description
-h, --help	Shows this help message.
-o OUTPUT_DIR	Location where the IP instance package will be generated. Default is the current directory.
-proj_dir PROJ_DIR	Location of the Radiant project, which owns generated IP instance package. Default is the current directory.
-ip IP_MODULE_DIR	Location of the IP module package.
-cfg IP_CONFIG_FILE	Location of the IP configuration file.
-skip_uniquify SKIP_UNIQUIFY_HDL	Controls whether uniquified HDL files are True or False.
-cfg_value IP_CONFIG_VALUE	IP configuration value: 'id:value; id1:value1'
-name IP_INSTANCE_NAME	Name of the IP instance package.
-platform IPGEN_PLATFORM	Name of ipgen's runtime environment: Radiant or Propel. The default is Radiant.
-vlnv IPGEN_VLNV	Location of the IP configuration file by vendor library name and version.
-rtl_library_path RTL_LIBRARY_PATH	The RTL library path for IP.
-a ARCHITECTURE	Target architecture name.
-p DEVICE	Target device name.
-t PACKAGE	Target package name.
-sp PERFORMANCE	Target performance name.
-f FAMILY	Target family name.
-op OPERATION	Specify the device operation.

Hard IP (Module/IP on Local) ID and Value for –cfg_value:

The options for IP configuration using the **–cfg_value <ID:value>** syntax can be found in the metadata.xml file located at: <radiant_install_path>\ip\<IP_name>\metadata.xml.



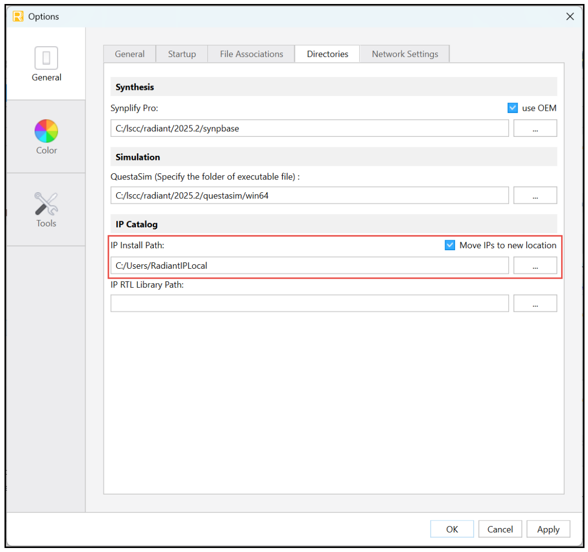
Soft IP (IP on Server) ID and Value for –cfg_value:

The options for IP configuration using the **–cfg_value <ID:value>** syntax can also be found in the metadata.xml file of each IP.

The only difference between Hard IPs and Soft IPs is the location from which Soft IPs are downloaded and installed.

To find the directory where Radiant installs all downloaded IPs:

1. Open Radiant software.
2. Click **Tools > Options**.
3. In the General section, click the **Directories** tab. You will be able to see the IP install path.



Generating a New Hard IP

This section provides an example of how to create a new SDR (Hard IP) for the LFMXO5 device.

ipgen -o **/home/centos/Temp_Proj/Radiant/SDR_IPgen/sdr_rx_new** -ip /**home/centos/lscc/radiant/2024.1/ip/lifcl/sdr** -name **sdr_rx_new** -cfg "**/home/centos/Temp_Proj/Radiant/SDR_IPgen/sdr_rx/sdr_rx.cfg"** -cfg_value **IO_TYPE:LVTTL33** -a **LFMXO5** -p **LFMXO5-100T** - t **BBG400** -sp **9_High-Performance_1.0V** -op **COM**

- **-o** – Directory where IP files are generated.
- **-ip** – Points out where the IP package is. All foundation IP can be found inside the Radiant installation path: (<radiant install path>/lscc/radiant/2024.1/ip/lifcl/sdr).
- **-name** – Name of your IP instance.
- **-cfg (optional)** – Contains all the parameter configuration of the IP. If you do not specify the cfg location, the IP configuration will be default.
- **-cfg_value** – IP configuration value <ID:value>
- **-a** – Device Architecture
- **-p** – Device Name
- **-t** – Device Package
- **-sp** – Device Performance
- **-op** – Operating Condition (Automotive = AUTO, Commercial = COM, Industrial = IND)

Generating a New Soft IP

This section shows an example of how to generate a new SPI Flash Controller IP (Soft IP) instance for the LIFCL device.

ipgen -o **C:/Users/Desktop/Temp_Proj/Radiant/SPI_Flash_Controller/SFC_IP** -ip **C:/lscc/radiant/IP/latticesemi.com_ip_spi_flash_controller_1.5.0 -name** SFC_v1 **-cfg_value PAGE_PROGRAM_SIZE:512** -a **LIFCL** -p **LIFCL-40** -t **QFN72** -sp **7_High-Performance_1.0V** -op **COM**

- **-o** – Directory where all your IP files are generated.
- **-ip** – Points out where the IP is downloaded and installed.
- **-name** – Name of your IP instance
- -**cfg_value** – IP configuration value <ID:value>
- **-a** – Device Architecture
- **-p** – Device Name
- **-t** – Device Package
- **-sp** – Device Performance
- **-op** – Operating Condition (Automotive = AUTO, Commercial = COM, Industrial = IND)

Regenerating an Existing IP

To update the configuration of an IP, use the **-cfg_value** syntax. -cfg_value is used to modify the existing options for specific IP setting ID.

Example of how to change or update the IO_TYPE setting for the SDR IP:

Option 1: Open the CFG file using a text editor and update the parameter value. Then, run the **ipgen** command to regenerate the IP. Ensure that the ipgen syntax correctly references the location of the existing IP.

Change this:

```
1 {
2   "IO_TYPE": "LVTTL33",
3   "BUS_WIDTH": 9,
4   "DATA_PATH_DELAY_INPUT": "Static Default"
5 }
```

To this:

```
{
  "IO_TYPE": "LVDS",
  "BUS_WIDTH": 9,
  "DATA_PATH_DELAY_INPUT": "Static Default"
}
```

Option 2: Modify the value using the cfg_value syntax.

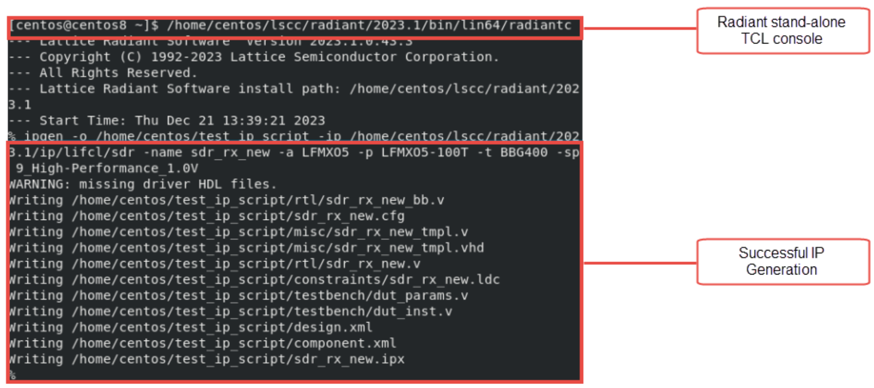
ipgen -o **/home/centos/Temp_Proj/Radiant/SDR_IPgen/sdr_rx_new** -ip **/home/centos/lscc/radiant/2024.1/ip/lifcl/sdr** -name **sdr_rx_new** -cfg “**/home/centos/Temp_Proj/Radiant/SDR_IPgen/sdr_rx/sdr_rx.cfg"** -cfg_value **<u>IO_TYPE:LVDS</u>** -a **LFMXO5** -p **LFMXO5-100T** -t **BBG400** -sp **9_HighPerformance_1.0V** -op **COM**

Using the Radiant TCL Console to Run ipgen:

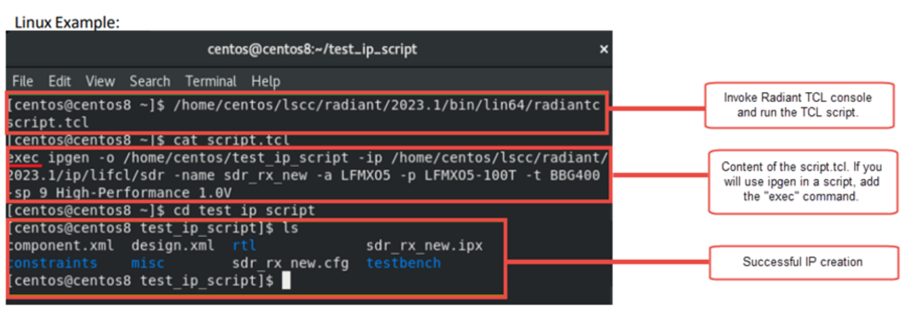
You can execute the **ipgen** command either through the TCL console within Radiant or by launching the stand-alone Radiant TCL console.

On Windows: Run pnmainc.exe inside **<radiant install path>/lscc/radiant/2024.1/bin/lin64/**.

On Linux: Run the Radiant stand-alone TCL console in your terminal: **<radiant install path>/lscc/radiant/2024.1/bin/lin64/radiantc**.



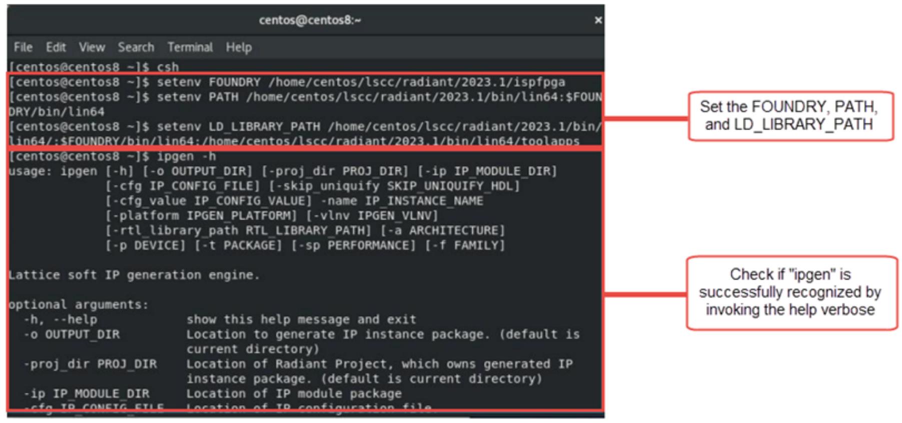
You can pass a script containing the IP generation command directly to the Radiant stand-alone TCL console without launching the console manually. See the example below.



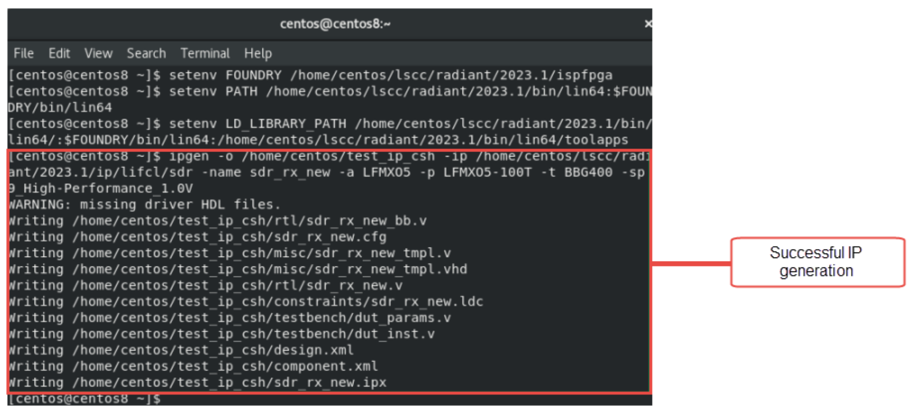
If **ipgen** is used in a script, make sure to include the **exec** command before **ipgen**.

Using CSH Linux Terminal to Invoke ipgen:

To run the **ipgen** command from a CSH terminal in Linux, you must first manually set up the required environment variables. Refer to the example below.

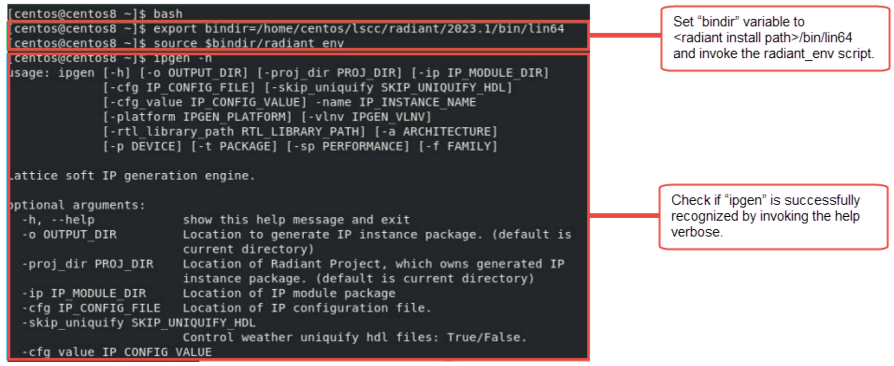


Example of ipgen in CSH terminal:

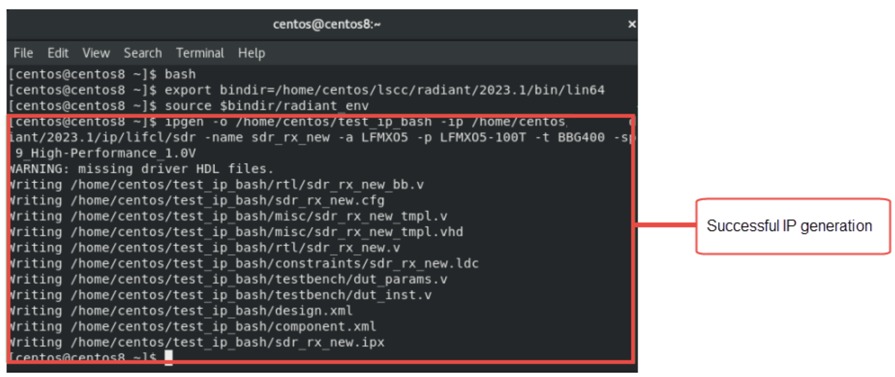


Using BASH Linux Terminal to Invoke ipgen:

To run the **ipgen** command in a bash terminal on Linux, you can set up the environment variables by executing the setup script provided by Radiant.



Example of ipgen in BASH terminal.



See Also

- Command Line Program Overview
- Command Line Data Flow

IP Packager

The IP Packager (ippkg) tool can be run from the command line, allowing IP developers to select files from disks and pack them into one IPK file.

The following describes the process of IP Packager:

- IP author prepares metadata files, RTL files, HTML files, etc (all files of a Soft IP).
- IP Packager GUI provides UI for IP author to select files from the disk, and call IP Packaging engine to pack them into an IPK file.
- IP Packaging engine encrypts RTL files if IEEE P1735-2014 V1 pragmas are specified in RTL source

Command Line Syntax

ippkg [**-h**] (**-metadata** METADATA_FILE | **-metadata_files** METADATA_LIST_NAME) (**-rtl** RTL_FILE | **-rtl_files** RTL_LIST_NAME) [-**ref_rtl** REF_RTL_FILE] [-**encrypt** FORCE_ENCRYPT_FILE | -**encrypt_files** FORCE_ENCRYPT_LIST_NAME] [**-plugin** PLUGIN_FILE] [**-ldc** LDC_FILE] [**-fdc** FDC_FILE] [**-testbench** TESTBENCH_FILE | **-testbench_files** TESTBENCH_LIST_NAME] **-ref_testbench** REF_TESTBENCH_FILE] [-**driver_file** DRIVER_FILE | -**driver_files** DRIVER_LIST_NAME] [-**eval_file** EVAL_FILE | -**eval_files** EVAL_LIST_NAME] (**-help_file** HELP_FILE | **-help_files** HELP_LIST_NAME) [-**license_file** LICENSE_FILE] [**-o** OUTPUT_ZIP_FILE] [**-key_file** KEY_FILE]

Table 26: IPPKG Command Line Options

Option	Description
-metadata	The file name will be fixed to ‘metadata.xml.
-metadata_files	Location of the file which stores the metadata files. One line is a file path in specified file. Must have a file named metadata.xml.
-rtl	Specify the IP RTL file.
-rtl_files	One line is a file path in specified file.
-ref_rtl	Specify the reference IP RTL file.
-encrypt	Encrypt the whole RTL files.
-encrypt_files	One line is a file path in specified file.
-plugin	The file name will be fixed to ‘plugin.py’.
-ldc	Specify the LDC file.
-fdc	Specify the FDC file.
-testbench	Specify the testbench file.
-testbench_files	One line is a file path in specified file.
-ref_testbench	Specify the reference testbench file.
-driver	Specify the driver file.
-driver_files	One line is a file path in specified file.
-eval	Specify the IP evaluation file.
-eval_files	One line is a file path in specified file.
-help_file	Specify the help file, must be <path>/introduction.html.
-help_files	One line is a file path in specified file.
-license_file	Specify the license file.

Table 26: IPPKG Command Line Options

Option	Description
-o	Specify the output zip file.
-key_file	Specify the key file to encrypt the RTL files.

Example

The following is an ippkg command line example:

```
ippkg -metadata c:/test/test.xml -rtl_files c:/test/rtl_list -
help_file c:/test/introduction.html
```

See Also

- Command Line Program Overview
- Command Line Data Flow

ECO Editor

The ECO Editor tool can be run from the command line.

ECO Editor is also able to dump the ECO Tcl commands which user performed in GUI view without saving any UDB file.

In the meanwhile, we will have one non-GUI ECO engine tool, which accepts the dumped Tcl script file with a UDB file and outputs a new UDB file.

User can set 'Place & Route design' milestone post-script by Tcl command prj_set_postscript par <eco.tcl>, then Radiant flow runs the ECO Tcl script automatically after running place & route.

You can bypass the ecoc engine by manually loading the UDB file using the des_read_udb command. This method is only supported in Tcl scripting mode.

Command Line Syntax

~~**ecoc** [**-s** *<script_file>*] [**-o** *<output.udb>*] *<input.udb>*]~~

This method is only supported in TCL scripting mode.

Table 27: ECO Editor Command Line Options

Option	Description
-s	ECO Tcl script file.
-o	Output UDB file.
<input.udb>	Input UDB file.

Example

The following is an ecoc command line example:

```
ecoc -s mem.tcl ebr_test_impl_1.udb
```

See Also

- Engineering Change Order Tcl Commands
- Command Line Program Overview
- Command Line Data Flow

Standalone Timing Analyzer

The Standalone Timing Analyzer can be run from the command line.

Command Line Syntax

tavmain <udb file name> [<pdc file name>]

Example

The following is a tavmain command line example:

```
tavmain design.udb design.pdc
```

See Also

- Command Line Program Overview
- Command Line Data Flow

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 Linux 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 Linux 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 Linux-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 Linux 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 Linux 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 Linux 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 Linux 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/console 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.

Information about TCL features and flow are listed below:

- Power Calculator Tcl Commands
  - Power Calculator Tcl Commands
- Simulation Related Commands
  - Simulation Libraries Compilation Tcl Commands
- Timing and Physical Constraints Commands
  - Radiant Software Timing Constraints Tcl Commands
  - Radiant Software Physical Constraints Tcl Commands
- Radiantc TCL Commands
  - Bitstream Generation Tcl Commands
  - Design Tcl Commands
  - Device Tcl Commands
  - Technology Mapping Tcl Commands
  - Placement Tcl Commands
  - Routing Tcl Commands
  - Timing Analysis Tcl Commands
- Other Commands
  - Design Object Tcl Commands
  - Engineering Change Order Tcl Commands

Additional References

If you are unfamiliar with the Tcl language you can get help by visiting the Tcl/tk web site at https://www.tcl.tk. If you already know how to use Tcl, see the Tcl 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.

Launching 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 On Windows, you can interact with the Tcl Console by anyone of the following methods:

- To launch the Radiant software GUI from the Windows Start menu, choose **Start > Lattice Radiant Software (version_number) > Radiant Software**.

  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 Windows Start menu choose **Start > Lattice Radiant Software (version_number) > Tcl Console**.

  A Windows command interpreter will be launched that automatically runs the **Tcl Console**.

- To run the interpreter from the command line, type the following:

  ```
  c:/lscc/radiant/<version_number>/bin/nt64/radiantc
  ```

  The Radiant **Tcl Console** is now available to run.

- To run the interpreter from a Windows PowerShell from the Windows Start menu choose **Start > 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/radiantc
  ```

  The Radiant **Tcl Console** is now available to run.

  The Radiant **Tcl Console** also supports passing command line arguments to scripts, allowing you to execute a script with specific parameters directly from the command line. For example, you can run a script with arguments using the following command:

  ```
  % radiantc commands.tcl arg_a arg_b arg_c
  ```

  The Radiant **TCL Console** does not require any special argument handling inside the TCL file.

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 On 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/<user_name>/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/<user_name>/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.

Running Radiant Tcl

This section describes the following features:

- How to run Radiant Tcl
- Using log/tcl files
- Accessing the Lattice run directory

Tcl supports the following command modes:

- **GUI Mode (radiant)** – you can use the **radiant** command to launch the GUI mode. Radiant's Tcl Console window is still accessible, however it currently only accepts project commands.
- **Pure interactive shell mode (radiantc)** – starts a pure interactive shell mode. If a tcl file is specified in the command line, the run is done in batch mode. Both project and non-project commands are supported by radiantc.
- **Interactive shell/GUI mode (radiantc -gui)** – starts as a shell mode. The **gui_start** command can be used to launch GUI views. When GUI views are closed, it returns to shell mode.

Log and Tcl Files

When using radiantc, the log message is automatically stored in the **radiantc.log.<pid>** file, and the tcl commands are kept in the **radiant.tcl.<pid>** file, where the <pid> is the process id for radiantc run.In the case the file "**radiantc.log.<pid>**" does exist, the file will be renamed to "**radiantc.log.<pid>.bak**". We do the same check for "**radiant.tcl.<pid>** "

The log message from Radiant (GUI) run is still stored in the **automake.log** file as before.

With radiantc mode, you can do following:

- Use the **RAT_LOG_DIR** environment variable to specify the directory where you want to keep the log/tcl files.

  If there is no environment variable, the log/tcl directory is set as *C:\Users\<User dir>\AppData\Roaming\LatticeSemi\DiamondNG\tcl* if you are using NT, and the current directory **./** when using Linux.
- Specify the run name to be used as the prefix of log/tcl file names.

  For example, if you run *radiantc -run mytest*, the log file will be named *mytest.log.<pid>* and the tcl file will be named *mytest.tcl*.<pid>.

Valid Characters in File Name and Project Path

In file names and project paths, use only the valid characters below. Using unsupported characters result in errors.

- Alphabet (A - Z; a - z)
- Number (0 - 9)
- Underscore (_)
- Minus (-)
- Space ( )

Notes:

1. Brackets [ ] in file names are acceptable in Windows but their use is not recommended.
2. For Linux, using the following symbols in file names are acceptable but not recommended:

- Vertical bar (|)
- Backslash (\)
- Greater than sign(>)
- Question mark (?)
- Less than sign (<)
- Colon (:)
- Asterisk (*)
- Quotation mark (")

Lattice Implementation Directory

In non-project flow, the lattice working directory **lattice_workdir** keeps the intermediate files during the run. The directory is created under your implementation directory.

- Under *lattice_workdir*, you can find map, plc, and rte subdirectories which keep the intermediate run results for technology map (map), placement (plc), and routing (rte).
- For example, if you want to find the placement log file *<file_name>.par*, you can find it under the *plc* subdirectory.

Attributes

In Radiant Tcl system, attributes are used for objects, such as system, design, or instance.

For example:

- *des_set_attribute* is used to set design attributes.
- *des_list_attribute* is used to list the current design attributes.

Understanding Design Flows

Radiant Tcl supports both project and non-project flows.

In project flow, you can do the following:

- Create a new project or load a saved project from an existing project file.
- You can then add, delete, or list project files (i.e., RTL, constraint, and strategy setting files). During the whole project flow, you can save it into a project (.rdf) file any time.

For non-project flow:

- With two specific commands, Radiant Tcl may start a non-project flow from an existing project flow. The Radiant database (.udb) file can be read as input to begin the non-project flow, after which the design process can continue.

There are four major differences between project and non-project flows:

1. The project flow design is saved as disk files instead of being stored in memory. You can use different strategies to create different implementations for the same project. On the other hand, design in non-project flow is kept in memory and can be accessed or changed interactively using commands.

   For example, you can use timing analysis commands to report timing results at different paths.

2. In project flow, you need to add project files. The time stamps of these files are checked for any new commands. If there are any changes in the project files, the flow will ask you to rerun the design at the right stages.

   For example, if you change a post-synthesis constraint (.pdc) file, you need to rerun technology mapping. However, if you change any RTL files or the pre-synthesis constraint files, you need to rerun logic synthesis.

   In non-project flow, the input file is a Radiant database (.udb) file, and the time stamp of such file is not checked during the design flow.

3. You cannot skip any design stage from the run commands in non-project flow.

   For example, you need to run placement (plc_run) before you run routing (rte_run) in non-project flow. However, you can use one of the prj_run commands (prj_run_bitstream, prj_run_map, prj_run_par, prj_run_synthesis) in project flow to run the design flow automatically from the current stage to the target stage.

4. Implementation strategies are used in project flow and implementation options in non-project flow.

   Implementation strategy is a subset of implementation options.

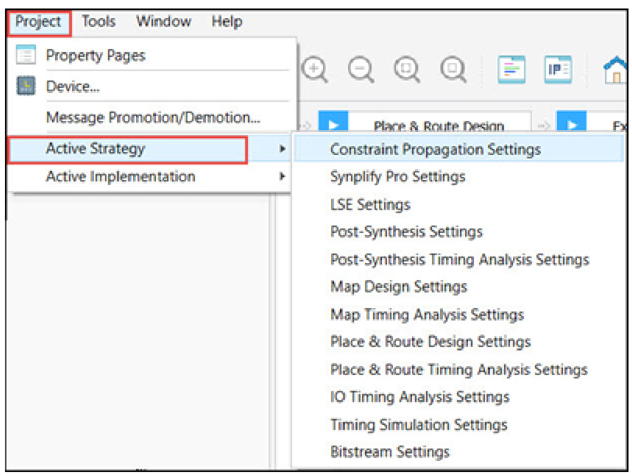
Using Implementation Strategies and Options

In Radiant, **strategies** are used to specify how to run specific tasks, such as logic synthesis, technology mapping, and placement and routing.

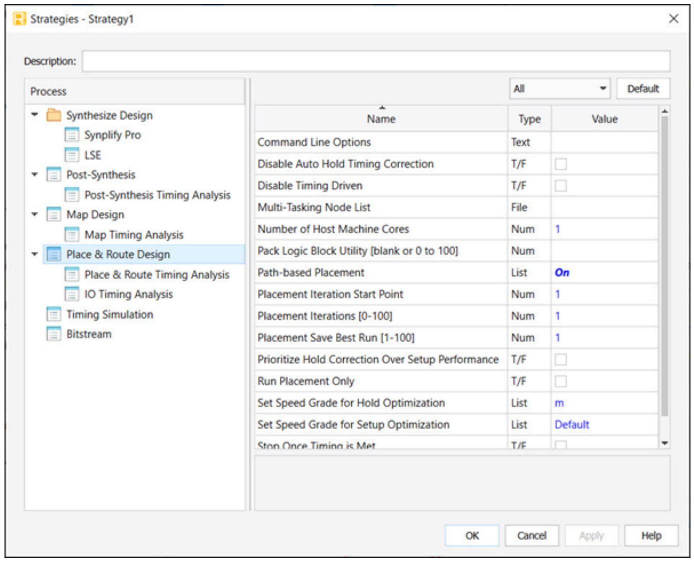
To see the list of strategies in Radiant:

- In the toolbar, click **Project > Active Strategy**.

  The list of strategies appear.



- For instance, if you click **Place & Route Design Settings** from the list, it redirects to the Place & Route Design section in the **Strategies** dialog box.



In TCL, **options** are used to specify how to run specific tasks.

- For example, you can use **par_list_option** after a design has been loaded to check the current placement and routing option setting, and use **par_set_option** to change them:

```
% par_list_option
PAR Options:
     command_line_options                           :
     disable_auto_hold_timing_correction            :
false
     disable_timing_driven                          :
false
     multi_tasking_node_list                        :
     number_of_host_machine_cores                   : 1
     placement_iteration_start_point                : 1
     placement_iterations                           : 1
     placement_save_best_run                        : 1
     prioritize_hold_correction_over_setup_performance :
false
     run_placement_only                             :
false
     set_speed_grade_for_hold_optimization          : m
     set_speed_grade_for_setup_optimization         :
default
     stop_once_timing_is_met                        :
false
```

Strategy and Option Guidelines:

1. Both strategy and option are device-dependent. You can only change them after a device is selected.
2. You do not need to specify all options in the Radiant Strategy Settings. However, all strategies used in Radiant must be converted to options with the following rules:
   - If you are using options, change the space ( ) used in strategy to an underscore (_) to separate words.
   - In strategy, each word is capitalized. You need to change them to the lower case format if you are using options.
   - For example:
     - Strategy uses the following format: **Disable Auto Hold Timing Correction**
     - Option uses the following format: **disable_auto_hold_timing_correction**

Running Project Flow

Running the project flow in Radiant Tcl is the same as the current process. You can use the Radiant software tools to add project files, run the design flow, and output specific timing reports at different design stages.

Most *prj_<command_name>* commands are designed for project flow run except **prj_open_milestone**. This command is used to switch from project flow to non-project flow.

prj_open_milestone

Commands to open project milestone netlist files.

Syntax

```
prj_open_milestone -postsyn | -postmap | -postpar
```

Arguments

- **-postsyn:** Open post-syn logical netlist (optional).
- **-postmap:** Open post-map physical netlist (optional).
- **-postpar:** Open post-par physical netlist (optional). Option '-postsyn' '-postmap' and '-postpar' are mutually exclusive.

Example

```
prj_open_milestone -postsyn
prj_open_milestone -postmap
prj_open_milestone -postpar
```

See Also

▶ "Running Project Flow" on page 278

Running Non-Project Flow

You must load the design from a disk file in order to start a non-project flow. For instance, the following commands load a *mapped.udb* file into memory, report the current design, and run placement.

```
des_read_udb mydb_map.udb
des_report
plc_run
```

Switching From Project Flow to Non-Project Flow

In project flow, you can use **prj_open_milestone** to open the run result at a given design stage and switch to non-project flow.

For instance, you can see the **postsyn** run result after opening the *mydb_syn.rdf* project by using the commands listed below.

The design is currently in memory, and you can report and save it as a Radiant database (.udb) file.

```
prj_open mydb_syn.rdf
prj_open_milestone -postsyn
des_report
des_write_udb -w mydb_syn.udb
```

Using **syn_run** instead of **prj_run_synthesis** for logic synthesis is another way to transition to a non-project flow from a project flow.

- When **syn_run** is used, the tool will first execute logical synthesis using the designated vendor tool before loading the post-synthesis design into memory.
- After that, the flow switches to non-project flow, and you can use non-project commands to complete the rest of the design process.

```
prj_create -name mytest -dev "LIFCL-40-8BG400C" -performance
"8_High-Performance_1.0V" -impl impl_prj_flow -impl_dir
impl_prj_flow -synthesis synplify
prj_add_source ./source/001_register_simple.v
prj_add_source ./others/constraints.sdc
prj_add_source ./others/register_simple1.sty
syn_run
des_report
des_write_udb -w mydb_syn.udb
```

With one of the two methods described above, you can change from project flow to non-project flow. Once the design is in memory (i.e., in non-project flow mode), it is not advised to use project commands.

Running Milestone Results in Non-Project Flow

In non-project flow, the current design is stored in memory, unlike in project flow which keeps previous designs at different milestones. To rerun a previous step due to constraint or option changes, you need to reload the previous milestone result and make a change, then rerun the command.

Note

Milestone refers to processing tasks, which include synthesizing, mapping, placing, and routing a design.

To rerun *par_run* with new par options, you need to do the following:

```
% par_run
% des_write_udb -w mydb_par.udb
% des_reload_milestone -postmap <your_mapped_udb>
% par_set_option (set your par run option)
% par_run
```

To add a new constraint as a post-synthesis constraint, reload the post-syn milestone as the constraints need to be processed through technology mapping.

```
% par_run
% des_write_udb -w mydb_par.udb
% des_reload_milestone -postsyn <your_synthesized_udb>
% ldc_set_location (set your constraint)
% map_run
% par_run
```

To modify timing constraints for logical synthesis, change your constraint file and restart the process from the beginning.

Opening GUI in Non-Project Flow

In order to open Radiant GUI views in non-project flow, you must run **radiantc** with **-gui** option to enable the linking of the GUI libraries during the run. You can open a **post-par udb** and use **gui_start** command to open GUI views.

```
>radiantc -gui
% des_read_udb <your_post_par.udb>
% gui_start
```

Certain GUI views require a post-synthesis netlist as the netlist view, which can be added by incrementally reloading the post-synthesis udb.

```
>radiantc -gui
% des_read_udb <your_post_par.udb>
% des_reload_milestone -inc -postsyn <your_synthesized_udb>
% gui_start
```

Design Flow Examples

1. **Running Pure-Project Flow**

   Tcl commands for pure-project run is shown in this use-case. To run a pure-project, add Register Transfer Level (RTL) files, constraint files then run the project using one of the prj_run commands (prj_run_bitstream, prj_run_map, prj_run_par, prj_run_synthesis) and save the project files at different stages.

   Note

   Pure-project design flow is saved as disk file and not kept in memory. As a result, you cannot run any non-project commands in a project flow. For further information, refer to Opening a Post-Synthesis Milestone.

   Open your console and type the following commands to create, add, run and exit the program:

   - To create a project, use prj_create command followed by choosing the right device and synthesis tool name:

     ```
     prj_create -name mytest -dev "LIFCL-40-8BG400C" -
     performance "8_High-Performance_1.0V" -impl impl_prj_flow
     -impl_dir impl_prj_flow -synthesis synplify
     ```

   - To add project files, use prj_add_source command followed by the path of the file you want to add:

     ```
     prj_add_source ./source/001_register_simple.v
     prj_add_source ./others/constraints.sdc
     prj_add_source ./others/register_simple1.sty
     ```

   - To save an updated project file, use prj_save command:

     ```
     prj_save mydb_syn.rdf
     prj_save new_test.rdf
     ```

```
prj_save mydb_map.rdf
prj_save mydb_par.rdf
```

- To run synthesis project files, use the prj_run_synthesis command.

```
prj_run_synthesis
```

- To end the program, quit the command to end the flow:

```
quit
```

2. Creating Project Files

This section illustrates how to create project files, add Register Transfer Level (RTL) file, and constraint files.

- To create a project, use prj_create command followed by choosing the right device and synthesis tool name:

```
prj_create -name mytest -dev "LIFCL-40-8BG400C" -
performance "8_High-Performance_1.0V" -impl impl_prj_flow
-impl_dir impl_prj_flow -synthesis synplify
```

- To add project files, use prj_add_source command followed by the file directory path you want to add:

```
prj_add_source ./source/001_register_simple.v
prj_add_source ./others/constraints.sdc
prj_add_source ./others/register_simple1.sty
```

- To switch from project flow to non-project flow, use the **syn_run** command:

```
syn_run
```

Use **syn_run** instead of **prj_run_synthesis** to change from project flow to non-project flow. After execution, the design is in memory and you can continue the rest of the flow with non-project commands.

Note

It is not recommended to run the flow with any of the prj_run commands (prj_run_synthesis, prj_run_map, prj_run_par, prj_run_bitstream) once switched to non-project flow.

- To run the design information, use des_report command:

```
des_report
```

- To run the map tool, use map_run command, followed by des_write_udb -w command to save the current design:

```
map_run
des_write_udb -w mydb_map.udb
```

- To run the placement tool and show the report placement information, use plc_run and plc_report commands followed by des_write_udb -w command to save the current design:

```
plc_run
plc_report
des_write_udb -w mydb_plc.udb
```

- To run the routing tool and show the routing result, use rte_run and rte_report commands followed by des_write_udb -w command to save the current design:

```
rte_run
rte_report
des_write_udb -w mydb_rte.udb
```

- To view the detailed report or summary timing, use the following command:

```
sta_report_timing -summary
```

- To generate bitsream file, use bit_generate command followed by -w command to force to overwrite the existing file:

```
bit_generate -w mydb_bit
```

- To end the program, quit the command to end the flow:

```
quit
```

3. Reading Post-Synthesis Design

This section shows you how to read in a design database and continue the design flow.

- To read the post-synthesis database, use des_read_udb command followed by the database file name. To view the report design information, use des_report command:

```
des_read_udb mydb_syn.udb
des_report
```

- To set design attributes, use des_set_attribute command followed by -impl_dir to implement name:

```
des_set_attribute -impl_dir impl_rdb_syn
```

- To run the map tool and to write the design as .udb file, use map_run command followed by des_write_udb and overwrite to save using -w command to completely run the design:

```
map_run
des_write_udb -w myrdbsyn_map.udb
```

- To run the placement tool and show the report placement information, use plc_run command and plc_report command followed by des_write_udb -w command to save the current design:

```
plc_run
plc_report
des_write_udb -w myrdbsyn_plc.udb
```

- To run the routing tool and show the routing result, use rte_run and rte_report commands followed by des_write_udb -w command to save the current design:

```
rte_run
rte_report
des_write_udb -w myrdbsyn_rte.udb
```

- To view the detailed report or summary timing, use sta_report_timing -summary command:

```
sta_report_timing -summary
```

- To generate bitstream file, use bit_generate command followed by -w command to force to overwrite the existing file:

```
bit_generate -w mydb_bit
```

- To end the program, quit the command to end the flow:

```
quit
```

4. **Reading the Post-Map UDB File**

This section shows you how to read a post-map .udb file and continue the design process.

- To read the post-map UDB file, use des_read_udb command followed by the database file name. In the next line, use des_set_attribute command followed by - impl_dir to define the directory where the project file is stored:

```
des_read_udb mydb_map.udb
des_set_attribute -impl_dir impl_udb_map
```

- To run the placement tool and show the report placement information, use plc_run and plc_report commands followed by des_write_udb -w command to save the current design:

```
plc_run
plc_report
des_write_udb -w myudbmap_plc.udb
```

- To end the program, quit the command to end the flow:

```
quit
```

5. **Reading and Routing the Post-Placement Design**

This section describes the process of reading the post-placement database, routing, and saving the design.

- To read the post-synthesis database, use des_read_udb command followed by the database file name. Execute using des_set_attribute command followed by - impl_dir to define the directory where the project file is stored.

```
des_read_udb mydb_plc.udb
des_set_attribute -impl_dir impl_rdb_plc
```

- To run the routing tool and show the routing result, use rte_run and rte_report commands followed by des_write_udb -w command to save the current design.

  Use sta_report_timing - summary command to view the detailed report or timing summary and use des_write_udb command with -w command to overwrite or save the design.

```
rte_run
rte_report
sta_report_timing -summary
des_write_udb -w myrdbplc_rte.udb
```

- To end the program, quit the command to end the flow.

```
quit
```

6. **Opening a Post-Synthesis Milestone**

This section shows how to open a post-synthesis milestone in a project, and load the run result in memory. When the design is in memory, you can continue to use non-project commands to continue the flow and access the design data.

- To open a project, use prj_open command followed by the file name.

```
prj_open mydb_syn.rdf
```

- To open a project milestone, use prj_open_milestone command followed by - postsyn to open the logical netlist.

```
prj_open_milestone -postsyn
```

- To run the map tool and write the design as a .udb file, use map_run command followed by des_write_udb and save using -w command to run the design.

```
map_run
des_write_udb -w myprjsyn_map.udb
```

- To end the program, quit the command to end the flow

```
quit
```

See Also

- Launching the Tcl Console
- Running Radiant Tcl
- Accessing Command Help and Command Options in the Tcl Console
- Creating and Running Custom Tcl Scripts
- Running Tcl Scripts When Launching the Radiant Software
- Radiant Software Tool Tcl Command Syntax
- Tcl Manual

Accessing Command Help and Command Options in the Tcl Console

There are two ways to find all the Tcl command syntax help to generate all the tools in the Tcl console.

To access command syntax help in the Tcl Console:

1. In the prompt, type **help <tool_*keyword*>** and press **Enter:**

   ```
   % help {des_*instance*}
   ```

   A list of valid command options appears in the Tcl Console.

2. In the Tcl Console, type the name of the command or function for more details on syntax and usage.

   For example:

   - des_list_instance is used to list design instances.
   - des_report_instance is used to report instance information.

   A list of valid arguments for that function appears.

   ```
   % help des_report_instance
   Usage information:
       Var/FlagName Type   Value Help
       ------------ ----   ----- ----
       (-help                    gives this help)
       -file        string ()    File name

       ?objects?    string ()    List instances.
   ```

Note

In interactive mode, you can use command **-help** to get the list of command information. However, most commands in the -help option are not true option, and the Tcl interpreter interprets command -help as a usage error, which prevents a script from finishing execution if it encounters this option. It is recommended to use **help {command}** as this works in both interactive and script settings.

Changes in the Interface

The Radiant Tcl has the same Windows settings and appearance as the Radiant Tcl, with some GUI improvements.

- Running the **radiantc** replaces the standalone tools such as Map, PAR, and Bitgen for milestone runs for **Map Design, Place & Route,** and **Export Files** using specific Tcl files, which are stored in the implementation directory as **<prj>_<impl>_map/par/bit.tcl.**
- When you open the milestone run result, the **File > Open** menu displays the **Post-Synthesis Milestone, Post-Map Milestone**, and **Post-Par Milestone**, allowing you to perform non-project commands such as des_report in the new **radiantc** terminal.

See Also

- Launching the Tcl Console
- Running Radiant Tcl
- Accessing Command Help and Command Options in the Tcl Console
- Creating and Running Custom Tcl Scripts
- Running Tcl Scripts When Launching the Radiant Software
- Radiant Software Tool Tcl Command Syntax
- Tcl 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 flexibility 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 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 bar 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 Radiant software Tcl script shows a 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 81: Sample Radiant Software Script

```
prj_archive -dir "C:/my_radiant/counter" -extract "C:/lscc/
radiant/1.1/examples/counter.zip"
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

- Launching the Tcl Console
- Running Radiant Tcl
- Accessing Command Help and Command Options in the Tcl Console
- Creating and Running Custom Tcl Scripts
- Running Tcl Scripts When Launching the Radiant Software
- Radiant Software Tool Tcl Command Syntax
- Tcl 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 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:

  ```
  radiant.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.

```
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 io1.rdf opens, and the MAP process automatically runs.

```
radiant.exe -t c:/test/mytcl.tcl
```

See Also

- Launching the Tcl Console
- Running Radiant Tcl
- Accessing Command Help and Command Options in the Tcl Console
- Creating and Running Custom Tcl Scripts
- Running Tcl Scripts When Launching the Radiant Software
- Radiant Software Tool Tcl Command Syntax
- Tcl Manual

Radiant Software Tool Tcl Command Syntax

This 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. The next sections provide additional information about the Tcl commands implemented in the Radiant software.

The Tcl commands are broken into major categories. The major categories are:

- General Radiant Commands
  - Radiant Software Tcl Console Commands
  - System Tcl Commands
  - Device Tcl Commands
  - Message Control Tcl Commands
- Project Flow Commands
  - Radiant Software Project Tcl Commands
- Non-project Commands
  - Synthesis Tcl Command
  - Place & Route Tcl Commands
  - Physical Synthesis Tcl Commands

General Radiant 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.

The following table provides a listing of all valid Radiant software Tcl Console-related commands.

Table 28: Radiant Software Tcl Console Commands

Command	Description
clear	Erase anything present in the Tcl Console pane and print the current *prompt* character in the upper left corner of the Tcl Console pane without erasing the command history.
history	List the command history in the Tcl Console that you executed in the current session. 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. The command history list is cleared when a project is *opened* or when the Tcl Console *reset* command is executed.
reset	Clear the Tcl Console pane and erase all entries in the command line history. **This command is only used in the GUI Tcl console and not supported in the standalone Tcl console.
save_script	Save 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. Usage: save_script <filename.ext> **This command is only used in the GUI Tcl console and not supported in the standalone Tcl console.
set_prompt	Set the default prompt 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. Usage: set_prompt <new_character> **This command is only used in the GUI Tcl console and not supported in the standalone Tcl console.

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, 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 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
```

System Tcl Commands

The **sys** commands are used to show Radiant tool and system settings information. You can get these commands by using **help {sys_*}**.

Radiant Software System Tcl Command Descriptions

The following table provides a listing of all valid Radiant software system-related Tcl command options and describes option functionality.

Table 29: System Tcl Commands

Command	Description
sys_install_path	Return Radiant executable path.
sys_install_version	Return Radiant version.
sys_list_attribute	List system attributes.
sys_set_attribute	Set system attributes.

sys_install_path

Return Radiant executable path

Syntax

```
sys_install_path
```

Arguments

Example

```
sys_install_path
```

See Also

sys_install_version

- "System Tcl Commands" on page 294

sys_install_version

Return Radiant version

Syntax

```
sys_install_version
```

Arguments

Example

```
sys_install_version
```

See Also

sys_install_path

- "System Tcl Commands" on page 294

sys_list_attribute

List system attributes.

Syntax

```
sys_list_attribute
```

Arguments

Example

```
sys_list_attribute
```

See Also

- `sys_set_attribute`
- "System Tcl Commands" on page 294

sys_set_attribute

Set system attributes.

Syntax

```
sys_set_attribute [-echo <value>] [-reg <value>]
[-gui <value>] [-msg <value>]
```

Arguments

- **-echo <value>:** value must be: {on,off} Allows you to enable or disable the command summary line. (optional)
- **-reg <value>:** value must be: {on,off}. Allows you to output RAT_REG_RECORD_ON or RAT_REG_RECORD_OFF. (optional)
- **-gui <value>:** value must be: {on,off}. The run is called from gui if its value is on. (optional)
- **-msg <filename>:** Set message setting file to <filename>. (optional)

Example

```
sys_set_attribute  -echo on -reg off
```

See Also

- `sys_list_attribute`
- "System Tcl Commands" on page 294

Device Tcl Commands

The **dev** commands are used re used to access or report device information. You can get these commands by using **help {dev_*}**.

The following table provides a listing of all valid Radiant software device-related Tcl command options and describes option functionality.

Table 30: Device Tcl Commands

Command	Description
dev_list_device	List device.
dev_list_family	List family.
dev_list_operation	List operation.
dev_list_package	List package.
dev_list_performance	List performance grade.

dev_list_device

List device

Syntax

```
dev_list_device -family <family> [-p <pattern>]
```

Arguments

- **-family <family>:** Specify the device family name.
- **-p <pattern>:** Specify the device name. Wildcard is supported. If not specified, return all device names under given family. (optional)

Example

```
dev_list_device -family LIFCL
dev_list_device -family LIFCL -p *17
```

See Also

- "Device Tcl Commands" on page 296

dev_list_family

List family

Syntax

```
dev_list_family [-p <pattern>]
```

Arguments

- **-p <pattern>:** Specify the family name. Wild card is supported. If not specified, return all device family names. (optional)

Example

```
dev_list_family
dev_list_family -p LIF*
```

See Also

- "Device Tcl Commands" on page 296

dev_list_operation

List operation

Syntax

```
dev_list_operation -family <family> -device <device> -package
<package> -performance <performance>
```

Arguments

- **-family <family>:** Specify the device family name.
- **-device <device>:** Specify the device name.
- **-package <package>:** Specify the package name.
- **-performance <performance>:** Specify the performance grade.

Example

```
dev_list_operation -family LIFCL -device LIFCL-17 -package
CABGA256 -performance 9_High-Performance_1.0V
```

See Also

- "Device Tcl Commands" on page 296

dev_list_package

List package

Syntax

```
dev_list_package -family <family> -device <device>
[-p <pattern>]
```

Arguments

- **-family <family>:** Specify the device family name.
- **-device <device>:** Specify the device name.

- **-p <pattern>:** Specify the package name. Wild card is supported. If not specified, return all package names under given family and device. (optional)

Example

```
dev_list_package -family LIFCL -device LIFCL-17
dev_list_package -family LIFCL -device LIFCL-17 -p *256
```

See Also

- "Device Tcl Commands" on page 296

dev_list_performance

List performance grade

Syntax

```
dev_list_performance -family <family> -device <device>
-package <package>
```

Arguments

- **-family <family>:** Specify the device family name.
- **-device <device>:** Specify the device name.
- **-package <package>:** Specify the package name.

Example

```
dev_list_performance -family LIFCL -device LIFCL-17
-package CABGA256
```

See Also

- "Device Tcl Commands" on page 296

IP Version Update Tcl Commands

This section provides the IP upgrade version Tcl command syntax, command options, and usage examples.

IP Version Update Command Descriptions

The following table provides a listing of all valid Radiant software IP version update-related Tcl command options and describes option functionality.

Table 31: IP Version Update Commands

Command	Description
ip_catalog_install	Install an IP from local or from server.
ip_catalog_list	List supported IPs in local.
ip_catalog_uninstall	Uninstall an IP from local.
ip_upgrade	Update all .ipx resource files with the new IP version in the Radiant project.

ip_catalog_install

Install one IP from local ipk file or one IP from server.

Syntax

```
ip_catalog_install [-vlnv <IP vlnv>] [-file <ipk file name>]
```

Arguments

- **-vlnv:** IP's information which contains vendor library name and version, use tcl 'ip_catalog_list -server' to get it (optional).

  Format: 'vendor:library:name:version'.
- **-file:** Location of ipk file (optional).

Example

```
ip_catalog_install -vlnv
latticesemi.com:module:sin_cos_table:1.3.0
ip_catalog_install -file D:/ipResource/sin_cos_table.ipk
```

See Also

- "ip_catalog_uninstall" on page 1789

ip_catalog_list

Return a list of IP, local IP contains the IP from Radiant install package, IP server and customer's packaged IP, server IP contains the IP in server. Only get local IP by default.

Syntax

```
ip_catalog_list [-name <IP wild name>]
[-library <IP wild library>][-vendor <IP wild vendor>]
[-version <IP wild version>][-server][-supported][-installed]
```

Arguments

- **-server:** Get IP list in server (optional).
- **-supported:** Only list supported local IPs (optional).
- **-installed:** Only list installed local IPs (optional).
- **-latest:** Only list latest version (optional).
- -**name <IP wild name>:** List IP with specified name (optional).
- **-library <IP wild library>:** List IP with specified library (optional).
- **-vendor <IP wild vendor>:** List IP with specified vendor (optional).
- **-version <IP wild version>:** List IP with specified version (optional).

Example

```
ip_catalog_list -server -name I2C
ip_catalog_list -installed
```

See Also

- "IP Version Update Tcl Commands" on page 300

ip_catalog_uninstall

Uninstall one IP from local IPs, ip_catalog_list can list all installed local IPs.

Syntax

```
ip_catalog_uninstall -vlnv <IP vlnv>
```

Arguments

- **-vlnv:** IP's information which contains vendor library name and version, use tcl 'ip_catalog_list -installed' to get it.

  Format: 'vendor:library:name:version'.

Example

```
ip_catalog_uninstall -vlnv
latticesemi.com:module:sin_cos_table:1.3.0
```

See Also

- “ip_catalog_install” on page 1787

ip_upgrade

Upgrade and regenerate the IP (latest version) in a project.

Syntax

```
ip_upgrade [-all] [-ipx <ipx file path>] [-force_update]
[-version <version number>]
```

Arguments

- **-all:** Upgrade all IPs to the latest version that can be found locally (optional).
- **-ipx <ipx file path>:** Upgrade one IP to the latest version that can be found locally (optional).
- **-version <version number>:** Upgrade one IP to the specified version that can be found locally (optional).
- **-force_update:** Upgrade the ipx file with latest version regardless of whether it is compatible or not, which may cause generation errors (optional).

Example

```
ip_upgrade -ipx D:/radiant_design/counter/adder1.ipx -
force_update
ip_upgrade -ipx D:/radiant_design/counter/adder1.ipx -version
1.3.0
ip_upgrade -all -force_update
```

See Also

- "IP Version Update Tcl Commands" on page 300

Message Control Tcl Commands

This section provides Message Control extended Tcl command syntax, command options, and usage examples.

Message Control Tcl Command Descriptions

The following table provides a listing of all valid Radiant software message control-related Tcl command options and describes option functionality.

Table 32: Message Control Tcl Commands

Command	Description
msg_clean	Clean promoted, disabled and suppressed message(s) from the current project.
msg_clean_setting	Clean the setting of promoted, disabled and suppressed message(s) from the current project.
msg_demote	Reset message promotion type from the current project.
msg_disable	Disable the message from the current project.
msg_list_status	List all promoted, disabled or suppressed messages from the current project.
msg_load	Load the setting of all promoted, disabled and suppressed messages from the file.
msg_promote	Set message promotion type from the current project.
msg_save	Save the setting of all promoted, disabled and suppressed messages to the file.
msg_suppress	Suppress the message from the current project.

msg_clean

Commands to clean promoted, disabled and suppressed message(s) from the current project.

Syntax

```
msg_clean <message Id>|-all
```

Arguments

- **<message Id>**: This option specifies implementation name (optional).
- **-all**: Clean the setting of all promoted, disabled and suppressed messages.

Example

```
msg_clean 1026001
msg_clean -all
```

See Also

- "Message Control Tcl Commands" on page 303

msg_clean_setting

Commands to clean the setting of promoted, disabled and suppressed message(s) from the current project.

Syntax

```
msg_clean_setting <message Id>|-warning|-info|-all
```

Arguments

- **<message Id>**: This option specifies the message id.
- **-warning**: Clean the setting for warning messages.
- **-info**: Clean the setting for info messages.
- **-all**: Clean the setting for all the messages.

Example

```
msg_clean_setting 1026001
msg_clean_setting -warning
msg_clean_setting -info
msg_clean_setting -all
```

See Also

- "Message Control Tcl Commands" on page 303

msg_demote

Commands to reset message promotion type from the current project.

Syntax

```
msg_demote <message Id>
```

Arguments

- **<message Id>**: This option specifies the message id.

Example

```
msg_demote 1026001
```

See Also

- "Message Control Tcl Commands" on page 303

msg_disable

Commands to disable the message from the current project.

Syntax

```
msg_disable <message Id>|-warning|-info
```

Arguments

- **<message Id>**: This option specifies the message id.
- **-warning**: Disable all warning messages.
- **-info**: Disable all info messages.

Example

```
msg_disable 1026001
msg_disable -warning
msg_disable -info
```

See Also

- "Message Control Tcl Commands" on page 303

msg_list_status

Commands to list all promoted, disabled or suppressed messages from the current project.

Syntax

```
msg_list_status [-promoted|-disabled|-suppressed]
```

Arguments

- **-promoted**: List all promoted messages (optional).
- **-disabled**: List all disabled messages (optional).
- **-suppressed**: List all suppressed messages (optional).

Example

```
msg_list_status
msg_list_status -promoted

msg_list_status -disabled
msg_list_status -suppressed
```

See Also

- "Message Control Tcl Commands" on page 303

msg_load

Commands to load the setting of all promoted, disabled and suppressed messages from the file, default is promote.xml.

Syntax

```
msg_load [<message xml or pmt file>]
```

Arguments

- **<message xml or pmt file>**: This option specifies the message xml or pmt file (optional).

Example

```
msg_load
msg_load promote.xml
msg_load promote.pmt
```

See Also

- "Message Control Tcl Commands" on page 303

msg_promote

Commands to set message promotion type from the current project. Use msg_save command after promoting a message to ensure the message promotion settings are saved.

Syntax

```
msg_promote <message Id> [-type 0|1|2|3]
```

Arguments

- **<message Id>:** This option specifies the message id.
- **-type 0|1|2|3:** This option the message promotion type, 0 is error, 1 is critical, 2 is warning, and 3 is info (optional).

Example

```
msg_promote 1026001
msg_promote 1026001 -type 2
```

See Also

- "Message Control Tcl Commands" on page 303

msg_save

Commands to save the setting of all promoted, disabled and suppressed messages to the file, default is promote.xml.

Syntax

```
msg_save [<message xml or pmt file>]
```

Arguments

- **<message xml or pmt file>**: This option specifies the message xml or pmt file (optional).

Example

```
msg_save
msg_save promote.xml
msg_save promote.pmt
```

See Also

- "Message Control Tcl Commands" on page 303

msg_suppress

Commands to suppress the message from the current project, set maximum display count.

Syntax

```
msg_suppress <message Id> -cnt 0|<count>
```

Arguments

- **<message Id>**: This option specifies the message id.
- **-cnt 0|<count>**: This option specifies maximum display count, 0 is disabled, and 'count' means how many times the message will only display.

Example

```
msg_suppress 1026001 -cnt 10
```

See Also

- "Message Control Tcl Commands" on page 303

Project Flow 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 33: Radiant Software Project Tcl Commands

Command	Description
prj_activate_impl	Activate implementation in the current project.
prj_add_source	Add sources to the current project.
prj_archive	Archive the current project.
prj_clean_impl	Clean up the implementation result files in the current project.
prj_clone_impl	Clone an existing implementation.
prj_close	Close the current project. Any unsaved changes are discarded.
prj_copy_strategy	Create a new strategy by copying from an existing strategy.
prj_create	Create a project.
prj_create_impl	Create a new implementation in the current project.
prj_create_strategy	Create a new strategy with default settings.
prj_device	List the device.
prj_disable_source	Disable the design sources from the current project.
prj_enable_source	Enable the design sources from the current project.
prj_get_impl_path	Return active or given implementation folder path.
prj_get_strategy_value	Get value of a strategy item.
prj_import_strategy	Import an existing strategy file.
prj_list_source	List source files in current project.
prj_list_strategy	List value of strategy items.
prj_open	Open a project file.
prj_open_milestone	Open project milestone netlist files.

Table 33: Radiant Software Project Tcl Commands

Command	Description
prj_remove_impl	Delete the specified implementation from the current project.
prj_remove_source	Delete design source from the current project.
prj_remove_strategy	Delete an existing strategy.
prj_run_bitstream	Run bitstream process.
prj_run_map	Run map process.
prj_run_par	Run place & route process.
prj_run_synthesis	Run synthesis process.
prj_save	Save current project.
prj_saveas	Save the current project as a new project with specified name and directory.
prj_set_device	Set the device.
prj_set_impl_opt	List, set or remove implementation options in the current project.
prj_set_opt	List, set or remove a project option.
prj_set_postscript	List or set user Tcl script after running milestone.
prj_set_prescript	List or set user Tcl script before running milestone.
prj_set_reference_udb	Set reference UDB for incremental run or disable it.
prj_set_source_format	Set a source format.
prj_set_source_opt	List, set or remove a source option.
prj_set_strategy	Associate the strategy with the specified implementation.
prj_set_strategy_value	Set value to a strategy item.
prj_set_top_module	Set top module in the current implementation.

This section displays the usage information of the commands:

prj_activate_impl

Activate implementation in the current project.

Syntax

```
prj_activate_impl <impl name>
```

Arguments

- **<impl name>**: Specify implementation name.

Example

```
prj_activate_impl "myimpl"
```

See also

- “Radiant Software Project Tcl Commands” on page 1796

prj_add_source

Add sources to the current project

Syntax

```
prj_add_source [-impl <implement name>]
[-simulate_only|-synthesis_only][-work <VHDL lib name>]
[[-opt <name=value>] ...] [-exclude] <src file>...
```

Arguments

- **-impl <implement name>**: Specify the implementation name. If not specified, use the default implementation (optional).
- **-simulate_only|-synthesis_only**: Indicate new added source can be simulation or synthesis only (optional).
- -**work <VHDL lib name>**: Indicate VHDL library name. Default library name is "work” (optional).
- **-opt <name=value>**: Specify one or multiple option value pair of source file (optional).
- **-exclude**: Indicate the source to be excluded from the project (optional).
- **<src file>**: Specify one or multiple source file paths.

Example

- prj_add_source -synthesis_only "D:/my_design/counter/source/count_attr.v"
- prj_add_source -work mywork D:/my_design/counter/source/my.vhd

See also

- “Radiant Software Project Tcl Commands” on page 1796

prj_archive

Archive the current project or extract the archive file.

Syntax

```
prj_archive [-includeAll] <archive_file>
: Archive the current project into the archive_file
prj_archive -extract -dir <destination directory>
<archive_file>
: Extract the archive file and load the project
```

Arguments

- **-includeAll:** Indicate to include all files under project folder into archive file (optional).
- **<archive_file>:** Specify the archive file (.zip) path.
- **-extract:** Indicate to extract the specified archive file.
- **-dir <destination directory>:** Specify the destination directory to extract archive file. This directory should exist in disk.

Example

```
prj_archive -includeAll "D:/my_desgin/counter/counter.zip"
prj_archive -dir D:/my_design/newcounter -extract
D:/my_design/counter/counter.zip
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_clean_impl

Clean up the implementation result files in the current project.

Syntax

```
prj_clean_impl [-impl <implement name>]
```

Arguments

- **-impl <implement name>:** Specify implementation name. If not specified, use the default implementation. (optional)

Example

```
prj_clean_impl
prj_clean_impl -impl myimpl
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_clone_impl

Clone an existing implementation.

Syntax

```
prj_clone_impl <new impl name> [-dir <new impl directory>]
[-copyRef] [-impl <original impl name>]
```

Arguments

- **-name <new impl name>:** Specify the new implementation name.
- **-dir <new impl directory>:** Specify the new implementation directory (optional).
- **-copyRef:** Indicate to copy all source files from original to new implementation (optional).
- **-impl <original impl name>:** Specify the original implementation name. If not specified, use the default implementation (optional).

Example

```
prj_clone_impl newimpl -copyRef -impl myimpl
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_close

Close the current open project.

Syntax

```
prj_close
```

Example

```
prj_close
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_copy_strategy

Create a new strategy by copying from an existing strategy.

Syntax

```
prj_copy_strategy -from <source strategy name> -name
<new strategy name> -file <strategy file name>
```

Arguments

- **-from <source strategy name>:** Specify the source strategy name**.**
- **-name <new strategy name>:** Specify new strategy name.
- **-file <strategy file name>:** Specify strategy file (.sty) path. This file is used to save new strategy setting.

Example

```
prj_copy_strategy -from Strategy1 -name mystrategy -file
d:/my_desgin/counter/mystrategy.sty
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_create

Create a new project.

Syntax

```
prj_create -name <project name> [-dir <project directory>]
[-dev <device name>] [-performance <performance grade>]
[-impl <initial implementation name>] [-impl_dir <initial
implementation directory>][-synthesis <synthesis tool name>]
```

Arguments

- **-name <project name>:** Specify the project name.
- **-dir <project directory>:** Specify the project directory (optional).
- **-dev <device name>**: Specify the device part name.
- **-performance <performance grade>:** Specify the device performance grade.
- **-impl <initial implementation name>:** Initial implement name, default name is 'impl_1' (optional).
- **-impl_dir <initial implementation directory>:** Implement directory name, default directory name is same as name implementation name (optional).
- **-synthesis <synthesis tool name>:** Synthesis tool type. The valid synthesis name is "synplify" and "lse".

Example

```
prj_create -name "counter" -impl "impl_1" -dev LIFCL-17-9BG256C
-performance "9_High-Performance_1.0V" -synthesis "synplify"
```

```
prj_create -name myproj -dev LIFCL-17-9BG256C -performance
"9_High-Performance_1.0V" -impl myimpl -impl_dir impldir -
synthesis lse
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_create_impl

Create a new implementation in the current project.

Syntax

```
prj_create_impl <new impl name> [-dir <implementation
directory>][-strategy <default strategy name>]
[-synthesis <synthesis tool name>]
```

Arguments

- **-name <new impl name>:** Specify the implementation name.
- **-dir <implementation directory>:** Specify the implementation directory. It's a subdirectory name under project path.
- **-strategy <default strategy name>:** Specify the strategy name (optional).
- **-synthesis <synthesis tool name>:** Synthesis tool type. The valid synthesis name is "synplify" and "lse".

Example

```
prj_create_impl myimpl -strategy Strategy1
prj_create_impl myimpl -dir myimpldir -synthesis "synplify"
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_create_strategy

Create a new strategy with default setting

Syntax

```
prj_create_strategy -name <new strategy name> -file
<strategy file name>
```

Arguments

- **-name <new strategy name>:** Specify new strategy name.

- **-file <strategy file name>:** Specify strategy file (.sty) path. This file is used to save new strategy setting.

Example

```
prj_create_strategy -name mystrategy -file
d:/my_desgin/counter/mystrategy.sty
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_device

List the device information of the current project

Syntax

```
prj_device [-family|-device|-package|-performance|
-operation|-part]
```

Arguments

- **-family:** Return the family name (optional).
- **-device:** Return the device name (optional).
- **-package:** Return the package name (optional).
- **-performance:** Return the performance grade (optional).
- **-operation:** Return the device operation (optional).
- **-part:** Return the device part.

Example

```
prj_device -family -device
prj_device -part
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_disable_source

Disable the design sources from the current project.

Syntax

```
prj_disable_source [-impl <implement name>] <src file> ...
```

Arguments

- **-impl <implement name>:** Specify the implementation name. If not specified, use the default implementation (optional).
- **<src file>:** Specify one or multiple source files to be disabled.

Example

```
prj_disable_source "D:/my_design/counter/source/count_attr.v"
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_enable_source

Enable the design sources from the current project.

Syntax

```
prj_enable_source [-impl <implement name>] <src file> ...
```

Arguments

- **-impl <implement name>:** Specify the implementation name. If not specified, use the default implementation (optional).
- **<src file>:** Specify one or multiple source files to be enabled.

Example

```
prj_enable_source "D:/my_design/counter/source/count_attr.v"
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_get_impl_path

Return active or given implementation folder path

Syntax

```
prj_get_impl_path [-impl <implement>]
```

Arguments

- **-impl <implement name>:** Specify implementation name. If not specified, use the default implementation.

Example:

```
prj_get_impl_path
prj_get_impl_path -impl myimpl
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_get_strategy_value

Get the value of a strategy item.

Syntax

```
prj_get_strategy_value [-strategy <strategy name>]
<option name>
```

Arguments

- **-strategy <strategy name>:** Specify the strategy name. If not specified, use the default strategy (optional).
- **<option name>:** Specify the strategy option name.

Example

```
prj_get_strategy_value -strategy Strategy1 map_ignore_sdc_error
prj_get_strategy_value par_disable_autohldtiming
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_import_strategy

Import an existing strategy file.

Syntax

```
prj_import_strategy -name <new strategy name> -file
<strategy file name>
```

Arguments

- **-name <new strategy name>:** Specify new strategy name.
- **-file <strategy file name>:** Specify import strategy (.sty) path.

Example

```
prj_import_strategy -name mystrategy -file d:/my_desgin/
counter/strategy1.sty
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_list_source

List source files in current project.

Syntax

```
prj_list_source [-o <output_file>]
```

Arguments

- **-o <output_file>:** Specify the output file name. If not specified, list source files in stdout (optional).

Example

```
prj_list_source
prj_list_source -o D:/my_design/counter/source.txt
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_list_strategy

List value of strategy items

Syntax

```
prj_list_strategy [-strategy <strategy name>] <pattern>
```

Arguments

- **-strategy <strategy name>:** Specify the strategy name. If not specified, use the default strategy (optional).
- **<pattern>:** Specify the strategy option name. Wildcard is supported.

Example

```
prj_list_strategy -strategy Strategy1 map_*
prj_list_strategy par_*
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_open

Open a project file.

Syntax

```
prj_open <project file>
```

Arguments

- **<project name>:** Specify project file (.rdf) path.

Example

```
prj_open "d:/my_desgin/counter/counter.rdf"
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_open_milestone

Commands to open project milestone netlist files.

Syntax

```
prj_open_milestone -postsyn | -postmap | -postpar
```

Arguments

- **-postsyn:** Open post-syn logical netlist (optional).
- **-postmap:** Open post-map physical netlist (optional).
- **-postpar:** Open post-par physical netlist.

Option '-postsyn' '-postmap' and '-postpar' are mutually exclusive.

Example

```
prj_open_milestone -postsyn
prj_open_milestone -postmap
prj_open_milestone -postpar
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_remove_impl

Delete the specified implementation from the current project.

Syntax

```
prj_remove_impl <impl name>
```

Arguments

- **<impl name>:** Specify the implementation name.

Example

```
prj_remove_impl "myimpl"
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_remove_source

Delete design source from the current project.

Syntax

```
prj_remove_source [-impl <implement name>] -all
    :Remove all the design sources in project
prj_remove_source [-impl <implement name>] <src file> ...
    :Remove the specified sources from the current project
```

Arguments

- **impl <implement name>:** Specify the implementation name. If not specified, use the default implementation.
- **-all:** Indicate to remove all sources from the current implementation.
- **<src file>**: Specify one or multiple source files to be removed. This option is mutually exclusive with "-all".

Example

```
prj_remove_source -all
prj_remove_source "D:/my_design/counter/source/count_attr.v"
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_remove_strategy

Delete an existing strategy

Syntax

```
prj_remove_strategy <strategy name>
```

Arguments

- **<strategy name>:** Specify the strategy name to be removed.

Example

```
prj_remove_strategy mystrategy
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_run_bitstream

Run bitstream process.

Syntax

```
prj_run_bitstream [-forceAll|forceOne]
```

Arguments

- **-forceAll:** Indicate to force to run all tasks (optional).
- **-forceOne**: Indicate to force to run current task (optional).

Example

```
prj_run_bitstream
prj_run_bitstream -forceAll
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_run_map

Run map process.

Syntax

```
prj_run_map [-forceAll|forceOne]
```

Arguments

- **-forceAll:** Indicate to force to run all tasks (optional).
- **-forceOne**: Indicate to force to run current task (optional).

Example:

```
prj_run_map
prj_run_map -forceAll
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_run_par

Run place and route process.

Syntax

```
prj_run_par [-forceAll|forceOne]
```

Arguments

- **-forceAll:** Indicate to force to run all tasks (optional).
- **-forceOne**: Indicate to force to run current task (optional).

Example

```
prj_run_par
prj_run_par -forceAll
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_run_synthesis

Run synthesis process

Syntax

```
prj_run_synthesis [-forceAll|forceOne]
```

Arguments

- **-forceAll:** Indicate to force to run all tasks (optional).
- **-forceOne**: Indicate to force to run current task (optional).

Example

```
prj_run_synthesis
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_save

Save the current project.

Syntax

```
prj_save [<project file>]
```

Arguments

- **<project file>:** Specify the Radiant project file (.rdf) path to be saved. If not specified, save to current opening project file (optional).

Example

```
prj_save
prj_save d:/my_desgin/counter/mycount.rdf
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_saveas

Save the current project as a new project with specified name and directory and change current workaround of the new project.

Syntax

```
prj_saveas -name <new project name>
-dir <new project directory> [-copy_gen]
```

Arguments

- **-name <new project name>:** Specify the new Radiant project name.
- **-dir <new project directory>:** Specify the new Radiant project file directory.
- **-copy_gen:** Indicate to copy intermediate database files, report files, and other Radiant-generated files into a new project.

Example

```
prj_saveas -name newcounter -dir
d:/my_desgin/newcounter -copy_gen
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_device

Set the device of current project.

Syntax

```
prj_set_device [-family <family name>] [-device <device name>]
[-package <package name>] [-performance <performance grade>]
[-operation <operation>] [-part <part name>]
```

Arguments

- **-family <family name>:** Specify the family name (optional).
- **-device <device name>:** Specify the device name (optional).
- **-package <package name>:** Specify the package name (optional).
- **-performance <performance grade>:** Specify the performance grade (optional).
- **-operation <operation>:** Specify the device operation (optional).
- **-part:** Specify the device part. If '-part' option is specified, other device options are ignored.

Example:

```
prj_set_device -family LIFCL -device LIFCL-17 -package CABGA256
-performance 8_High-Performance_1.0V  -operation COM
prj_set_device -part LIFCL-17-8BG256C
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_impl_opt

List, set, or remove implementation options in the current project.

Syntax

```
prj_set_impl_opt [-impl <implement name>]:
List all the options in the specified implementation
prj_set_impl_opt [-impl <implement name>] <option name> [option
value list]: List or set the implementation's option value
```

```
prj_set_impl_opt [-impl <implement name>] -append <option name>
<option value>: Append a value to the specified option value
prj_set_impl_opt [-impl <implement name>] -rem <option name>...
: Remove the the options in the implementation"
```

Arguments

- **-impl <implement name>:** Specify the implementation name. If not specified, use the default implementation.
- **<option value>:** Specify the option value.
- **<option value list>:** Specify one or multiple option values. If not specified, it lists the option value of given option name.
- **-append <option name> <option value>:** Append the specified value to current option value (optional).
- **-rem <option name>:** Specify one or multiple option names to be removed.

Example

```
# list all the options of the implementation
prj_set_impl_opt -impl impl_1
# set the option myoption value to "value1"
prj_set_impl_opt myoption value1
# list the option myoption value
prj_set_impl_opt myoption
# append "value2" to the option myoption
prj_set_impl_opt -append myoption value2
# remove the option
prj_set_impl_opt -rem myoption
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_opt

List, set or remove a project option

Syntax

```
prj_set_opt: List all the options in the current project
prj_set_opt <option name> [<option value list>]: List or
set the option value
prj_set_opt -append <option name> <option value>: Append
a value to the specified option value
prj_set_opt -rem <option name>: Remove the options of
the current project"
```

Arguments

- **<option name>:** Specify the project option name to be listed, set or removed.
- **<option value>:** Specify the option value.
- **<option value list>:** Specify the one or multiple option values. If not specified, it lists the option value instead (optional).
- **-append <option name> <option value>:** Append the specified value to current option value (optional).
- **-rem <option name>:** Specify one or multiple option names to be removed.

Example

```
# list all the options in the current project
prj_set_opt
# set the option myoption value to "value1"
prj_set_opt myoption value1
# list the option myoption value
prj_set_opt myoption
# append "value2" to the option myoption
prj_set_opt -append myoption value2
# remove the option
prj_set_opt -rem myoption
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_postscript

List or set user Tcl script after running the milestone.

Syntax

```
prj_set_postscript [-impl <implement name>]
<milestone name> <script_file>
```

Arguments

- **-impl <implement name>:** Specify the implementation name. If not specified, use the default implementation (optional).
- **- <milestone name>:** Milestone name should be 'syn', 'map', 'par', 'export'.
- **<script_file>:** User Tcl script file path.

Example

```
prj_set_postscript map my_map.tcl
prj_set_postscript -impl myimpl par my_par.tcl
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_prescript

List or set user Tcl script before running the milestone.

Syntax

```
prj_set_prescript [-impl <implement name>]
<milestone name> <script_file>
```

Arguments

- **-impl <implement name>:** Specify the implementation name. If not specified, use the default implementation (optional).
- **- <milestone name>:** Milestone name should be 'syn', 'map', 'par', 'export'.
- **<script_file>:** User Tcl script file path.

Example

```
prj_set_prescript map my_map.tcl
prj_set_prescript -impl myimpl par my_par.tcl
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_reference_udb

Set reference UDB for incremental run or disable it.

Syntax

```
prj_set_reference_udb {-clean | -lastrun | -postsyn <postsyn
udb> -postmap <posmap udb> -postpar <postpar udb> [-pdc <pdc>]}
```

Arguments

- **-clean:** Disable incremental run. This option is mutually exclusive with others.
- **-lastrun:** Incremental run based on previous run result. This option is mutually exclusive with others.
- **-postsyn <postsyn udb>:** Specify postsyn reference udb (.udb) file path. This option is mutually exclusive with '-clean' & '-lastrun'.
- **-postmap <postsyn udb>:** Specify postmap reference udb (.udb) file path. This option is mutually exclusive with '-clean' & '-lastrun'.

- **-postpar <postsyn udb>:** Specify postpar reference udb (.udb) file path. This option is mutually exclusive with '-clean' & '-lastrun'.
- **-pdc <pdc>:** (Optional) Specify constraint file for map. This option is mutually exclusive with '-clean' & '-lastrun'.

Example

```
prj_set_reference_udb -lastrun
prj_set_reference_udb -postsyn ref_udb/postsy.udb -postmap
ref_udb/postmap.udb -postpar ref_udb/postpar.udb -pdc ref_udb/
user.pdc
prj_set_reference_udb -clean
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_source_format

Set a source format.

Syntax

```
prj_set_source_format -src <source name>
[-impl <implement name>] <format>: Currently,
only Verilog or VHDL is supported"
```

Arguments

- **-impl <implement name>:** Specify the implementation name. If not specified, use the default implementation (optional).
- **-src <source name>:** Specify the source file path to be set.
- **<format>:** Indicate source file format. Only Verilog or VHDL is supported.

Example

```
prj_set_source_format -src "D:/my_design/counter/source/
count_attr.v" Verilog
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_source_opt

List, set, or remove a source option.

Syntax

```
prj_set_source_opt -src <source name> [-impl <implement name>]
: List all the options in the specified source
prj_set_source_opt -src <source name> [-impl <implement name>]
<option name> [option value list]
: List or set the source's option value
prj_set_source_opt -src <source name> [-impl <implement name>]
-append <option name> <option value>
: Append a value to the specified option value
prj_set_source_opt -src <source name> [-impl <implement name>]
-rem <option name>...: Remove the options of the source
```

Arguments

- **-src <source name>:** Specify the source file path to be set.
- **-impl <implement name>:** Specify the implementation name. If not specified, use the default implementation (optional).
- **<option value>:** Specify the option value.
- **<option value list>:** Specify the one or multiple option values. If not specified, it lists the option value instead (optional).
- **-append <option name> <option value>:** Append the specified value to current option value (optional).
- **-rem <option name>:** Specify one or multiple option names to be removed.

Example

```
# list all the options of the source file
prj_set_source_opt -src "D:/my_design/counter/source/
count_attr.v"
# set the option myoption value to "value1"
prj_set_source_opt  -src "D:/my_design/counter/source/
count_attr.v" myoption value1
# list the option myoption value
prj_set_source_opt -src "D:/my_design/counter/source/
count_attr.v" myoption
# append "value2" to the option myoption
prj_set_source_opt -src "D:/my_design/counter/source/
count_attr.v" -append myoption value2
# remove the option
prj_set_source_opt -src "D:/my_design/counter/source/
count_attr.v" -rem myoption
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_strategy

Associate the strategy with the specified implementation.

Syntax

```
prj_set_strategy [-impl <implementation name>] <strategy name>
```

Arguments

- **-impl <implement name>:** Specify the implementation name. If not specified, use the default implementation (optional).
- **<strategy name>:** Specify strategy name associated with implementation.

Example

```
prj_set_strategy -impl myimpl mystrategy
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_strategy_value

Set value to a strategy item

Syntax

```
prj_set_strategy_value [-strategy <strategy name>] <option
name=option value> ...
```

Arguments

- **-strategy <strategy name>:** Specify the strategy name. If not specified, use the default strategy (optional).
- **<option name=option value>:** Specify one or multiple strategy option key value pair.

Example

```
prj_set_strategy_value -strategy Strategy1
map_ignore_sdc_error=True map_infer_gsr=True
prj_set_strategy_value -strategy Strategy1
par_disable_autohldtiming=True par_place_iterator=50
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

prj_set_top_module

Set top module in the current implementation.

Syntax

```
prj_set_top_module <top module>
```

Arguments

- **<top module>:** Specify new top module name.

Example

```
prj_set_top_module counter
```

See Also

- "Radiant Software Project Tcl Commands" on page 1796

Non-project Commands

Non-project Tcl commands allow you to run individual tools and processes without creating or managing a full Radiant project. These commands are useful for quick, script-based operations such as compiling files, running synthesis or place-and-route on the fly, or generating outputs in standalone workflows. The following are key non-project Tcl commands.

- Synthesis Tcl Command
- Place & Route Tcl Commands
- Physical Synthesis Tcl Commands

Synthesis Tcl Command

The **syn** command is used to run the synthesis process. You can change from project flow to a non-project flow by using the **syn_run** command.

Radiant Software Synthesis Tcl Command Description

The following table provides a listing of all valid Radiant software synthesis-related Tcl command options and describes option functionality.

Table 34: Synthesis Tcl Command

Command	Description
syn_run	Run synthesis tool.

syn_run

Run synthesis tool.

Syntax

```
syn_run
```

Arguments

Example

```
syn_run
```

See Also

- "Synthesis Tcl Command" on page 332

Place & Route Tcl Commands

The **par** commands are used to run the place & route process. You can run place & route results from project flow to a non-project flow by using the **par_run** command.

Radiant Software Place & Route Tcl Command Descriptions

The following table provides a listing of all valid Radiant software place & route-related Tcl command options and describes option functionality.

Table 35: Place & Route Tcl Command

Command	Description
par_list_option	List placement routing options.
par_run	Run placement routing.
par_set_option	Set placement routing options.

par_list_option

Lists placement and routing options.

Syntax

```
par_list_option
```

Example

```
par_list_option -optionA
```

See also

- "Place & Route Tcl Commands" on page 333

par_run

Runs placement and routing.

Syntax

```
par_run
```

Example

```
par_run
```

See also

- "Place & Route Tcl Commands" on page 333

par_set_option

Sets placement and routing options.

Syntax

```
par_set_option {<key> <value> ...}
```

Example

```
par_set_option {number_of_host_machine_cores 8
path_based_placement on}
```

See also

- "Place & Route Tcl Commands" on page 333

Physical Synthesis Tcl Commands

The commands in this section are used to run and set physical synthesis options.

Physical Synthesis Tcl Command Descriptions

The following table provides a listing of all valid Radiant software physical synthesis-related Tcl command options and describes option functionality.

Table 36: Placement Tcl Commands

Command	Description
plc_opt	Run post-placement physical synthesis.
plc_opt_list_option	List post-placement physical synthesis options.
plc_opt_set_option	Set post-placement physical synthesis.

plc_opt

Runs post-placement physical synthesis.

Syntax

```
plc_opt
```

Example

```
plc_opt
```

See Also

- "Physical Synthesis Tcl Commands" on page 335

plc_opt_list_option

Lists post-placement physical synthesis options.

Syntax

```
plc_opt_list_option
```

Example

```
plc_opt_list_option
```

See Also

- "Physical Synthesis Tcl Commands" on page 335

plc_opt_set_option

Sets post-placement physical synthesis options.

Syntax

```
plc_opt_set_option {<key value> ... }
```

Example

```
plc_list_option
plc_opt_set_option {sweet_spot_relocation 1
critical_instance_duplication 1}
```

See Also

- "Physical Synthesis Tcl Commands" on page 335

Reveal Commands

Reveal provides a powerful set of Tcl commands that enable automated control of debug flows, from configuring analyzers to capturing and exporting trace data. These commands allow you to script common Reveal operations, integrate debug tasks into larger toolchains, and create repeatable workflows without relying on the graphical interface. The following sections describe the essential Tcl commands available in Reveal.

- Reveal Inserter Tcl Commands
- Reveal Analyzer Tcl Commands

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 Radiant software Reveal Inserter-related Tcl command options and describes option functionality.

Table 37: Reveal Inserter Tcl Commands

Command	Description
rvl_add_controller	Add Controller core.
rvl_add_core	Add Inserter core.
rvl_add_te	Add Inserter trigger expression.
rvl_add_trace	Add Inserter trace signals.
rvl_add_tu	Add Inserter trigger unit.

Table 37: Reveal Inserter Tcl Commands

Command	Description
rvl_close_project	Close Inserter project.
rvl_del_controller	Delete Controller core.
rvl_del_core	Delete Inserter core.
rvl_del_te	Delete Inserter trigger expression.
rvl_del_trace	Delete Inserter trace signal.
rvl_del_tu	Delete inserter trigger unit.
rvl_group_trace	Group Inserter trace signals.
rvl_list_core	List Inserter core.
rvl_list_te	List Inserter trigger expression.
rvl_list_trace	List inserter trace signals.
rvl_list_tu	List Inserter trigger unit.
rvl_move_trace	Move Inserter trace signals.
rvl_new_project	Create inserter project.
rvl_open_project	Open Inserter project.
rvl_rename_core	Rename Inserter core.
rvl_rename_te	Rename Inserter trigger expression.
rvl_rename_trace	Rename Inserter trace bus.
rvl_rename_tu	Rename Inserter trigger unit.
rvl_run_project	Run Inserter project.
rvl_save_project	Save Inserter project.
rvl_set_controller	List or set Controller options.
rvl_set_core	Set Inserter core.
rvl_set_te	Set Inserter trigger expression.
rvl_set_traceoptn	List or set Inserter trace options.
rvl_set_trigoptn	List or set Inserter trigger options.
rvl_set_tu	Set Inserter trigger unit.
rvl_ungroup_trace	Ungroup Inserter trace signals.

rvl_add_controller

Command to add the Controller Core into current project.

Syntax

```
rvl_add_controller
```

Arguments

- **None**

Example

```
rvl_add_controller
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_add_core

Command to add a new core in current project.

Syntax

```
rvl_add_core <core name>
```

Arguments

- **<core name>:** Specify the core name.

Example

```
rvl_add_core top_LA1
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_add_te

Commands to add a new trigger expression to a debug core in current project.

Syntax

```
rvl_add_te [-core <core name>] [-ram EBR|Slice] [-name <new TE
name>] [-expression <expreesion string>]
[-max_seq_depth <max depth>] [-max_event_count
<max event count>]
```

Arguments

- **-core <core name>:** Specify a core name (optional).
- **-ram EBR|Slice:** Specify the RAM type (optional).
- **-name <new TE name>:** Specify the TE name (optional).

- **-expression <expreesion string>:** Set Expression. A default trigger expression name will be created if it is omitted in command.
- **-max_seq_depth <max depth>:** Set the Max Sequence Depth (optional).
- **-max_event_count <max event count>:** Set the Max Event Counter (optional).

Example

```
rvl_add_te
rvl_add_te -name TENew -expression TU1 -core top_LA0
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_add_trace

Commands to add signals in a debug core in current project.

Syntax

```
rvl_add_trace [-core <core name>] [-insert_at <position>]
<signals list>

You can specify an existing trace signal/bus name or a position
number in a trace bus as the inserting position.
```

Arguments

- **-core <core name>**: Specify a core name (optional)
- **-insert_at <position>**: Specify the inserting position (optional)
- **<signals list>**: Specify signals.

Example

```
rvl_add_trace -insert_at leds {clk2}
rvl_add_trace -insert_at leds {clkoe2 clkout1}
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_add_tu

Commands to add a new trigger unit to a debug core in current project.

Syntax

```
rvl_add_tu [-core <core name>] [-radix bin|oct|dec|hex]
[-name <new TU name>] <TU definition>
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **-radix bin|oct|dec|hex**: Specify the Radix (optional).
- **-name <new TU name>**: Specify the TU name (optional).
- **TU definition**: TU definition format: "{signal list} Operator Value" (optional).

  Operator must be "==", "!=", ">", ">=", "<", "<=", ".RE."(rising edge), ".FE."(falling edge) and ".SC."(serial compare).

  A default trigger unit name will be created if it's omitted in command.

Example

```
rvl_add_tu -radix bin
rvl_add_tu -radix bin -name
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_close_project

Commands to close the current reveal insert project.

Syntax

```
rvl_close_project [-force]
```

Arguments

- -**force**: Force to close (optional).

Example

```
rvl_close_project -force
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_del_controller

Commands to remove the Controller Core from current project.

Syntax

```
rvl_del_controller
```

Arguments

- None

Example

```
rvl_del_controller
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_del_core

Commands to remove an existing core from current project.

Syntax

```
rvl_del_core <core name>
```

Arguments

- **<core name>**: Specify the core name.

Example

```
rvl_del_core top_LA1
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_del_te

Commands to delete an existing trigger expression in a debug core in current project.

Syntax

```
rvl_del_te [-core <core name>] [TE name]
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **-name <TE name>:** Specify the TE name (optional).

Example

```
rvl_del_te TE3
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_del_trace

Commands to delete trace signals in a debug core in current project

Syntax

```
rvl_del_trace [-core <core name>] <signals list>
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **<signals list>**: Specify signals.

Example:

```
rvl_del_trace {clk2}
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_del_tu

Commands to delete an existing trigger unit in a debug core in current project.

Syntax

```
rvl_del_tu [-core <core name>] <TU name>
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **<TU name>:** Specify the TU name (optional).

Example

```
rvl_del_tu tuNew
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_group_trace

Commands to group specified trace signals in a debug core in current project into a bus.

Syntax

```
rvl_group_trace [-core <core name>] -bus <bus name> <signals
list>
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **-bus <bus name>**: Specify a bus name.
- **<signals list>**: Specify signals.

Example

```
rvl_group_trace -bus BusNew {clkout1 clkoe2}
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_list_core

Commands to list the default core in current project.

Syntax

```
rvl_list_core
```

Arguments

- **None**

Example

```
rvl_list_core
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_list_te

Commands to list all trigger expressions in a debug core in current project.

Syntax

```
rvl_list_te [-core <core name>]
```

Arguments

- **-core <core name>**: Specify a core name (optional).

Example

```
rvl_list_te
```

See Also

- “Reveal Inserter Tcl Commands” on page 336

rvl_list_trace

Commands to list all trace signals in a debug core in current project

Syntax

```
rvl_list_trace [-core <core name>]
```

Arguments

- **-core <core name>**: Specify a core name (optional).

Example

```
rvl_list_trace
```

See Also

- “Reveal Inserter Tcl Commands” on page 336

rvl_list_tu

Commands to list all trigger unit in a debug core in current project.

Syntax

```
rvl_list_tu [-core <core name>]
```

Arguments

- **-core <core name>**: Specify a core name (optional).

Example

```
rvl_list_tu
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_move_trace

Commands to move and rearrange the order of signals in a debug core in current project.

Syntax

```
rvl_move_trace [-core <core name>] [-move_to <position>]
<signals list>
```

You can specify an existing trace signal/bus name or a position number in a trace bus as the new position.

Arguments

- **-core <core name>**: Specify a core name (optional).
- **-move_to <position>:** Specify the new position (optional).
- **<signals list>**: Specify signals.

Example

```
rvl_move_trace -move_to clkoe2 {clkout1}
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_new_project

Commands to create a new Reveal Inserter project.

Syntax

```
rvl_new_project [-overwrite <new rvl file name>]
[-stage presyn|postsyn|postpar]
```

Arguments

- **-overwrite <new rvl file name>**: Specify project name (optional).
- **-stage presyn|postsyn|postpar:** Specify the Debug Stage.

Example

```
rvl_new_project -overwrite newRvl -stage presyn
rvl_new_project -stage presyn
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_open_project

Commands to open a Reveal Inserter project file.

Syntax

```
rvl_open_project <reveal project file>
```

Arguments

- **<reveal project file>**: Specify the Reveal Inserter project file.

Example

```
rvl_open_project newRvl.rvl
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_rename_core

Commands to rename an existing core in current project.

Syntax

```
rvl_rename_core <core name> <new name>
```

Arguments

- **<core name>:** Specify existing core name.
- **<new name>:** Specify the new core name.

Example

```
rvl_rename_core top_LA1 new_LA
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_rename_te

Commands to rename an existing trigger expression in a debug core in current project.

Syntax

```
rvl_rename_te [-core <core name>] <old name> <new name>
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **<old name>:** Specify the old name.
- **<new name>**: Specify the new name.

Example

```
rvl_rename_te TE2 TENew
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_rename_trace

Commands to change the name of a trace bus in a debug core in current project.

Syntax

```
rvl_rename_trace [-core <core name>] -bus <bus name> <new bus
name>
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **-bus <bus name>:** Specify the existing bus.
- **<new bus name>**: Specify the new bus name.

Example

```
rvl_rename_trace -bus leds newBus
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_rename_tu

Commands to rename an existing trigger unit in a debug core in current project.

Syntax

```
rvl_rename_tu [-core <core name>] <old name> <new name>
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **<old name>:** Specify the old name.
- **<new name>**: Specify the new name.

Example

```
rvl_rename_tu tu1 tu1_new
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_run_project

Commands to run inserting debug core task or DRC checking on the current reveal insert project.

Syntax

```
rvl_run_project [-save] [-saveAs <file_name>] [-overwrite] [-
drc] [-insert_core <core_name>]...
```

Arguments

- **-save**: Save the project before run command (optional).
- **-saveAs:** Save as a different file before run command (optional).
- **-overwrite**: Overwrite the existing file if the saved as to file exists already.
- **-drc:** Run DRC checking only (optional).
- **-insert_core <core_name>**: Specify the core to be inserted.

  All cores will be inserted if none is specified.

Example

```
rvl_run_project
rvl_run_project -drc
rvl_run_project -save
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_save_project

Commands to save the current reveal insert project.

Syntax

```
rvl_save_project [-overwrite <reveal project file>]
```

Arguments

- **-overwrite <reveal project file>**: Specify project name (optional).

Example

```
rvl_save_project newRvl.rvl
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_set_controller

Commands to list or set options of Controller items in current project.

Syntax

```
rvl_set_controller [-item LED|Switch|Register|PLL1|PLL2|...]
      [-set_opt [Insert=on|off] [Width=1|2|3|...|32]]
      [-set_sig [SWn=signal] [LEDn=signal] [Clock=clk_signal]
[Clock_enable=en_signal]
      [Wr_Rdn=wr_rdn_signal] [Address=addr_bus]
[WData=wdata_bus] [RData=rdata_bus]]
```

Arguments

- **-item LED|Switch|Register|PLL1|PLL2|...**: Specify item to be set (optional).
- **-set_opt [Insert=on|off] [Width=1|2|3|...|32]**
  - **Insert=on|off**: Insert the item which specified by -item (optional).
  - **Width=1|2|3|...|32**: Set width for LED and Switch (optional).
- **-set_sig [SWn=signal] [LEDn=signal] [Clock=clk_signal] [Clock_enable=en_signal] [Wr_Rdn=wr_rdn_signal] [Address=addr_bus] [WData=wdata_bus] [RData=rdata_bus]**
  - **SWn=signal**: Specify the signal where n=1 to Width for Switch (optional).

- **LEDn=signal**: Specify the signal where n=1 to Width for LED (optional).
- **Clock=clk_signal**: Specify the signal for Register (optional).
- **Clock_enable=en_signal**: Specify the signal for Clock_enable of Register (optional).
- **Wr_Rdn=wr_rdn_signal for Register**: Specify the signal for Wr_Rdn of Register.
- **Address=addr_bus**: Specify the Address for Wr_Rdn of Register.
- **WData=wdata_bus**: Specify the WData for Register (optional).
- **RData=rdata_bus**: Specify the RData for Register (optional).

Example

```
rvl_set_controller -item PLL1 -set_opt {insert=on}
rvl_set_controller -item Switch -del SW2
rvl_set_controller -item Switch -set_sig {SW0=clkoe1}
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_set_core

Commands to set a core as the default core in current project.

Syntax

```
rvl_set_core [<core name>]
```

Arguments

- **<core name>**: Specify the core name (optional).

Example

```
rvl_set_core
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_set_te

Commands to change an existing trigger expression in a debug core in current project.

Syntax

```
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>
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **-ram <EBR|Slice>**: Specify the RAM type (optional).
- **-expression <expression string>**: Set Expression (optional).
- **-max_seq_depth <max depth>**: Set the Max Sequence Depth (optional).
- **-max_event_count <max event count>**: Set the Max Event Counter (optional).
- **-name <TE name>**: Specify the TE name (optional).

Example

```
rvl_set_te -ram EBR TE
rvl_set_te -expression tu1 TE
rvl_set_te -max_seq_depth 8 TE
rvl_set_te -max_event_count 32 TE
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_set_traceoptn

Commands to list or set trace options of a debug core in current project.

Syntax

```
rvl_set_traceoptn [-core <core name>]
        [SampleClk = <signal name>]
        [SampleEnable = on|off]
        [SampleEnableSig = <signal name>]
        [SampleEnablePri = Active_Low|Active_High]
        [BufferDepth = 16|32|...|65536]
        [Implementation = EBR|DRAM]
        [Pipelining = 0|1|2|3]
        [Timestamp = on|off]
        [TimestampBits = <bits>]
        [CaptureMode = single|multiple]
        [MinimumSamplesPerTrigger = 8|16|32|64|128]
        [IncludeTrigger = on|off]
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **SampleClk = <signal name>**: Specify the Sample Clock (optional).

- **SampleEnable = on|off**: Enable the Sample (optional).
- **SampleEnableSig = <signal name>**: Specify the signal for Sample (optional). (Depends on whether SampleEnable is on.)
- **SampleEnablePri = Active_Low|Active_High**: Specify the mode of Sample (optional). (Depends on whether SampleEnable is on.)
- **BufferDepth = 16|32|...|65536**: Specify the Buffer Depth (optional).
- **Implementation = EBR|DRAM**: Specify the mode of Implementation (optional).
- **Pipelining = 0|1|2|3|**: Specify the depth of Pipelining for trace/trigger signals (optional).
- **Timestamp = on|off**: Enable the Timestamp (optional).
- **TimestampBits = <bits>**: Specify the Bits of Timestamp, which must be in the range from (BufferDepth bits + 1) to 63 (optional). (Depends on whether Timestamp is on and BufferDepth.)
- **CaptureMode = single|multiple**: Specify the Capture Mode (optional).
- **MinimumSamplesPerTrigger = 8|16|32|64|128**: Specify the minimum samples per trigger (optional). (Depends on whether CaptureMode is multiple and BufferDepth)
- **IncludeTrigger = on|off**: Enable to include trigger signals in trace data (optional).

Example

```
rvl_set_traceoptn
rvl_set_traceoptn CaptureMode=multiple
MinimumSamplesPerTrigger=16
rvl_set_traceoptn SampleEnableSig=dn31
rvl_set_traceoptn TimestampBits=63
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_set_trigoptn

Commands to list or set trigger options of a debug core in current project.

Syntax

```
rvl_set_trigoptn [-core <core name>]
        [DefaultRadix = bin|oct|dec|hex]
        [EventCounter = on|off]
        [CounterValue = 2|4|8|16|...|65536]
        [TriggerOut = on|off]
        [OutNetType = IO|NET|BOTH]
        [OutNetName = <net name>]
        [OutNetPri = Active_Low|Active_High]
        [OutNetMPW = <pulse number>]
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **DefaultRadix = bin|oct|dec|hex**: Specify the default Radix (optional).
- **EventCounter = on|off**: Enable the Event Counter (optional).
- **CounterValue = 2|4|8|16|...|65536**: Specify the value of Counter (optional). (depend on EventCounter is on)
- **TriggerOut = on|off**: Enable the Trigger Out (optional).
- **OutNetType = IO|NET|BOTH**: Specify the Net Type (optional). (depend on TriggerOut is on)
- **OutNetName = <net name>**: Specify the Net Name (optional). (depend on TriggerOut is on)
- **OutNetPri = Active_Low|Active_High**: Specify the Polarity mode (optional). (depend on TriggerOut is on)
- **OutNetMPW = <pulse number>**: Specify the Net Type (optional). (depend on TriggerOut is on)

Example

```
rvl_set_trigoptn
rvl_set_trigoptn TriggerOut=on
rvl_set_trigoptn OutNetMPW=1
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_set_tu

Commands to set a trigger unit in a debug core in current project.

Syntax

```
rvl_set_tu [-core <core name>] [-radix bin|oct|dec|hex]
-name <TU name> [-add_sig <signal list>] [-del_sig
<signal list>] [-set_sig <signal list>]
[-expr <TU definition>] [-op <operator>] [-val <value>]
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **-radix bin|oct|dec|hex**: Specify the Radix (optional).
- **-name <TU name>**: Specify the TU name (optional).
- **-add_sig <signal list>**: Add signals (optional).
- **-del_sig <signal list>**: Delete signals (optional).

- **-set_sig <signal list>**: Set signals (optional).
- **-expr <TU definition>**:Define the TU Expression (optional). TU definition format: "{signal list} Operator Value"
- **-op <operator>**: Define the TU Operator (optional). Operator must be "==", "!=", ">", ">=", "<", "<=", ".RE."(rising edge), ".FE."(falling edge) and ".SC."(serial compare)
- **-val <value>**: Set value for Operator (optional).

Example

```
rvl_set_tu -name fff -radix Oct
rvl_set_tu -name fff -op .RE.
rvl_set_tu -name fff -val 2
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

rvl_ungroup_trace

Commands to ungroup trace signals in a trace bus in a debug core in current project.

Syntax

```
rvl_ungroup_trace [-core <core name>] <bus name>
```

Arguments

- **-core <core name>**: Specify a core name (optional).
- **<bus name>**: Specify a bus name.

Example

```
rvl_ungroup_trace BusNew
```

See Also

- "Reveal Inserter Tcl Commands" on page 336

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 Radiant software Reveal Analyzer-related Tcl command options and describes option functionality.

Table 38: Reveal Analyzer Tcl Commands

Command	Description
rva_add_token	Add Analyzer token.
rva_add_tokenset	Add Analyzer token set.
rva_close_controller	Close Controller connection.
rva_close_project	Close Analyzer project.
rva_del_token	Delete Analyzer token.
rva_del_tokenset	Delete Analyzer token set.
rva_export_project	Export Analyzer waveform.
rva_export_tokenset	Export Analyzer token set.
rva_get_trace	List Analyzer trace signals.
rva_import_tokenset	Import Analyzer token set.
rva_manualtrig	Manual Trigger to capture data.
rva_new_project	Create Analyzer project.
rva_open_controller	Open Controller connection.
rva_open_project	Open Analyzer project file.
rva_read_controller	Read data from 32-bit address in hex.
rva_rename_te	Rename Analyzer trigger expression.
rva_rename_tu	Rename Analyzer trigger unit.
rva_run	Run Analyzer project.
rva_run_controller	Run Controller command.
rva_save_project	Save the current Analyzer project.
rva_set_controller	Set Controller options.
rva_set_core	Set Analyzer core.
rva_set_project	Set Analyzer options.
rvl_set_te	Set Analyzer trigger expression.

Table 38: Reveal Analyzer Tcl Commands

Command	Description
rva_set_token	Set Analyzer token.
rva_set_tokenset	Set Analyzer token set.
rva_set_trigoptn	Set Analyzer trigger options.
rva_set_tu	Set Analyzer trigger unit.
rva_stop	Stop Analyzer project without capturing data.
rva_target_controller	Set Controller core as target.
rva_write_controller	Write Controller data to 32-bit address in hex.

rva_add_token

Adds a token with new name and value in a specific token set.

Syntax

```
rva_add_token <tokenset name> <name=value>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_add_tokenset

Adds a token set.

Syntax

```
rva_add_tokenset [-tokenset <tokenset name>]
[-bits <token bits>] [-token <name=value>]
```

Arguments

- **tokenset**: Sets the token set name
- **-bits**: Sets the token set bits
- **-token**: Add extra token

  If no arguments are specified, add a token set with default name and bits.

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_close_controller

Closes Controller connection to Lattice device after read/write has finished.

Syntax

```
rva_close_controller
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_close_project

Closes the current Reveal Analyzer project.

Syntax

```
rva_close_project
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_del_token

Deletes a specific token in a token set.

Syntax

```
rva_del_token <tokenset name> <token name>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_del_tokenset

Deletes the specific token set.

Syntax

```
rva_del_tokenset <tokenset name>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_export_project

Exports waveform in VCD or TEXT from the current Reveal Analyzer project.

Syntax

```
rva_export_project <-vcd|-txt> <file name> [option]
```

Arguments

- **-vcd**: <file name> [module <title>]
- **-txt**: <file name> [-siglist <signal list>]

  By default, the module title is "<unknown>" and all signals are exported.

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_export_tokenset

Exports all token set to a specific file.

Syntax

```
rva_export_tokenset <file name>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_get_trace

Lists all trace signals in Analyzer core.

Syntax

```
rva_get_trace
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_import_tokenset

Imports and merges all token set from a specific file.

Syntax

```
rva_import_tokenset <file name>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_manualtrig

Manually triggers Reveal Analyzer to capture data.

Syntax

```
rva_manualtrig
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_new_project

Creates a new Reveal Analyzer project.

Syntax

```
rva_new_project -rva <file name> -rvl <file name> -dev <val> -
cable <val> -port <val> [-xcf <file name>]
```

Arguments

- **-rva**: Set the new rva file name
- **-rvl**: Set the associated rvl file name
- **-dev**: Set the device name
- **-cable**: Set type of cable as USB or USB2
- **-port**: Set logical port of cable as integer
- **-xcf**: Optional xcf file name for multiple device chain

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_open_controller

Open Controller connection to Lattice device before read/write began.

Syntax

```
rva_open_controller
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_open_project

Opens Controller connection to Lattice device before read/write begins.

Syntax

```
rva_open_project <rva file name>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_read_controller

Reads data from 32bit address in hex.

Syntax

```
rva_read_controller -addr <addr32>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_rename_te

Renames an existing trigger expression in a debug core in current project.

Syntax

```
rva_rename_te <name> <new name>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_rename_tu

Renames an existing trigger unit in a debug core in the current project.

Syntax

```
rva_rename_tu <name> <new name>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_run

Runs Reveal Analyzer until trigger condition to capture data.

Syntax

```
rva_run
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_run_controller

Runs command for Virtual LED, Virtual Switch, User Memory, and Hard IP.

Runs command for User Control Register and User Status Register.

Syntax

```
rva_run_controller -read_led|-read_switch|-write_switch
<data>|-dump_memfile <mem_file>|-load_memfile <mem_file>|
-read_ip <ipname>|-write_ip <ipname>|-eye_monitor <ipname>|
-read_status <regname>|-read_control <regname>|-write_control
<regname> -data <data>|-toggle_control <regname>|
-dump_status <reg_file>|-dump_control <reg_file>|-load_control
<reg_file>
```

Arguments

- **read_led**: Reads data from Virtual LED
- **read_switch**: Reads data from Virtual Switch
- **write_switch**: Writes data to Virtual Switch
- **dump_memfile**: Dumps data from User Memory to mem_file
- **load_memfile**: Loads data from mem_file to User Memory
- **read_ip**: Reads data from Hard IP register of ipname
- **write_ip**: Writes data to Hard IP register of ipname
- **eye_monitor**: Runs Eye-Opening Monitor for PCS channel of ipname

- **mem_calibration**: Run memory calibration of ipname
- **read_status**: Reads data from User Status Register of regname
- **read_control**: Reads data from User Control Register of regname
- **write_control**: Writes data to User Control Register of regname
- **toggle_control**: Toggles data bit in User Control Register of regname
- **dump_status**: Dumps data from User Status Register to reg_file
- **dump_control**: Dumps data from User Control Register to reg_file

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_save_project

Saves the current Reveal Analyzer project.

Syntax

```
rva_save_project [<new rva file name>]
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_set_controller

Sets the options for Controller core.

Syntax

```
rva_set_controller [-option <value>]
```

- **cable_type**: Sets type of cable as USB or USB2.
- **cable_port**: Sets logical port of cable as integer.

  If no arguments are specified, return list of options and values.

```
rva_set_controller -hard_ip <ipname> [-option <value>]
```

- **hard_ip**: Sets options of DPHY1, DPHY2, I2CFIFO1, SGMIICDR1, SGMIICDR2, PCIELL1, PCSCH1 to PCSCH8

  If no option/value specified, then return list of options and values.

Arguments

- **DPHY1, DPHY2 options and values**:
  - **high_speed_select**: Sets value as "1.5Gbps and below", "2.5Gbps"
  - **bypass_pll_clock**: Sets value as Enabled, Disabled

  - **pll_cn_counter**: Sets value as integer from 0 to 31
  - **pll_cm_counter**: Sets value as integer from 0 to 255
  - **pll_co_counter**: Sets value as integer from 0 to 7
- **I2CFIFO1 options and values**:
  - **generic_call_response**: Sets value as Enabled, Disabled
  - **sda_input_delay**: Sets value as Enabled, Disabled
  - **sda_output_delay**: Sets value as Enabled, Disabled
  - **fifo_mode_select**: Sets value as "Register Mode", "FIFO Mode"
  - **fifo_clock_stretching**: Sets value as Enabled, Disabled
  - **fifo_almost_full_value**: Sets value as integer from 0 to 31
  - **fifo_sleep_clock**: Sets value as Enabled, Disabled
  - **slave_address_match_wakeup**: Sets value as Enabled, Disabled
  - **fifo_almost_full_wakeup**: Sets value as Enabled, Disabled
  - **master_read_complete_wakeup**: Sets value as Enabled, Disabled
- **SGMIICDR1, SGMIICDR2 options and values**:
  - **cdr_loss_of_lock**: Sets value as "Lock=+/-1000ppm x2 Unlock=+/-1500ppm x2", "Lock=+/-2000ppm x2 Unlock=+/-2500ppm x2", "Lock=+/-4000ppm Unlock=+/-7000ppm", "Lock=+/-300ppm Unlock=+/-450ppm"
  - **loopback_control**: Sets value as Disabled, "Bit Clock to IO", "Serial Data to IO"
- **PCIELL1 options and values**:
  - **swing_control**: Sets value as full, reduced
- **PCSCH1 to PCSCH8 options and values**:
  - **tx_differential_amplitude**: Sets value as 1000mV, 900mV, 800mV, 700mV, 600mV, 500mV, 400mV, 300mV, 200mV, 100mV, 0mV
  - **tx_output_termination**: Sets value as "100 ohm", "150 ohm"
  - **tx_postcursor_ratio**: Sets value as integer from 0 to 63 of 128th
  - **tx_precursor_ratio**: Sets value as integer from 0 to 63 of 128th
  - **tx_invert_polarity**: Sets value as on, off
  - **rx_input_termination**: Set value as "100 ohm", "150 ohm"
  - **rx_equalization**: Sets value as "RL2plus 8GT/s", "RL2plus 5GT/s", "RL2plus 2.5GT/s", "SS_LMS 8GT/s", "SS_LMS 5GT/s", "SS_LMS 2.5GT/s"
  - **rx_invert_polarity**: Sets value as on, off
  - **loopback_mode**: Set value as Disabled, "PMA Near-End Serial Loopback", "PMA Far-End Parallel Loopback", "8B/10B PCS Near-End Parallel Loopback", "8B/10B PCS Far-End Parallel Loopback", "64B/66B PCS Loopback Path A", "64B/66B PCS Loopback Path B", "64B/66B PCS Loopback Path C"

- **reset_tx_mpcs**: Sets value as on, off
- **reset_rx_mpcs**: Sets value as on, off
- **reset_tx_pll**: Sets value as on, off
- **reset_rx_pll**: Sets value as on, off
- **force_tx_driver**: Sets value as on, off
- **force_rx_driver**: Sets value as on, off
- **eye_quality**: Sets quality of eye diagram as high, normal, low

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_set_core

Sets a core as the default core in the current project.

Syntax

```
rva_set_core [-name <name>] [-run <on|off>]
```

Arguments

- **name**: Selects core. This is needed for other actions.
- **run**: Turns the run option on or off for core.

  If no arguments are specified, return list of core.

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_set_project

Sets the current Reveal Analyzer project options.

Syntax

```
rva_set_project [-frequency <val>] [-period <val>] [-tckdelay
<val>] [-cabletype <val>] [-cableport <val>]
```

Arguments

- **frequency**: Sets the frequency value for sample clockin MHz
- **period**: Sets the period value for sample clock in ns or ps
- **tckdelay**: Sets the TCK Divider Setting upto 30

- **cabletype**: Sets the type of cable. Values must be "LATTICE", "USB", "USB2"
- **cableport**: Sets the port number as integer >=0

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_set_te

Changes an existing trigger expression in a debug core in current project.

Syntax

```
rva_set_te [-name <name>] [-expression <expression list>] [-
enable <on|off>]
```

Arguments

- **name**: Selects TE. If no options are specified, return options and values for the selected TE.
- **expression**: Sets TE expression
- **enable**: Enables/disables TE

  If no arguments are specified, return list of TE.

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_set_token

Selects a specific token in a specific token set.

Syntax

```
rva_set_token <tokenset name> <token name> -name <new token
name> -value <new token value>
```

Arguments

- **name**: Sets token name
- **value**: Sets token value

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_set_tokenset

Selects a specific token set.

Syntax

```
rva_set_tokenset  <tokenset name> [-name <new token set name>]
[-bits <new token bits>]
```

Arguments

- **name**: Renames a token set
- **bits**: Sets number of bits in token

See Also

- “Reveal Analyzer Tcl Commands” on page 355

rva_set_trigoptn

Sets trigger options of Analyzer core in the current project.

Syntax

```
rva_set_trigoptn  [-teall <AND|OR>] [-finalcounter <on|off>]
[-finalcountervalue <val>][-samples <val>]
[-numtriggers <val>] [-position <pre|center|post|val>]
```

Arguments

- **teall**: Sets AND ALL or OR ALL for all TE
- **finalcounter**: Turns final trigger counter on/off
- **finalcountervalue**: Sets final trigger counter value
- **samples**: Sets number of samples to capture
- **numtriggers**: Sets number of triggers to capture
- **position**: Sets trigger position to pre-selected or user value

  If no arguments specified, then return list of options.

See Also

- “Reveal Analyzer Tcl Commands” on page 355

rva_set_tu

Sets a trigger unit in a debug core in current project.

Syntax

```
rva_set_tu  [-name <name>] [-operator {== | != | > | >= | < |
<= | "rising edge" | "falling edge"}][-value <value>] [-radix
{bin | oct | dec | hex | <token>}]
```

Arguments

- **name**: Selects TU. If no options are specified, return options and value for the selected TU.
- **operator**: Sets TU comparison operator
- **value**: Sets TU value
- **radix**: Sets TU radix

  If no arguments are specified, return list of TU.

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_stop

Stops Reveal Analyzer without capturing data.

Syntax

```
rva_stop
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_target_controller

Sets Controller core as target before read/write begins.

Syntax

```
rva_target_controller
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

rva_write_controller

Writes data to 32bit address in hex.

Syntax

```
rva_write_controller -addr <addr32> -data <data>
```

See Also

- "Reveal Analyzer Tcl Commands" on page 355

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 Radiant software Power Calculator-related Tcl command options and describes option functionality.

Table 39: Power Calculator Tcl Commands

Command	Description
pwc_add_inactiveimpl	Add inactive implementation name.
pwc_add_ipblock	Add IP Block row.
pwc_calculate	Run power calculation for current project.
pwc_close_project	Close the current project.
pwc_gen_csvreport	Generate CSV report and write to file.
pwc_gen_htmlreport	Generate HTML report and write to file.
pwc_gen_report	Generate text report and write to file.
pwc_new_project	Create a new project.
pwc_open_project	Open a project file.
pwc_remove_ipblock	Remove IP Block row.
pwc_save_project	Save the current project.
pwc_set_af	Set activity factor.
pwc_set_afpervcd	Open vcd file and set frequency and activity factor.
pwc_set_ambienttemp	Set ambient temperature value.
pwc_set_device	Set device information.
pwc_set_estimation	Set estimated routing option.
pwc_set_freq	Set frequency.

Table 39: Power Calculator Tcl Commands

Command	Description
pwc_set_ipblock	Set IP Block row.
pwc_set_junctiontemp	Set junction temperature value.
pwc_set_powerscreen	Set power screen type.
pwc_set_processtype	Set device power process type.
pwc_set_supply	Set multiplication factor and voltage of named power supply.
pwc_set_thetaja	Set thermal profile.
pwc_set_userenteredtj	Set user entered Tj value.

pwc_add_inactiveimpl

Commands to add inactive implementation name.

Syntax

```
pwc_add_inactiveimpl [<implementation name>]
```

Arguments

- **<implementation name>**: This option specifies implementation name.

Example

```
pwc_add_inactiveimpl impl1
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_add_ipblock

Commands to add IP Block row.

Syntax

```
pwc_add_ipblock -iptype <iptype name>
```

Arguments

- **iptype <iptype name>**: This option specifies IP type.

Example

```
pwc_add_ipblock -iptype Logic
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_calculate

Commands to run power calculation for current project.

Syntax

```
pwc_calculate
```

Example

```
pwc_calculate
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_close_project

Commands to close the current project.

Syntax

```
pwc_close_project
```

Example

```
pwc_close_project
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_gen_csvreport

Commands to generate CSV report and write to file.

Syntax

```
pwc_gen_csvreport <report file>
```

Arguments

- **<report file>**: This option specifies the report file name.

Example

```
pwc_gen_csvreport test.csv
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_gen_htmlreport

Commands to generate HTML report and write to file.

Syntax

```
pwc_gen_htmlreport <report file>
```

Arguments

- **<report file>**: This option specifies the report file name.

Example

```
pwc_gen_htmlreport test.html
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_gen_report

Commands to generate text report and write to file.

Syntax

```
pwc_gen_report <report file>
```

Arguments

- **<report file>**: This option specifies the report file name.

Example

```
pwc_gen_report test.html
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_new_project

Commands to create a new project.

Syntax

```
pwc_new_project <project file>
```

Arguments

- **<project file>**: This option specify new project file name.

Examples

```
pwc_new_project test.pcf
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_open_project

Commands to open a project file.

Syntax

```
pwc_open_project <project file>
```

Arguments

- **<project file>**: This option specifies the project file name.

Example

```
pwc_open_project test.pcf
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_remove_ipblock

Commands to remove IP Block row.

Syntax

```
pwc_remove_ipblock -iptype <iptype name> -matchkeys {<key1>
<value1>}+
```

Arguments

- **iptype <iptype name>**: This option specifies IP type.
- **matchkeys {<key1> <value1>}+**: This option specifies match keys.

Example

```
pwc_remove_ipblock -iptype Logic -matchkeys Clk clk_1
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_save_project

Commands to save the current project.

Syntax

```
pwc_save_project <project file>
```

Arguments

- **<project_file>**: This option specifies the project file name.

Example

```
pwc_save_project test.pcf
```

See Also

- “Power Calculator Tcl Commands” on page 368

pwc_set_af

Commands to set activity factor.

Syntax

```
pwc_set_af <activity factor> [-keepClocksAF]|<activity factor>
-iptype <iptype name> -clk <clock name>
```

Arguments

- **<activity factor>**: This option specifies default activity factor.
- **-keepClocksAF**: This option specifies activity factor except clocks tab (optional).
- **-iptype <iptype name>**: This option is to set activity factor for given IP.
- **-clk <clock name>**: This option is to set activity factor for given IP clock name.

Example

```
pwc_set_af 20
pwc_set_af 30 -keepClocksAF
pwc_set_af 40 -iptype Logic -clk clk_1
```

See Also

- “Power Calculator Tcl Commands” on page 368

pwc_set_afpervcd

Commands to open vcd file and set frequency and activity factor.

Syntax

```
pwc_set_afpervcd -vcd <vcd file> [-module <module name>]
```

Arguments

- **-vcd <vcd file>**: This option specifies the vcd file name.
- **-module <module name>**: This option specifies the module name (optional).

Example

```
pwc_set_afpervcd -vcf test.vcd
pwc_set_afpervcd -vcf test.vcd -module abc
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_ambienttemp

Commands to set ambient temperature value.

Syntax

```
pwc_set_ambienttemp <ambient temperature>
```

Arguments

- **<ambient temperature>**: This option specifies implementation name (optional).

Example

```
pwc_set_ambienttemp 30
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_device

Commands to set device information.

Syntax

```
pwc_set_device -family <family name>|-device <device name>|-
package <package name>|-speed <speed name>|-operating
<operating name>|-part <part name>
```

Arguments

- **-family <family name>**: This option specifies family name.
- **-device <device name>**: This option specifies device name.
- **-package <package name>**: This option specifies package name.
- **-speed <speed name>**: This option specifies speed name.
- **-operating <operating name>**: This option specifies operating name.
- **-part <part name>**: This option specifies part name.

Example

```
pwc_set_device -family LIFCL
pwc_set_device -device LIFCL-17
pwc_set_device -package QFN72
pwc_set_device -speed 8_Low-Power_1.0V
pwc_set_device -operating Commercial
pwc_set_device -part LIFCL-17-8SG72Cpwc_add_inactiveimpl impl1
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_estimation

Commands to set estimated routing option.

Syntax

```
pwc_set_estimation <estimated routing option>
```

Arguments

- <estimated routing option>: This option specifies estimated routing type.

Example

```
pwc_set_estimation Low
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_freq

Commands to set frequency.

Syntax

```
pwc_set_freq <frequency>|<clock> <frequency>|-freq <frequency>
-usefreqtwr 0|1 -freqtwropt min|pref|trace
```

Arguments

- **<frequency>**: This option specifies default frequency.
- **<clock> <frequency>**: This option specifies clock frequency.
- **-freq <frequency>**: This option specifies frequency.
- **-usefreqtwr 0|1**: This option specifies if it is to set frequency by importing timing report or not.
- **-freqtwropt min|pref|trace**: This option specifies the selection for importing timing report.

Example

```
pwc_set_freq 100
pwc_set_freq clk_1 100
pwc_set_freq -freq 100 -usefreqtwr 1 -freqtwropt min
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_ipblock

Commands to set IP Block row.

Syntax

```
pwc_set_ipblock -iptype <iptype name> -matchkeys {<key1>
<value1>}+ -setkey <key> <value>
```

Arguments

- -iptype <iptype name>: This option specifies ip type.
- -matchkeys {<key1> <value1>}+: This option specifies match keys.
- -setkey <key> <value>: This option specifies key value.

Example

```
pwc_set_ipblock -iptype Logic -matchkeys Clk clk_1 -setkey LUT
5000
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_junctiontemp

Commands to set junction temperature value.

Syntax

```
pwc_set_junctiontemp <junction temperature>
```

Arguments

- **<junction temperature>**: This option specifies the junction temperature value.

Example

```
pwc_set_junctiontemp 60
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_powerscreen

Commands to set power screen type.

Syntax

```
pwc_set_powerscreen <power screen>
```

Arguments

- **<power screen>**: This option specifies power screen value.

Example

```
pwc_set_powerscreen P20
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_processtype

Commands to set device power process type.

Syntax

```
pwc_set_processtype <process type>
```

Arguments

- **<process type>** This option specifies process type.

Example

```
pwc_set_processtype Typical
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_supply

Commands to set multiplication factor and voltage of named power supply.

Syntax

```
pwc_set_supply -type <type> -voltage <voltage> -dpm <dpm>
```

Arguments

- **-type <type>:** This option specifies supply type.
- **-voltage <voltage>:** This option specifies voltage value.
- **-dpm <dpm>:** This option specifies dpm value.

Example

```
pwc_set_supply -type Vcc -voltage 0.8 -dpm 1.5
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_thetaja

Commands to set thermal profile.

Syntax

```
pwc_set_thetaja <theta JA>|-board <board type> -heatsink
<heatsink value>|-board <board type> -heatsink <heatsink type>
-airflow <airflow type>
```

Arguments

- **<theta JA>**: This option specifies user defined theta JA.
- **-board <board type>**: This option specifies board type.
- **-heatsink <heatsink value>**: This option specifies custom profile heat sink value.
- **-heatsink <heatsink type>**: This option specifies heat sink type.
- **-airflow <airflow type>**: This option specifies airflow type.

Example

```
pwc_set_thetaja 20
pwc_set_thetaja -board Medium Board -heatsink 10
pwc_set_thetaja -board Medium Board -heatsink No Heat Sink -
airflow 0 LFM
```

See Also

- "Power Calculator Tcl Commands" on page 368

pwc_set_userenteredtj

Commands to set user entered Tj value.

Syntax

```
pwc_set_userenteredtj 1|0
```

Arguments

- **1**: This option specifies user entered Tj.
- **0**: This option specifies auto-computed Tj.

Example

```
pwc_set_userenteredtj 1
```

See Also

- "Power Calculator Tcl Commands" on page 368

Simulation Libraries Compilation Tcl Commands

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 Radiant software Simulation Libraries Compilation-related Tcl command options and describes option functionality.

Note

Running cmpl_libs may take a long time and may cause the Radiant software to hang.

- It is recommended to run cmpl_libs using the Radiant Tcl Console (**Start Menu > Lattice Radiant Software > Accessories > TCL Console**).

or,

- Run cmpl_libs.tcl using the command line console. Refer to Running cmpl_libs.tcl from the Command Line.

Table 40: Simulation Libraries Compilation Tcl Command

Command	Description
cmpl_libs	Compile simulation library.
sim_generate_script	Generate a simulation script for use with ModelSim or QuestaSim.

This section displays the usage information of the commands:

cmpl_libs

The **-sim_path** argument specifies the path to the simulation tool executable (binary) folder. This option is mandatory. Currently, only QuestaSim simulators from Mentor and Xcelium simulator from Cadence 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.
- To execute this script error free, Siemens Questa 2020.4 or a later version should be used for compilation.

The -sim_vendor argument specifies simulation vendor. This argument is optional. If you do not use this argument, Mentor is chosen by default, and the libraries are compiled for QuestaSim. If you are using QuestaSim choose mentor and Xcelium choose cadence for -sim_vendor.

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.

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.

Note

- 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.
- 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.

```
% cmpl_libs
Usage: cmpl_libs -sim_path <sim_path> [-sim_vendor
{siemens<default>|synopsys|cadence}] [-device
{iCE40UP|lifcl|lfcpnx|lfcpnx_rcs|lfd2nx|lfd2nx1|lfd2nx2|lfmxo5|
lfmxo5t|lfmxo5t1|lav_ate_es|lav_atg_es|lav_atx_es|lav_ate_b|ln2
_mh|ln2_ct|lav_ate|lav_atg|lav_atx|ut24c|ut24cp|all<default>}]
[-target_path <target_path>]
```

sim_generate_script

The sim_generate_script command is used to generate the QuestaSim qrun argument (.f) template file from the current Radiant project settings, including RTL, test bench, and other files. If there is no open Radiant project, this command will fail to run.

This command does not support modifying the original file list of the Radiant project. Simulation Wizard does not call this command to generate the qrun argument file. If you want to modify simulation files, you can edit the qrun argument file manually.

Note

The sim_generate_script command does not launch QuestaSim as the Simulation Wizard. However, as long as QuestaSim executes the command correctly, it can generate any type of file as an extension of the script.

Syntax

```
sim_generate_script -top <top module> [-qrun <true|false>]
[-stage <rtl|postsyn|postroute|timing>] [-inst <instance name>]
[-debug] [-run <timesteps><time_units>|0]
[-t <multiplier><time_unit>] [-no_gui] [-w] -o <do file>
```

Arguments

- **-top**: top module for simulation.
- **-qrun**: invoke QuestaSim with Qrun (optional).
- **-stage**: valid types are 'rtl', 'postsyn', 'postroute', 'timing'. 'rtl' is default (optional).
- **-inst**: specify instance for the associated SDF file (optional).
- **-debug**: apply full debug mode to the simulator (optional).
- **-run**: specify the number of timesteps for the simulation to run (optional). valid <time_unit> must be one of the fs, ps, ns, us, ms, sec. '0' means 'run -all'.
- **-t**: specify the simulator time resolution (optional). <multiplier> is optional. <time_unit> must be one of the fs, ps, ns, us, ms, sec.
- **-no_gui**: runs scripted batch simulations (optional).
- **-w**: allow to overwrite existed simulation do file or arguments file (optional).
- **-o**: specify simulation do file path or arguments file path.

Example

```
sim_generate_script -top test -run 0 -w -o test.f
sim_generate_script -top count -stage rtl -o count.f
```

See Also

- "Simulation Libraries Compilation Tcl Commands" on page 381

Simulation Libraries Compilation Tcl Command Example

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 QuestaSim is specified by -sim_path.

```
cmpl_libs -sim_path C:/questasim64_10.4e/win64 -target_path c:/
mti_libs
```

Timing and Physical Constraints Commands

Radiant Software Timing Constraints Tcl Commands

The section describes the commands are used to set timing constraints.

Radiant Software Timing Constraints Tcl Command Descriptions

The following table provides a listing of all valid Radiant software timing constraints-related Tcl command options and describes option functionality.

Table 41: Radiant Software Timing Constraints Tcl Commands

Command	Description
create_clock	Create a named or virtual clock.
create_generated_clock	Create a generated clock object.
set_clock_groups	Set clock groups.
set_clock_latency	Define a clock's source or network latency.
set_clock_uncertainty	Set clock uncertainty.
set_false_path	Define false path.
set_input_delay	Set input delay on ports.
set_load	Set capacitance on ports.
set_max_delay	Specify maximum delay for timing paths.
set_max_skew	Define maximum skew of path.
set_min_delay	Specify minimum delay for timing paths.
set_multicycle_path	Define multicycle path.
set_output_delay	Set output delay on ports.
set_hierarchy_separator	Set or get hierarchy separator.

create_clock

Commands to create a new clock object and define its characteristics.

Syntax

```
create_clock -period <period_value> [-name <clock_name>]
    [-waveform {<value1 value2>}] [<source_object>] [-add]
```

Arguments

- **-period period_value:** This value 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.
- **-name clock_name:** This 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 (optional).
- **-waveform {value1 value2}:** These 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 value1 must also be less than the clock period (optional).
- **-source_object:** This value is the object on which the clock constraint is defined. The source object can be a port, pin or a net object in the design. If you specify a clock constraint on a source object that already has a clock, the new clock replaces the existing one (optional).
- **-add:** This option allows defining another clock on an object that already has a clock (optional).

Example 1

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 2

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]
```

See Also

- "Radiant Software Timing Constraints Tcl Commands" on page 384

create_generated_clock

Commands to create an internally generated clock object and define its characteristics.

Syntax

```
create_generated_clock
```

```
[-name <clock_name>] -source <source_object>
[-edges {<value1 value2 value3>}]
[-edge_shift {<value1 value2 value3>]
[-divide_by <factor>]
[-multiply_by <factor>]
[-duty_cycle <percent>]
[-invert]
[-add]
[-master_clock <clock>] <object_list>
```

Arguments

- **-name clock_name:** This name string specifies the name of the generated clock (optional).
- **-source source_object:** This value is an object on which the source clock of the generated clock is defined. The source object can be a port, pin or a net object. The period of the generated clock is derived from the clock on the source object using the multiply and divide factors.
- **-edges {value1 value2 value3}:** These values are a list of three numbers which correspond to the edge of the source clock from which the generated clock has been obtained. This option cannot be used with '-divide_by', '-multiply_by' and '-duty_cycle' (optional).
- **-edge_shift {value1 value2 value3}:** These values are a list of three numbers that represents the amount of shift (optional).
- **-divide_by factor:** This value specifies the frequency division factor. Default value is 1. 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 (optional).
- **-multiply_by factor:** This value specifies the frequency multiplication number to be used when finding the generated clock frequency. Default value is 1 (optional).
- **-duty_cycle factor:** 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. This option cannot be used with '-divide_by' (optional).
- **-invert:** This option inverts the clock edge (optional).
- **-add:** This option allows defining another clock on an object that already has a generated clock.
- **-master_clock clock:** This value specifies the master clock for the generated clock object if there are multiple clocks found on the source object (optional).
- **-object_list:** This value is a list of clock source pins,ports,or nets.

Example 1

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]
```

Example 2

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]
```

Example 3

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

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_clock_groups

Commands to specify 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 (-logically_exclusive | -physically_exclusive
| -asynchronous) [-group <clock_objects>] [-group
<clock_objects>]
```

Arguments

- **-logically_exclusive:** This option specifies that clocks are mutually exclusive and will not reach the flip flops at the same time due to logic implementation (optional).
- **-physically_exclusive:** This option specifies that clocks are mutually exclusive. Only one clock group is active at any given time (optional).
- **-asynchronous:** This option specifies that the clock groups are asynchronous to each other. Two clocks are asynchronous with respect to each other if they have no phase relationship at all. One option among '-logically_exclusive', '-physically_exclusive' and '-physically_exclusive' is needed and they are mutually exclusive (optional).
- **-group clock_objects:** This value specifies the clock objects in a group. At least one '-group' is needed. Each group of clocks is exclusive or asynchronous with the clocks which are specified in the other group. Specifying one group indicates that the clocks in this group are exclusive or asynchronous with all other clocks in the design including one clock which is created in the future (optional).

Example 1

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]
```

Example 2

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 -logically_exclusive -group [get_clocks
{clka_port clkb_port}] -group [get_clocks clkc_port]
```

See Also

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_clock_latency

Commands to define a clock's source or network latency.

Syntax

```
set_clock_latency [-rise] [-fall] [-early | -late] -source
<latency> <object_list>
```

Arguments

- **-rise**: This option specifies the latency for the rising clock edge (optional).
- **-fall:** This option specifies the latency for the falling clock edge (optional).
- **-early|-late:** This option specifies how early|late the clock edge arrives. Option '-early' and '-late' are mutually exclusive (optional).
- **-source:** This option indicates the specified latency as a source latency.
- **<latency>:** This value specifies the value of latency.
- **object_list>:** This value is a list of clocks. You can also use the keyword '[all_clocks]' to include all clocks.

Example

In the following example, 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_latency 3 -source -early [get_clocks clk1]
set_clock_latency 4 -source -late [all_clocks]
```

See Also

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_clock_uncertainty

Commands to indicate that the clock of interest has uncertainties in its period.

Syntax

```
set_clock_uncertainty [-setup] [-hold] [-from <clock>] [-to
<clock>] <uncertainty> [<clock_list>]
```

Arguments

- **-setup:** This option indicates the constraint for setup analysis (optional).
- **-hold:** This option indicates the constraint for hold analysis (optional).
- **-from clock:** This value specifies uncertainty source clock and the timer requires the 'from' option to be accompanied by a 'to' option (optional).
- **-to clock:** This value specifies uncertainty destination clock (optional).
- **<uncertainty>:** This value specifies the amount by which the clock period in uncertain.
- **<clock_list>:** This value is a list of clocks. You can also use the keyword '[all_clocks]' to include all clocks. One option between '-from' and 'clock_list' is needed and they are mutually exclusive (optional).

Examples

The following two examples specify clock uncertainty, one with a from/to. 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_clock_uncertainty 0.5 [get_clocks clk1]
set_clock_uncertainty 0.4 -from [get_clocks clk2] -to
[get_clocks clk3]
```

See Also

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_false_path

Commands to identify paths that are considered false and excluded from timing analysis.

Syntax

```
set_false_path
[-from <port_list|pin_list|instance_list|clock_list>]
[-to <port_list|pin_list|instance_list|clock_list>]
[-through <port_list|pin_list|instance_list|net_list>]
[-rise_from <clock_list>] [-rise_to <clock_list>]
[-fall_from <clock_list>] [-fall_to <clock_list>]
[-setup] [-hold]
```

Arguments

- **-from port_list|pin_list|instance_list|clock_list:** This value specifies the timing path start points (optional).
- **-to port_list|pin_list|instance_list|clock_list:** This value specifies the timing path end points (optional).
- **-through object_list:** This value is a list of through points which the paths should be blocked (optional).
- **-rise_from|fall_from clock_list:** Same as '-from' but only applies to clock objects. Option '-from', 'rise_from' and '-fall_from' are mutually exclusive.
- **-rise_to|fall_to object_list clock_list:** Same as '-to' but only applies to clock objects. Option '-to', 'rise_to' and '-fall_to' are mutually exclusive. At least one option among '-from|-rise_from|-fall_from|-to|-rise_to|-fall_to|-through' is needed.
- **-setup:** This option indicates to eliminate setup timing analysis for specified timing paths (optional).
- **-hold:** This option indicates to eliminate hold timing analysis for specified timing paths (optional).

Example 1

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]
```

Example 2

The following example specifies all paths through the net UO/sigA as false:

```
set_false_path -through [get_nets UO/sigA]
```

See Also

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_input_delay

Commands to define the arrival time of an input relative to a clock.

Syntax

```
set_input_delay -clock <clock_object> [-clock_fall] [-max |-
min] [-add_delay] <delay_value> <port_list>
```

Arguments

- **-clock clock_object:** This value specifies the clock reference to which the specified input delay is related.
- **-clock_fall:** This option is a switch to specify the falling edge of clock to trigger (optional).
- **-max:** This option specifies that the delay value is the maximum delay (optional).
- **-min:** This option specifies that the delay value is the minimum delay. Option '-max' and '-min' are mutually exclusive (optional).
- **add_delay:** This option is used to define more than one input delay on a given port (optional).
- **<delay_value>:** This 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.
- **<port_list>:** This value 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.

Example 1

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_input_delay 4.0 -clock [get_clocks vclk] -add_delay
[get_ports out2]
```

Example 2

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

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_load

Commands to set capacitance on ports.

Syntax

```
set_load <capacitance> <objects>
```

Arguments

- **<capacitance>:** This value specifies the load in capacitance in pF.
- **<objects>:** This value provides one or more output ports in the current design to which capacitance is assigned. You can also use the keyword '[all_outputs]' to include all output ports.

Example

The following example sets a capacitive load of 25 pF for output port OUT:

```
set_load 25 [get_ports {out}]
```

See Also

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_max_delay

Commands to specifies the maximum delay for the timing paths.

Syntax

```
set_max_delay
[-from <port_list|pin_list|instance_list|clock_list>]
[-to <port_list|pin_list|instance_list|clock_list>]
[-through <port_list|pin_list|instance_list|net_list>]
[-datapath_only]
[-rise_from <clock_list>]
[-rise_to <clock_list>]
[-fall_from <clock_list>]
[-fall_to <clock_list>] <delay_value>
```

Arguments

- **-from port_list|pin_list|instance_list|clock_list:** This value specifies the timing path start points (optional).
- **-to port_list|pin_list|instance_list|clock_list:** This value specifies the timing path end points (optional).
- **-through object_list:** This value specifies the timing path pass through points. The timing paths must go through these points (optional).
- **-rise_from|fall_from clock_list:** Same as '-from' but only applies to clock objects. Option '-from', 'rise_from' and '-fall_from' are mutually exclusive.
- **-rise_to|fall_to clock_list:** Same as '-to' but only applies to clock objects. Option '-to', 'rise_to' and '-fall_to' are mutually exclusive. At least one option among '-from|-rise_from|-fall_from|-to|-rise_to|-fall_to|-through' is needed.
- **-datapath_only:** This option calculates the Clock-to-Q delay of the first flop, the total delay between the flops, and the setup time of the second flop (optional).
- **<delay_value>:** This value specifies a floating point number in nanoseconds that represents the required maximum delay value for specified paths.

Example

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 <port_list|pin_list|instance_list|clock_list>]
[-to <port_list|pin_list|instance_list|clock_list>]
[-through <port_list|pin_list|instance_list|net_list>]
[-datapath_only]
[-rise_from <clock_list>]
[-rise_to <clock_list>]
[-fall_from <clock_list>]
[-fall_to <clock_list>] <delay_value>
```

See Also

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_max_skew

Commands to define the max skew of a path.

Syntax

```
set_max_skew <net_list|pin_list> <skew_value>
```

Arguments

- **<net_list|pin_list:** This value is a list of pins or nets.
- **<skew_value>:** This value controls the delay to the various loads of the nket relative to each other.

Example

The following example shows that the first constraint requires the maximum difference between the net delays of the loads on the net connected to pin "A" to be less than or equal to 250 picoseconds. The second constraint requires the difference between the net delays to the various loads of the net "ff1/Q_c" to be less than or equal to 370 picoseconds:

```
set_max_skew [get_pins A] 0.25
set_max_skew [get_nets ff1/Q_c] 0.37
```

See Also

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_min_delay

Commands to specifie the minimum delay for the timing paths.

Syntax

```
set_min_delay
[-from <port_list|pin_list|instance_list|clock_list>]
[-to <port_list|pin_list|instance_list|clock_list>]
[-through <port_list|pin_list|instance_list|net_list>]
[-rise_from <clock_list>]
[-rise_to <clock_list>]
[-fall_from <clock_list>]
[-fall_to <clock_list>]
<delay_value>
```

Arguments

- **-from port_list|pin_list|instance_list|clock_list:** This value specifies the timing path start points (optional).

- **-to port_list|pin_list|instance_list|clock_list:** This value specifies the timing path end points (optional).
- **-through object_list:** This value specifies the timing path pass through points. The timing paths must go through these points (optional).
- **-rise_from|fall_from clock_list:** Same as '-from' but only applies to clock objects. Option '-from', 'rise_from' and '-fall_from' are mutually exclusive.
- **-rise_to|fall_to object_list:** Same as '-to' but only applies to clock objects. Option '-to', 'rise_to' and '-fall_to' are mutually exclusive. At least one option among '-from|-rise_from|-fall_from| -to|-rise_to|-fall_to|-through' is needed.
- **<delay_value>:** This value specifies a floating point number in nanoseconds that represents the required minimum delay value for specified paths.

Example

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

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_multicycle_path

Commands to define a path that takes multiple clock cycles.

Syntax

```
set_multicycle_path
[-from <port_list|pin_list|instance_list|clock_list>]
[-to <port_list|pin_list|instance_list|clock_list>]
[-through <port_list|pin_list|instance_list|net_list>]
[-rise_from <clock_list>] [-rise_to <clock_list>]
[-fall_from <clock_list>] [-fall_to <clock_list>]
[-setup | -hold] [-start | -end] <path_multiplier>
```

Arguments

- **-from port_list|pin_list|instance_list|clock_list:** This value specifies the timing path start point (optional).
- **-to port_list|pin_list|instance_list|clock_list:** This value specifies the timing path end point (optional).

- **-through port_list|pin_list|instance_list|net_list:** This value specifies the timing path pass through point. The timing path must go through this object (optional).
- **-rise_from|fall_from clock_list:** Same as '-from' but only applies to clock objects. Option '-from', 'rise_from' and '-fall_from' are mutually exclusive.
- **-rise_to|fall_to clock_list:** Same as '-to' but only applies to clock objects. Option '-to', 'rise_to' and '-fall_to' are mutually exclusive. At least one option among '-from|-rise_from|-fall_from|-to|-rise_to|-fall_to|-through' is needed.
- **-setup:** This option set Setup multiplier (optional).
- **-hold:** This option set Hold multiplier. Option '-setup' and '-hold' are mutually exclusive (optional).
- **-start:** This option indicates that Multiplier measured against startpoint (optional).
- **-end:** This option indicates that Multiplier measured against endpoint. Option '-start' and '-end' are mutually exclusive (optional).
- **<path_multiplier>:** This value specifies a value that represents the number of cycles.

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

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_output_delay

Commands to define the output delay of an output relative to a clock.

Syntax

```
set_output_delay -clock <clock_object> [-clock_fall]
[-max | -min] [-add_delay] <delay_value> <port_list>
```

Arguments

- **-clock clock_object:** This value specifies the clock reference to which the specified output delay is related.
- **-clock_fall:** This option is a switch to specify the falling edge of clock to trigger (optional).

- **-max:** This option specifies that the delay value is the maximum delay (optional).
- **-min:** This option specifies that the delay value is the minimum delay. Option '-max' and '-min' are mutually exclusive (optional).
- **-add_delay:** This option is used to define more than one output delay on a given port (optional).
- **<delay_value>:** This value specifies the amount of time from a reference clock to a primary output port.
- **<port_list>:** This value provides one or more output ports in the current design to which delay_value is assigned. You can also use the keyword '[all_outputs]' to include all output ports.

Example 1

```
The following example sets an output delay of 1.2 ns for
all outputs relative to CLK1
set_output_delay 1.2 -clock [get_clocks CLK1]
[get_ports OUT1]
set_output_delay 1.2 -clock [get_clocks CLK1]
[all_outputs]
```

Example 2

```
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

- "Radiant Software Timing Constraints Tcl Commands" on page 384

set_hierarchy_separator

Commands to set or get hierarchy separator.

Syntax

```
set_hierarchy_separator [<separator>]
```

Arguments

- **separator**: This option specifies the separator which is going to be set as hierarchy separator (optional).

If no separator is specified, current separator is print out.

Example 1

```
The following example sets '/'(forward slash)
as hierarchy separator:
     set_hierarchy_separator /
```

Example 2

```
The following example gets the current hierarchy separator:
     set_hierarchy_separator
```

See Also

- "Radiant Software Timing Constraints Tcl Commands" on page 384

Radiant Software Physical Constraints Tcl Commands

The **ldc** commands are used to set physical constraints.

Radiant Software Physical Constraints Tcl Command Descriptions

The following table provides a listing of all valid Radiant software physical constraints-related Tcl command options and describes option functionality.

Table 42: Radiant Software Physical Constraints Tcl Commands

Command	Description
ldc_create_constraint_set	Create a new constraint set.
ldc_create_group	Define a single identifier that refers to a group of objects.
ldc_create_macro	Define a macro.
ldc_create_region	Define a rectangular area.
ldc_create_vref	Define a voltage reference.
ldc_current_constraint_set	Switch the constraint set by the name.
ldc_define_attribute	Define LSE attributes.
ldc_define_global_attribute	Define LSE global attributes.
ldc_delete_constraint	Clear constraints by given IDs.
ldc_delete_constraint_set	Delete the constraint set by the name.
ldc_get_groups	Return group objects.
ldc_list_constraint	List constraints from database.
ldc_list_constraint_sets	List all existing constraint sets.
ldc_prohibit	Prohibit site or region.

Table 42: Radiant Software Physical Constraints Tcl Commands

Command	Description
ldc_read	Read constraint file.
ldc_set_attribute	Set object attributes.
ldc_set_location	Set object location.
ldc_set_port	Set port constraint attributes.
ldc_set_sysconfig	Set sysconfig attributes.
ldc_set_vcc	Set the voltage and/or derate for the bank, core or dphy.
ldc_write	Write constraint file.

ldc_create_constraint_set

Commands to create a new constraint set.

Syntax

```
ldc_create_constraint_set [-name_style logic ]
[-copy_from <copy_from_name>] <name>
```

Arguments

- **-name_style logic**: This value indicates that name style of new constraint set is logic. When this option is set, only constraint file with logical names can be loaded in this constraint set (optional).
- **-copy_from copy_from_name:** This value specifies an existing constraint set from which the new set will copy constraints (optional).
- **name:** This name string specifies the name of the new constraint set.

Example

```
The following example creates a new constraint set 'test'
with name style 'logic':
ldc_create_constraint_set -name_style logic test
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_create_group

Commands to define a single identifier that refers to a group of objects.

Syntax

```
ldc_create_group -name <group_name>
[-bbox {height  width}] <objects>
```

Arguments

- **-name group_name:** This value specifies the group name.
- **-bbox {height width}:** This value specifies the maximum number of rows (R) and columns (C), of the group's bounding box. This option is not applicable to port (optional).
- **object:** Objects named in the group, either port, instance, pin or net.

Example

```
ldc_create_group -name group1 [get_ports {a*}]
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_create_macro

Commands to define an instance in a design as a macro.

Syntax

```
ldc_create_macro [-name <macro_name>]
[-use_pio <ports_list>] <instance>
```

Arguments

- **-name macro_name:** This value specifies the macro name (optional).
- **-use_pio ports_list:** This value specifies macro ports that are directly connected outside of the design (optional).
- **instance:** The instance which is defined as a macro.

Example

```
ldc_create_macro -name macro_top -use_pio {din*}
[get_cells u_top]
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_create_region

Commands to define a rectangular area.

Syntax

```
ldc_create_region -name <region_name> (-anchor <anchor>|
      -site <site>) -width <width> -height <height>
      [-buffer_left <left>] [-buffer_right <right>]
      [-buffer_top <top>] [-buffer_bottom <bottom>]
      [-exclusive] [-routing_exclusive]
```

Arguments

- **-name region_name:** This value specifies the region name.
- **-anchor anchor:** This value specifies anchor.
- **-site site:** This value specifies the site name. Option '-anchor' and '-site' are mutually exclusive.
- **-width width:** This value specifies the width of the placement region in columns.
- **-height height:** This value specifies the height of the placement region in rows.
- **-buffer_left left:** This value specifies the left buffer zone of the routing region (optional).
- **-buffer_right right:** This value specifies the right buffer zone of the routing region (optional).
- **-buffer_top top:** This value specifies the top buffer zone of the routing region (optional).
- **-buffer_bottom bottom:** This value specifies the bottom buffer zone of the routing region. Options '–buffer_left','-buffer_right', '-buffer_top' and '-buffer_bottom' are only for creating macro region. Without these options routing region is identical to placement region (optional).
- **-exclusive:** This option indicates that the current buffer cannot be used to place and route other logic during the Reuse stage (optional).
- **-routing_exclusive:** This option indicates that the current buffer cannot be used to route other logic during the Reuse stage. Options '–exclusive' and '-routing_exclusive' are only for creating macro region (optional).

Example

```
ldc_create_region -name top_region -site R17C2D
-width 23 -height 39
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_create_vref

Commands to define a voltage reference.

Syntax

```
ldc_create_vref -name <vref_name> -site <site_name>
```

Arguments

- **-name vref_name:** This value specifies voltage reference name.
- **-site site_name:** This value specifies PIO site of the target device.

Example

```
ldc_create_vref -name VREF1_BANK_3 -site N21
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_current_constraint_set

Commands to switch the constraint set by the name.

Syntax

```
ldc_current_constraint_set [<name>]
```

Arguments

- **name:** This option specifies the constraint set which is going to be active. If no constraint set is specified, current active constraint set is print out (optional).

Example 1

The following example switches the constraint set 'test' to be active:

```
ldc_current_constraint_set test
```

Example 2

The following example gets the active constraint set:

```
ldc_current_constraint_set
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_define_attribute

Commands to define LSE attribute.

Syntax

```
ldc_define_attribute -attr <attr_name> -value
<attr_value> -object_type <type> -object <objects>
[-disable] [-comment <comment>]
```

Arguments

- **-attr attr_name:** This value indicates LSE attributes name, such as 'loc'.
- **-value attr_value:** This value indicates LSE attribute value such as '2'.
- **-object_type type:** This value indicates the type of object to which the attribute belongs, such as 'port'.
- **-object objects:** This value indicates the specific object to which the attribute applies, such as 'clk'.
- **-disable:** This option indicates that the command is disabled (optional).
- **-comment:** This option specifies comment of the command (optional).

Example

```
ldc_define_attribute -attr loc -value 2 -object_type
port -object {clk}
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_define_global_attribute

Commands to define global LSE attribute

Syntax

```
ldc_define_global_attribute -attr <attr_name> -value
    <attr_value> [-disable] [-comment <comment>]
```

Arguments

- **-attr attr_name:** This value indicates LSE attributes name, such as syn_ramstyle'.
- **-value attr_value:** This value indicates LSE attribute value such as 'registers'.
- **-disable:** This option indicates that the command is disabled (optional).
- **-comment:** This option specifies comment of the command (optional).

Example

```
ldc_define_global_attribute -attr syn_ramstyle
-value registers
```

See Also

- “Radiant Software Physical Constraints Tcl Commands” on page 398

ldc_delete_constraint

Commands to clear constraints by given IDs.

Syntax

```
ldc_delete_constraint [-all] [<id>] [<id>...]
```

Arguments

- **-all**: This option indicates that the command will delete all constraints (optional).
- **-id**: This value specifies the id of constraint which is going to be deleted (optional).

Example

The following example deletes constraint with index 3:

```
ldc_delete_constraint 3
```

See Also

- “Radiant Software Physical Constraints Tcl Commands” on page 398

ldc_delete_constraint_set

Commands to delete the constraint set by the name.

Syntax

```
ldc_delete_constraint_set <name>
```

Arguments

- **name:** This value specifies the name of the constraint set which is going to be deleted. Active constraint set can't be deleted.

Example

```
The following example deletes constraint set 'test':
ldc_delete_constraint_set test
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_get_groups

Commands to return group objects.

Syntax

```
ldc_get_groups [-regexp] [-nocase] [<patterns>]
```

Arguments

- **-regexp:** This option indicates that patterns are full regular expressions (optional).
- **-nocase:** This option indicates that matching is case-insensitive (optional).
- **patterns:** This value specifies the patterns against which group names are matched. If no pattern is specified, set '*' as default (optional).

Example

```
ldc_get_groups -nocase -regexp group\[0-9]+
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_list_constraint

Commands to list constraints from database.

Syntax

```
ldc_list_constraint [-all|-a]
```

Arguments

- **-all|-a**: This option indicates that the command will get all the constraints including disabled or incorrect constraints (optional).

Example

The following example lists enabled and correct constraints from database:

```
ldc_list_constraint
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_list_constraint_sets

Commands to list all existing constraint sets.

Syntax

```
ldc_list_constraint_sets
```

Example

The following example lists all existing constraint sets in the current view:

```
ldc_list_constraint_sets
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_prohibit

Commands to prohibit the use of a site or all sites in a region.

Syntax

```
ldc_prohibit (-site <site> | -region <region>)
```

Arguments

- **-site site:** This value specifies site name.
- **-region region:** This value specifies region name. Option '-site' and '-region' are mutually exclusive.

Example

```
ldc_prohibit -site AB
ldc_prohibit -region regionA
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_read

Commands to read constraint file.

Syntax

```
ldc_read <file>
```

Arguments

- **file:** This value specifies the constraint file which is going to be read.

Example

```
The following example reads constraint file test.sdc:
ldc_read test.sdc
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_set_attribute

Commands to set constraint attributes to the objects.

Syntax

```
ldc_set_attribute <key-value list> [objects]
```

Arguments

- **key-value list:** This value is a list of constraint attributes.
- **objects:** This value specifies object(s), to which constraint attributes are set. If no object is specified, attributes are set to design (optional).

Example

```
ldc_set_attribute GSR_NET=TRUE [get_nets {my_gsr}]
```

See Also

- “Radiant Software Physical Constraints Tcl Commands” on page 398

ldc_set_location

Commands to set object location.

Syntax

```
ldc_set_location [-site <site_name>] [-bank <bank_num>]
    [-region <region_name>] <object>
```

Arguments

- **-site site_name:** This value specifies name of the site to place object (optional).
- **-bank bank_num:** This value specifies number of the bank to place object (optional).
- **-region region_name:** This value specifies name of the user defined region to place object. One option among '-site' ,'-bank' and '-region' is needed and they are mutually exclusive (optional).
- **object:** Object to be located.

Example

```
ldc_set_location -site U6 [get_ports clk]
ldc_set_location -region region0 [ldc_get_groups {group1}]
```

See Also

- “Radiant Software Physical Constraints Tcl Commands” on page 398

ldc_set_port

Commands to set port constraint attributes.

Syntax

```
ldc_set_port (-iobuf [-vref <vref_name>]| -sso)
<key-value list> [ports]
```

Arguments

- **-iobuf:** This option indicates that IOBUF attributes are set.
- **-vref vref_name:** This value specifies voltage reference name. It must be bound to -iobuf (optional).
- **-sso:** This option indicates that SSO attributes are set. Option '-iobuf' and '-sso' are mutually exclusive.
- **key-value list:** This value is a list of attributes. Attributes must be enclosed with curly braces.
- **ports:** This value specifies ports to be set. If no port is specified, set constraint to all ports (optional).

Example

```
ldc_create_vref -name VERF2 -site R11
ldc_set_port -iobuf -vref VERF2 {IO_TYPE=HSTL15_I
VREF=VREF2_LOAD} [get_ports clk_third]
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_set_sysconfig

Commands to set sysconfig attributes.

Syntax

```
ldc_set_sysconfig <key-value list>
```

Arguments

- **key-value list:** This value is a list of SYSCONFIG attributes. Attributes must be enclosed with curly braces.

Example

```
ldc_set_sysconfig {JTAG_PORT=ENABLE PROGRAMN_PORT=ENABLE
    MCCLK_FREQ=56.2 DONE_OD=ON}
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_set_vcc

Commands to set the voltage and/or derate for the bank, core or dphy.

Syntax

```
ldc_set_vcc (-bank <bank_number>|-core|-dphy DPHY0/DPHY1)
    [-derate <derate>] [<voltage>]
```

Arguments

- **-bank bank_number:** This value specifies the bank to be set with voltage or voltage derating percent.
- **-core:** This option indicates that voltage or voltage derating percent is set to core.
- **-dphy DPHY0/DPHY1:** This option indicates that voltage or voltage derating percent is set to dphy. Value of option '-dphy' can only be 'DPHY0' or 'DPHY1'. Option '-bank', '-core' and '-dphy' are mutually exclusive.
- **-derate:** This value specifies voltage derating percent (optional).
- **voltage:** This value can be any compatible voltage supply to that bank or the core or dphy; for example, 2.5, 3.3. At least one option between '-derate' and 'voltage' is needed (optional).

Example

```
ldc_set_vcc -bank 1 3.3
ldc_set_vcc -dphy DPHY0 -derate -3
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

ldc_write

Commands to write constraint file.

Syntax

```
ldc_write [-overwrite|w] [-sdc] [-exclude_sdc]
[-include_error] <file>
```

Arguments

- **-overwrite|w:** This option indicates that the command will overwrite file of the same name (optional).
- **-sdc:** This option indicates that only SDC timing constraints and Synthesis attributes will be written to file (optional).
- **-exclude_sdc:** This option indicates that SDC timing constraints and Synthesis attributes won't be written to file. Option '-sdc' and '-exclude_sdc' are mutually exclusive (optional).
- **-include_error:** This option indicates that incorrect constraints will also be written to file (optional).
- **file:** This value specifies the constraint file which is going to be written.

Example

```
ldc_write -sdc -include_error test.sdc
```

See Also

- "Radiant Software Physical Constraints Tcl Commands" on page 398

Radiantc TCL Commands

Radiant has two design flow modes: GUI and pure TCL mode. In pure TCL mode, you can use all TCL commands. However, in GUI mode (including the TCL section within it), Radiant oversees the project process and status without retaining the design in memory. Consequently, any TCL commands that require access to design data are unavailable. The list of these commands is provided below.

- Bitstream Generation Tcl Commands
- Design Tcl Commands
- Device Tcl Commands
- Technology Mapping Tcl Commands
- Placement Tcl Commands
- Routing Tcl Commands
- Unified Timing Analysis

- Timing Analysis Tcl Commands

Bitstream Generation Tcl Commands

The **bit** command is used for bitstream generation.

Bitstream Generation Tcl Command Descriptions

The following table provides a listing of all valid Radiant software bitstream generation-related Tcl command options and describes option functionality.

Table 43: Bitstream Generation Tcl Commands

Command	Description
bit_generate	Generate a bitstream file.
bit_list_option	List bit generation options.
bit_set_option	Set bit generation options.

bit_generate

Generate bitstream file.

Syntax

```
bit_generate [-w] <filename>
```

Arguments

- **-w**: Overwrite existing bitstream file if <filename> exists.
- **<filename>**: Specify output bitstream file. Do not include '.bit' as it is appended (required).

Example

```
bit_generate -w mydesign
```

See Also

- "Bitstream Generation Tcl Commands" on page 412

bit_list_option

List bitgen options.

Syntax

```
bit_list_option
```

Example

```
bit_list_option
```

See Also

- "Bitstream Generation Tcl Commands" on page 412

bit_set_option

Set bitgen options.

Syntax

```
bit_set_option {<key> <value> ...}
```

Example

```
bit_list_option
bit_set_option {enable_early_io_wakeup true ip_evaluation
false}
```

See Also

- "Bitstream Generation Tcl Commands" on page 412

Design Tcl Commands

The **des** commands are used to read or write the Radiant unified design database (.udb) file and list and report netlist objects such as instance, net, or port. You can get these commands by using **help {des_*}**.

The following table provides a listing of all valid Radiant software design-related Tcl command options and describes option functionality.

Table 44: Design Tcl Commands

Command	Description
des_get_ram_cells	Retrieves memory cells in the design that are implemented as either distributed RAM (LUTRAM) or block RAM (EBR).
des_get_ram_style	Returns the memory style (LUTRAM or EBR) of a specified RAM cell in the design.
des_get_util_data	Return the required data provided/used in device/ design.
des_list_attribute	List design attributes.
des_list_instance	List instance.

Table 44: Design Tcl Commands

des_list_net	List all nets.
des_list_port	List port.
des_list_reference_udb	List reference udb(s) for increment flow.
des_read_udb	Read in a .udb file.
des_reload_milestone	Reload milestone result.
des_report	Report design info.
des_report_flow_status	Report design flow status.
des_report_instance	Report instance information.
des_report_net	Report a net.
des_report_operating_condition	Report operating condition.
des_report_port	Report port information.
des_set_attribute	Set design attributes.
des_set_operating_condition	Change operating condition.
des_set_reference_udb	Set reference udb(s) for increment flow.
des_write_udb	Write the design into a .udb file.

des_get_ram_cells

Retrieves memory cells in the design that are implemented as either distributed RAM (LUTRAM) or block RAM (EBR).

Syntax

```
des_get_ram_cells -ramstyle <lutram|ebr>
```

Arguments

- **-ramstyle**: Specifies the memory implementation style to filter the results.

  Valid values:
  -lutram: Distributed RAM (also known as LUTRAM or DPRAM)
  -ebr: Embedded Block RAM (also known as EBR)

Example

```
des_get_ram_cells -ramstyle lutram
Retrieves all memory cells implemented using distributed RAM
(LUTRAM).
```

See Also

- "Design Tcl Commands" on page 413

des_get_ram_style

Returns the memory style (LUTRAM or EBR) of a specified RAM cell in the design.

Syntax

```
des_get_ram_style <cellname>
```

Arguments

- **cellname:** The name of the memory cell whose implementation style is to be queried.

Example

```
des_get_ram_style cellname
Returns the memory style (either 'lutram' for distributed RAM
or 'ebr' for block RAM) of the specified cell.
```

See Also

- "Design Tcl Commands" on page 413

des_get_util_data

Returns the required resource provided/used in device/design.

Syntax

```
des_get_util_data [-device][-lut][-reg][-slice][-dpram]
[-ripplelogic][-ebr][-dsp][-pio][-pll][-gsr][-osc]
```

Arguments

- **-device**: Returns the resource data provided in the device (optional).
- **-lut**: Returns number of LUTs (optional).
- **-reg**: Returns number of REGs (optional).
- **-slice**: Return number of Slices (optional).
- **-dpram**: Returns number of DPRAMs (optional).
- **-ripplelogic**: Return number of slices used in carry chains (optional).
- **-ebr**: Return number of EBRs (optional).
- **-dsp**: Return number of DSPs (optional).
- **-pio**: Return number of PIOs (optional).
- **-pll**: Return number of PLLs (optional).
- **-gsr**: Return number of GSRs (optional).
- **-osc**: Return number of oscilators (optional).

Example

```
des_get_util_data -pio
Return # of PIO instances used in the design.

des_get_util_data -device -pio
Return the # of PIOs provides in the device.
```

If the return value is -1, it means that the selected data type is not available in the device, or the data type has not been supported.

See Also

- "Design Tcl Commands" on page 413

des_list_attribute

Reports design attributes.

Syntax

```
des_list_attribute
```

Example

```
des_list_attribute
```

See Also

- "Design Tcl Commands" on page 413

des_list_instance

Commands to list specified instances.

Syntax

```
des_list_instance [<instances>]
```

Arguments

- **<instances>**: Only lists instances that match the pattern specified in <instances> (optional). If no pattern is specified, all instances will be listed.

Example

```
des_list_instance *CWDATA_c*
```

See Also

- "Design Tcl Commands" on page 413

des_list_net

Commands to list specified nets.

Syntax

```
des_list_net [-clock] [-fanout <n>] [-lsort <method>]
[-lsize <n>] [<nets>]
```

Arguments

- **-clock**: Lists clock nets only (optional).
- **-fanout <n>**: Lists nets with # of fanouts >= n (optional).
- **-lsort <method>**: Choose sorting method: 0 - id; 1 - all pins; 2 - clk pins; 3 - global control pins; 4 - normal pins (optional). Defaults to sorting by id.
- **-lsize <size>**: Limits the number of listed nets to size. Defaults to showing all nets (optional).
- **<nets>**: Only lists net names that match the pattern specified in <nets> (optional).

Example

```
des_list_net -fanout 10 -lsort 2 -lsize 10 *D0*
```

This command lists the first 10 nets that start with D0, have a fanout of at least 10, sorted by clk pins first.

See Also

- "Design Tcl Commands" on page 413

des_list_port

Commands to list specified ports.

Syntax

```
des_list_port [<ports>]
```

Arguments

- **<ports>**: Only lists instances that match the pattern specified in <ports> (optional). If no pattern is specified, all ports will be listed.

Example

```
des_list_port *CWDATA_c*
```

See Also

- "Design Tcl Commands" on page 413

des_list_reference_udb

Lists reference udb(s) for incremental flow, if there's any.

Syntax

```
des_list_reference_udb
```

Example

```
des_list_reference_udb
```

See Also

- "Design Tcl Commands" on page 413

des_read_udb

Commands to read database from udb file.

Syntax

```
des_read_udb [-logview <logical udb file>]
[-phyview <physical udb file>] [<udb file>]
```

Arguments

- **-logview <logical udb file>**: Specifies the logical udb file name (optional).
- **-phyview <physical udb file>**: Specifies the physical udb file name (optional).
- **<udb file>**: Specifies the udb file name (optional).

  -logview, -phyview, and <udb file> options are mutually exclusive.

  If either -logview, -phyview, or both are specified, you cannot specify a udb file.

Example

```
des_read_udb -logview logic.udb -phyview phy.udb
des_read_udb post_par.udb
```

See Also

- "Design Tcl Commands" on page 413

des_reload_milestone

Reloads milestone result.

Syntax

```
des_reload_milestone [-force] [-inc] [-postsyn|-postmap|
-postplc|-postrte|-postpar] <udb_name>
```

Arguments

- **-force**: Forcibly opens the milestone (optional).
- **-inc**: Loads the milestone result incrementally (optional).
- **-postsyn**: Opens a post-synthesis milestone (optional).
- -**postmap**: Opens a post-map milestone (optional).
- **-postplc**: Opens a post-placement milestone (optional).
- **-postrte**: Opens a post-routing milestone (optional).
- **-postpar**: Opens a post-par milestone (optional).
- **<udb_name>**: Milestone udb to open (required).

Example

```
des_reload_milestone -postsyn -inc mydb_syn.udb
```

See Also

- "Design Tcl Commands" on page 413

des_report

Reports design information.

Syntax

```
des_report
```

Example

```
des_report
```

See Also

- "Design Tcl Commands" on page 413

des_report_flow_status

Reports design flow status.

Syntax

```
des_report_flow_status
```

Example

```
des_report_flow_status
```

See Also

- "Design Tcl Commands" on page 413

des_report_instance

Commands to report specified instances.

Syntax

```
des_report_instance [<instances>]
```

Arguments

- **<instances>**: Only reports instances that match the pattern specified in <instances> (optional).

Example

```
des_report_instance *CWDATA_c*
```

See Also

- "Design Tcl Commands" on page 413

des_report_net

Commands to report specified nets.

Syntax

```
des_list_net [-file <filename>] [<nets>]
```

Arguments

- **-file <filename>**: Outputs results into <filename> (optional).
- **<nets>**: Only reports nets that match the pattern specified in <nets> (optional).

Example

```
des_report_net -file mynet.txt *CWDATA_c*
```

See Also

- "Design Tcl Commands" on page 413

des_report_operating_condition

Reports operating condition information.

Syntax

```
des_report_operating_condition
```

Example

```
des_report_operating_condition
```

See Also

- "Design Tcl Commands" on page 413

des_report_port

Commands to report specified ports.

Syntax

```
des_report_port [<port>]
```

Arguments

- **<ports>**: Only reports ports that match the pattern specified in <ports> (optional). If no pattern is specified, all ports are reported.

Example

```
des_report_port *CWDATA_c*
```

See Also

- "Design Tcl Commands" on page 413

des_set_attribute

Changes design attributes.

Syntax

```
des_set_attribute [-impl_dir <name>]
```

Arguments

- **-impl_dir <name>**: Sets implementation directory to <name> (optional). Defaults to current directory otherwise.

Example

```
des_set_attribute -impl_dir impl_1
```

See Also

- "Design Tcl Commands" on page 413

des_set_operating_condition

Changes operating condition of device.

Syntax

```
des_set_operating_condition [-temperature <temp>]
[-speed_grade <speed_grade>]
```

Arguments

- **-temperature <temp>**: Set device temperature to <temp> degrees Celsius (optional). Defaults to -1.
- **-speed_grade <speed_grade>**: Sets device speed grade to specified setting <speed_grade> (optional)

Example

```
des_set_operating_condition -temp 100 -speed_grade 9_High-
Performance_1.0V
```

Sets temperature to 100 Celcius with High Performance speed grade.

See Also

- "Design Tcl Commands" on page 413

des_set_reference_udb

Sets reference udb(s) for incremental flow.

Syntax

```
des_set_reference_udb [-postsyn <postsyn_udb>]
[-postmap <postmap_udb>][-postpar <postpar_udb>]
[-pdc <pdc_file>] [-clean]
```

Arguments

- **-postsyn <postsyn_udb>:** Set post-synthesis reference udb file to <postsyn_udb> (optional).
- **-postmap <postmap_udb>:** Set post-map reference udb file to <postmap_udb> (optional).
- **-postpar <postpar_udb>:** Set post-par reference udb file to <postpar_udb> (optional).
- **-pdc <pdc_file>:** Set reference pdc file to <pdc_file> (optional).
- **-clean:** Reset all reference udb settings (optional).

Example

```
des_set_reference_udb -postsyn mydb_syn.udb -postpar
mydb_par.udb -pdc mypdc.pdc
```

Set post-synthesis reference udb to mydb_syn.udb, post-par udb to mydb_par.udb and pdc file to mypdc.pdc.

```
des_set_reference_udb -clean
```

Removes any reference udb settings.

See Also

- "Design Tcl Commands" on page 413

des_write_udb

Commands to write udb database file.

Syntax

```
des_write_udb [-w] [-logview] [-phyview] [<udb file>]
```

Arguments

- **-w**: Overwrite existing file (optional).
- **-logview:** Output logical view (optional).
- **-phyview**: Output physical view (optional).

- **<udb file>**: This option specifies the udb file name (optional).

  -logview, -phyview, and <udb file> options are mutually exclusive.

Example

```
des_write_udb -logview logic.udb -phyview phy.udb
des_write_udb -w post_par.udb
```

See Also

- "Design Tcl Commands" on page 413

Device Tcl Commands

The **dev** commands are used re used to access or report device information. You can get these commands by using **help {dev_*}**.

The following table provides a listing of all valid Radiant software device-related Tcl command options and describes option functionality.

Table 45: Device Tcl Commands

Command	Description
dev_report	Report the device info.

dev_report

Report device information.

Syntax

```
dev_report
```

Arguments

Example

```
dev_report
```

See Also

- "Device Tcl Commands" on page 425

Technology Mapping Tcl Commands

The **map** commands are used to map or transfer a design from a logical view to a physical view.

Radiant Software Technology Mapping Tcl Command Descriptions

The following table provides a listing of all valid Radiant software mapping-related Tcl command options and describes option functionality.

Table 46: Technology Mapping Tcl Commands

Command	Description
map_list_option	List map options.
map_run	Run map tool.
map_set_option	Set map options.

map_list_option

List map options.

Syntax

```
map_list_option
```

Example

```
map_list_option
```

See Also

- "Technology Mapping Tcl Commands" on page 426

map_run

Run the map tool.

Syntax

```
map_run
```

Example

```
map_run
```

See Also

- "Technology Mapping Tcl Commands" on page 426

map_set_option

Set the map options.

Syntax

```
map_set_option {<key> <value> ...}
```

Example

```
map_list_option
map_set_option {infer_gsr true pdc_file myfile.pdc}
```

See Also

- "Technology Mapping Tcl Commands" on page 426

Placement Tcl Commands

The **plc** commands are used for automatic/interactive placement.

Placement Tcl Command Descriptions

The following table provides a listing of all valid Radiant software placement-related Tcl command options and describes option functionality.

Table 47: Placement Tcl Commands

Command	Description
plc_list_option	List placement options.
plc_output_instance_location	Output instance locations in LDC format.
plc_report	Report placement information.
plc_run	Run placement tool.
plc_set_option	Set placement options.

plc_list_option

Lists placement options.

Syntax

```
plc_list_option
```

Example

```
plc_list_option
```

See Also

- "Placement Tcl Commands" on page 427

plc_output_instance_location

Outputs instance locations in LDC format.

Syntax

```
plc_output_instance_location [-all] [-pio] [-dsp] [-ebr] [-
slice] [-all_constraint] -file <filename> [<pattern>]
```

Arguments

- **-all**: Selects all instance types to output (optional).
- **-pio**: Selects PIO instances to output (optional).
- **-dsp**: Select DSP instances to output (optional).
- **-ebr**: Select EBR instances to output (optional).
- **-slice**: Select slices to output (optional).
- **-all_constraint**: Select all constraint instances to output (optional).
- **-file <filename>**: Specify output file to write to (required).
- **-<pattern>**: Only output instances that match <pattern> (optional).

Example

```
plc_output_instance_location -dsp -file newfile.ldc CW*
```

This example outputs DSP instances that start with CW into a new constraint file called **newfile.ldc**.

See Also

- "Placement Tcl Commands" on page 427

plc_report

Reports placement information such as sites, comps, bounding box, and estimated wiring segments.

Syntax

```
plc_report
```

Example

```
plc_report
```

See Also

▶ "Placement Tcl Commands" on page 427

plc_run

Runs placement tool.

Syntax

```
plc_run
```

Example

```
plc_run
```

See Also

▶ "Placement Tcl Commands" on page 427

plc_set_option

Sets placement options.

Syntax

```
plc_set_option {<key> <value> ...}
```

Example

```
plc_list_option
plc_set_option {disable_timing_driven true path_based_placement
on}
```

See Also

▶ "Placement Tcl Commands" on page 427

Routing Tcl Commands

The **rte** commands are used for automatic/interactive routing.

Routing Tcl Command Descriptions

The following table provides a listing of all valid Radiant software routing-related Tcl command options and describes option functionality.

Table 48: Routing Tcl Commands

Command	Description
rte_list_option	List routing options.
rte_report	Report routing results.
rte_run	Run the routing tool.
rte_set_option	Set routing options.

rte_list_option

List routing options.

Syntax

```
rte_list_option
```

Example

```
rte_list_option
```

See Also

- "Routing Tcl Commands" on page 429

rte_report

Report routing results.

Syntax

```
rte_report
```

Arguments

- **-net**: Report the nets (optional).

Example

```
rte_report
```

See Also

- "Routing Tcl Commands" on page 429

rte_run

Run the routing tool.

Syntax

```
rte_run
```

Example

```
rte_run
```

See Also

- "Routing Tcl Commands" on page 429

rte_set_option

Set routing options.

Syntax

```
rte_set_option {<key value> ... }
```

Example

```
rte_list_option
rte_set_option {disable_timing_driven true}
```

See Also

"Routing Tcl Commands" on page 429

Unified Timing Analysis

This introduces a Consolidated Timing Analysis (TA) experience that unifies the separate timing analysis workflows into a single, streamlined Tcl command interface.

In earlier releases, you may have used different tools — Post-PAR Timing Report, Timing Analyzer, Standalone Timing Analyzer, or radiantc — each with its own report formats, Tcl capabilities, and constraint support.

Starting with 2026.1, these have been consolidated at the Tcl level so you can work with one consistent set of commands for timing reporting, constraint manipulation, and path analysis.

Features and Enhancements

- **Unified Tcl Reporting Commands**
  A new set of consolidated Tcl commands replaces the previously

fragmented reporting interfaces. You can now generate timing reports — including standard and custom reports — using a single, consistent command set. This applies to all FPGA families supported by Radiant.

- **Consolidated Constraint Manipulation & Reporting**
  Timing constraint manipulation is available through unified Tcl commands. You can create, query, and report on constraints from a single interface, eliminating the need to switch between tools when managing SDC/PDC constraints and inspecting their effects.
- **sta_get_path_info Command with Start/End Point Support**
  The **sta_get_path_info** command replaces the legacy sta_path_info and introduces the ability to extract start point and end point objects directly from a timing path. This makes the scripted timing analysis significantly more powerful.
- **Improved Constraint Coverage**
  The 2026.1 release includes comprehensive coverage of complex constraint combinations, means more reliable handling of advanced SDC/PDC constraint scenarios.
- **Renamed Commands and Options**

Several Tcl commands and options have been updated for consistency, update them accordingly:

Table 49: Renamed Commands and Options

Previous Command/Option	Current Command/Option
sta_get_clocks	get_clocks
sta_get_terms	get_ports get_pins
sta_get_info	sta_get_clock_info sta_get_term_info
sta_path_info	sta_get_term_info

Note

The legacy **sta_path_info** command displays a deprecation warning in 2026.1 and is planned for removal in a future release. Migrate your scripts to use sta_get_path_info.

Timing Analysis Tcl Commands

The **sta** commands are used to run timing analysis interactively, with two major categories, namely **sta_get** and **sta_report**. The sta_get command returns the value, which can be used in other sta commands or printed out to the standard output. The sta_report command sends the information to the standard output.

You can get these commands by using **help {sta_*}**.

For a complete discussion on interactive static timing analysis using the TCL commands, refer to the FPGA-AN-02091, Interactive Timing Analysis Using TCL in Lattice Radiant Design Software application note.

As an added reference, you may also refer to the FPGA-AN-02059, Lattice Radiant Timing Constraints Methodology application note, which provides guidelines and best practices for defining and managing timing constraints in Radiant.

The following table provides a listing of all valid Radiant software timing analysis-related Tcl command options and describes option functionality.

Table 50: Timing Analysis Tcl Commands

Command	Description
sta_get_clocks	Get a list of clocks that satisfy the name.
sta_get_coverage	Get the timing coverage for connections.
sta_get_info	Get information about a clock or a terminal (port/ pin).
sta_get_paths	Change operating condition of the device.
sta_get_slack	Get the total negative, worst or terminal slack for the design.
sta_get_terms	Get a list of ports and pins that satisfy the name.
sta_get_term_info	Get information on a given path.
sta_get_clock_info	Get information about a clock.
sta_get_term_info	Get information about a terminal.
sta_report_clocks	Report on a clock or clocks.
sta_report_constraints	List timing constraints.
sta_report_timing	Report detailed or summary timing.
sta_report_unconstrained	List unconstrained ports, pins, or nets.
rte_set_option	Change the style of the timing report.

Note

- To prevent errors, use des_read_udb command first to read the PAR UDB before performing timing analysis.
- If you do not change the default PAR Timing Strategy (Speed Grade for Hold Analysis) and the Junction Temperature value in Device Constraint Editor, there may be differences between the Radiant GUI and the TCL results.
- In the Radiant GUI Timing Analysis report, the Hold Analysis of STA uses Speed Grade "m" and minimum temperature for the operating condition (for example, 0 °C for commercial and –40 °C for industrial). In the TCL timing analysis, on the other hand, the Hold Analysis of STA uses the speed grade of the device and the maximum temperature for the operating condition (for example, 85 °C for commercial and 100 °C for industrial).

sta_get_clocks

Commands to list requested clocks.

Syntax

```
sta_get_clocks [<name>]
```

Arguments

- **<name>:** Only list clock names that match the pattern specified in <name> (Required)

Example

```
sta_get_clocks clk *
```

See Also

- "Timing Analysis Tcl Commands" on page 432

sta_get_clock_info

Command to get information about a clock.

Syntax

```
sta_get_clock_info -name <clock name> [-waveform] [-period]
    [-duty_cycle] [-fanout]
```

Arguments

- **<name> <clock name>:** Name of clock whose info is requested (required).
- **-waveform:** Returns waveform of the clock (optional).
- **-period:** Returns period of the clock (optional).
- **-duty_cycle:** Returns duty cycle of the clock (optional).
- **-fanout:** Returns fanout (number of loads) on the clock network (optional). Use one and only one of -waveform, -period, -duty_cycle, and -fanout.

Example

```
sta_get_clock_info -name clk1 -duty_cycle
    sta_get_clock_info -name clk1 -fanout
```

See Also

- sta_get_info and sta_report_clocks
- "Timing Analysis Tcl Commands" on page 432

sta_get_coverage

Commands to list specified instances.

Syntax

```
sta_get_coverage
```

Example

```
sta_get_coverage
```

See Also

- "Timing Analysis Tcl Commands" on page 432

sta_get_info

Commands to get information about clocks or terminals.

Syntax

```
sta_get_info [-clock <clock name>] [-waveform]
[-period][-duty_cycle] [-terminal <terminal name>]
[-isclockpin][-isinput] [-isoutput] [-net] [-cell]
[-termClocks]
```

Arguments

- **-clock <clock name>:** Name of clock whose info is requested <clock name> (optional)
- **-waveform:** requests waveform of clock (optional)
- **-period:** period of clock will be returned (optional)
- **-duty_cycle:** duty cycle will be returned (optional)
- **-terminal <terminal name>:** Only report on terminal that match the pattern specified in <terminal name> (optional)
- **-isclockpin:** Returns true if the terminal is a clock pin (optional)
- **-isinput:** Returns true if the terminal is an input (optional)
- **-isoutput:** Returns true if the terminal is an output (optional)
- **-net:** Net connected to terminal (optional)
- **-cell:** Parent cell of terminal (optional)
- **-termClocks:** User defined clocks that arrive at the terminal

  Option -clock is mutually exclusive with options -terminal, -isclockpin, -isinput, -isoutput, -net, -cell, and -termClocks.

  Option -terminal is mutually exclusive with options -waveform, -period, and -duty_cycle. (optional)

Example

```
sta_get_info -clock clk1 -duty_cycle
sta_get_info -terminal abc/LSR -net
```

See Also

- "Timing Analysis Tcl Commands" on page 432

sta_get_paths

Change operating condition of device.

Syntax

```
sta_get_paths [-n <number>][-nperend <number>]
[-slack_limit <number>][-from <ports/pins>]
[-from_clock <clocks>] [to <ports/pins>]
[-to_clock <clocks>] [-hold]
```

Arguments

- **-n <number>:** The total number of paths reported is up to <number> paths (optional).
- **-nperend <number>:** The potential number of paths reported per endpoint. The number of paths is still controlled by n (optional).
- **-slack_limit <number>:** The upper limit of the slack of the paths that are being reported. The number of paths is still controlled by n (optional).
- **-from <ports/pins>:** The from pins of the report are given by <ports/pins> (optional).
- **-from_clock <clocks>:** The launch clock of the report are given by <ports/pins> (optional).
- **-to <ports/pins>:** The to pins of the report are given by <ports/pins> (optional).
- **-to_clock <clocks>:** The latch clocks of the report are given by <ports/pins> (optional).
- **-hold:** Report hold timing paths (optional).

Example

```
sta_get_paths -n 5 -from abcd/Q0
```

See Also

- "Timing Analysis Tcl Commands" on page 432

sta_get_slack

Command to report slack of design or terminal.

Syntax

```
sta_get_slack [-tns] [-worst] [-hold] [-terminal <terminal>]
```

Arguments

- **-tns:** Get the total negative slack of the design (optional)
- **-worst:** Get the worst slack of the design (optional)
- **-hold:** Get the hold slack (optional)

- **-terminal <terminal>:** Get the slack of the terminal specified by <terminal> (optional)

  Option -terminal is mutually exclusive with the options -tns and -worst

Example

```
sta_get_slack -terminal abcd/e -hold
sta_get_slack -tns
```

See Also

- “Timing Analysis Tcl Commands” on page 432

sta_get_terms

Command to get a list of terms.

Syntax

```
sta_get_terms [-ports | -pins] [<name of ports or pins>]
```

Arguments

- **-ports:** The name represents ports (optional)
- **-pins:** The name represents pins (optional)
- **<name>:** Name or ports or pins (required). Option '-ports and -pins are mutually exclusive.

Example

```
sta_get_terms *
sta_get_terms -ports *
sta_get_terms -pins */LSR
```

See Also

- “Timing Analysis Tcl Commands” on page 432

sta_get_path_info

Get information on a given path.

Syntax

```
sta_get_path_info [-path <path>] [-slack] [-constraint]
[-skew] [-launch_clock] [-capture_clock] [-levels]
[-start_point] [-end_point] [-datapath]
```

Arguments

- **-path:** A path object obtained with sta_get_paths. (required)
- **-slack:** Return the slack of the path. (optional)
- **-constraint:** Returns the value of the constraint that must be met for the given path (optional).
- **-skew:** Returns the clock skew (optional).
- **-launch_clock:** Return the clock launching the data at the start point (optional).
- **-capture_clock:** Return the clock capturing the data at the endpoint (optional).
- **-levels:** Return number of logic levels (optional).
- **-start_point:** Return the start_point of the path (optional).
- **-end_point:** Return the end_point of the path (optional).
- **-datapath:** Return a Tcl list of data-path timing point names in order (same order as detailed path reporting: from path start to path end) (optional).

Example

```
sta_get_path_info -path <path>
```

See Also

- "Timing Analysis Tcl Commands" on page 432

sta_get_term_info

Command to get information about a terminal.

Syntax

```
sta_get_term_info -name <terminal name> [-isclockpin]
    [-isinput] [-isoutput] [-istimingstart] [-istimingend]
    [-termClocks]
```

Arguments

- **-name <terminal name>**: Only report on terminal that matches the pattern specified in <terminal name> (required).
- **-isclockpin:** Returns true if the terminal is a clock pin (optional).
- **-isinput:** Returns true if the terminal is an input (optional).
- **-isoutput:** Returns true if the terminal is an output (optional).
- **-istimingstart:** Returns true if the terminal is a timing start point (optional).
- **-istimingend:** Returns true if the terminal is a timing end point (optional).
- **-termClocks:** User defined clocks that arrive at the terminal (optional).

Use one and only one of -isclockpin, -isinput, -isoutput, -termClocks, -istimingstart or -istimingend.

Example

```
sta_get_term_info -name abc/LSR -isclockpin
    sta_get_term_info -name abc/LSR -istimingstart
```

See Also

- sta_get_clock_info and sta_get_slack
- “Timing Analysis Tcl Commands” on page 432

sta_report_clocks

Report design clocks.

Syntax

```
sta_report_clocks [-clocks <clocks>] [-file <file name>] [-app]
```

Arguments

- **-clocks <clocks>:** Report clocks that are indicated by <clocks> (optional)
- **-file <file name>:** Name of file that report is to be redirected to (optional)
- **-app:** Append report to end of file (optional)

Example

```
sta_report_clocks -clocks cl* -file abc -app
```

See Also

- “Timing Analysis Tcl Commands” on page 432

sta_report_constraints

Command to list timing constraints.

Syntax

```
sta_report_constraints [-file <file name>] [-app]
```

Arguments

- **-file <file name>:** Name of file that report is to be redirected to <file name> (optional)
- **-app:** Append the report to an existing file (optional)

Example

```
sta_report_constraints
sta_report_constraints -file abc -app
```

See Also

- "Timing Analysis Tcl Commands" on page 432

sta_report_timing

Change operating condition of device.

Syntax

```
sta_report_timing [-n <number>] [-endpoints]
[-from <ports/pins>] [-from_clock <clocks>]
[to <ports/pins>] [-to_clock <clocks>] [-hold]
[-summary] [-file <file name>] [-app]
```

Arguments

- **-n <number>:** The total number of paths reported is up to <number> paths (optional).
- **-endpoints:** Report the endpoints only (optional)
- **-from <ports/pins>:** The from pins of the report are given by <ports/pins> (optional)
- **-from_clock <clocks>:** The launch clock of the report are given by <ports/pins> (optional)
- **-to <ports/pins>:** The pins of the report are given by <ports/pins> (optional)
- **-to_clock <clocks>:** The latch clocks of the report are given by <ports/pins> (optional)
- **-hold:** Report hold timing paths (optional)
- **-summary:** Report setup and hold timing errors and total negative slack at the current speed grade (optional)

- **-file <file name>:** Redirect the report to the file <file name> (optional)
- **-app:** Append report to the end of the file (optional)

Example

```
sta_report_timing -n 5 -from abcd/Q0
```

See Also

- "Timing Analysis Tcl Commands" on page 432

sta_report_unconstrained

Command to report unconstrained terminals.

Syntax

```
sta_report_unconstrained [-n <number>] [-start] [-end]
[-io] [-clock_nets] [-file <file name>] [-app]
```

Arguments

- **-n <number>:** Report up to the number specified in <number> (optional)
- **-start:** Report unconstrained timing start point pins (optional)
- **-end:** Report unconstrained timing endpoint pins (optional)
- **-io:** Report unconstrained IOs (optional)
- **-clock_nets:** Report clock nets without a clock definition (optional)
- **-file:** File to which the report is redirected (optional)
- **-app:** Append report at the end of the file (optional)

Example

```
sta_report_unconstrained -file abc -start -app
sta_report_unconstrained
```

See Also

- "Timing Analysis Tcl Commands" on page 432

Other Commands

Design Object Tcl Commands

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.

The **get_info** commands are used for constraints to return a single object, or a collection of objects. You can get this command by using **help {get_*}**.

Radiant Software Design Object Tcl Command Descriptions

The following table provides a listing of all valid Radiant software design object-related Tcl command options and describes option functionality.

In the example below, <*target name*> is a regular expression that matches one object only.

Clock Object

SDC Collection	Description
get_clocks	Return clock objects.
all_clocks	Return a list of all clocks in the current design.

This section displays the usage information of the commands:

get_clocks

Commands to get a list of clocks in the current design.

Syntax

```
get_clocks [-regexp] [-nocase] [-of_objects|-of <objects>]
[<patterns>]
```

Arguments

- **-regexp:** This option indicates that patterns are full regular expressions (optional).
- **-nocase:** This option indicates that matching is case-insensitive (optional).

- **-of_objects|-of <objects>:** This option indicates that the command will get clocks of these objects (PORTs or PINs or NETs) (optional). When this argument is effective, other arguments won't work.
- **<patterns>:** This value specifies the patterns against which clock names are matched (optional). If no pattern is specified, set '*' as default.

Example

```
get_clocks -nocase -regexp clk\[0-9]+
get_clocks -of [get_ports clk]
```

- "Clock Object" on page 443

all_clocks

```
% help all_clocks
all_clocks -  Commands to get a list of all clocks in the
current design
  Usage:
    all_clocks
```

Radiant Software Clock Object Tcl Commands Example

This section illustrates and describes a few samples of Radiant clock object Tcl commands.

Example 1

<*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.

```
get_clocks <target name>
[get_clocks clock_fast]

[all_clocks]
```

Port Object

SDC Collection	Description
get_port_info	Return port object information.
get_ports	Return port objects.
all_inputs	Return a list of all input ports in the current design.
all_outputs	Return a list of all output ports in the current design.

get_port_info

Commands to return port info.

Syntax

```
get_port_info [-name] [-is_inout] [-is_input] [-is_output]
<port_object>
```

Arguments

- **-name**: This option indicates that the command will output the name of the specified port (optional).
- -**is_inout**: This option indicates that the command will output if the specified port is an inout port (optional).
- **-is_input**: This option indicates that the command will output if the specified port is an input port (optional).
- **-is_output**: This option indicates that the command will output if the specified port is an output port (optional).
- **<port_object>**: This value specifies one port object whose information will be output.

Example

The following example outputs whether port '-seg_out[0]' is an output port:

```
get_port_info -is_output [get_ports {seg_out[0]}]
```

See Also

- “Port Object” on page 445

get_ports

Commands to get a list of ports in the current design.

Syntax

```
get_ports [-hierarchical] [-hsc <separator>] [-regexp]
[-nocase] [-of_objects|-of <objects>] [-filter <expression>]
[<patterns>]
```

Arguments

- **-hierarchical|-hier:** This option indicates that the command will get ports from all levels of the design hierarchy (optional).
  Without this argument, the command will only get ports from the top of the design hierarchy.
- **-hsc separator**: Specifies a separator for hierarchical paths. Default is '/' (optional).
- **-regexp:** This option indicates that patterns are full regular expressions (optional).
- **-nocase:** This option indicates that matching is case-insensitive (optional).
- **-of_objects| -of <objects>:** This option indicates that the command will get ports of these objects (NETs, INSTANCES, or PINs) (optional).
  -of is an alias for -of_objects. When this argument is effective, other arguments won't work.
- **-filter <expression>:** This option filters ports based on property values (optional). The expression format supports:
  - Simple comparisons: PROPERTY_NAME == "value" or PROPERTY_NAME != "value"
  - Comparison operators: ==, !=, <, >, <=, >=, =~ (pattern matching)
  - Logical operators: && (AND), || (OR), ! (NOT)
  - Parentheses for grouping: (PROPERTY1 == "value1") && (PROPERTY2 == "value2")

String values can be enclosed in quotes or used without quotes. Numeric values can be used without quotes.

Wildcard pattern matching is supported:

- *: matches any sequence of characters
- ?: matches any single character

The =~ operator always uses wildcard pattern matching (equivalent to == with wildcards).

Special properties: NAME (object hierarchical name), DIRECTION (port direction: IN, OUT, INOUT, TRI), CLASS (always "PORT").

Examples:

- -filter {DIRECTION == "IN"}
- -filter {NAME =~ "clk*"}
- -filter {DIRECTION == "OUT" && NAME =~ "data*"}

  - -filter {NAME =~ "*clk*" && DIRECTION == "IN"} (combine NAME with DIRECTION)
- **<patterns>:** This value specifies the patterns against which net names are matched (optional). If no pattern is specified, set '*' as default.

Example

```
get_ports -nocase -regexp clk\[0-9]+
get_ports -of_objects [get_nets {clk*}]
get_ports -filter {DIRECTION == "IN"}
get_ports -filter {NAME =~ "*clk*" && DIRECTION == "IN"
```

See Also

- "Port Object" on page 445

all_inputs

Commands to get a list of all input ports in the current design.

Syntax

```
all_inputs
```

Example

```
The following example returns all input ports in the current
design:
all_inputs
```

See Also

- "Port Object" on page 445

all_outputs

Commands to get a list of all output ports in the current design.

Syntax

```
all_outputs
```

Example

```
The following example returns all output ports in the current
design:
all_outputs
```

See Also

- "Port Object" on page 445

Cell Object

SDC Collection	Description
get_cell_info	Return cell information.
get_cells	Return instance objects.

This section displays the usage information of the commands:

get_cell_info

```
% help get_cell_info
get_cell_info - Commands to return cell info
  Usage:
    get_cell_info [-name] [-inpins] [-outpins] [-pins]
<cell_object>
```

get_cells

Commands to get a list of cells in the current design.

Syntax

```
get_cells [-hierarchical] [-regexp] [-nocase] [-logical]
[-physical] [-placement] [-of_objects|-of <objects>] [-filter
<expression>] [<patterns>]
```

Arguments

- **-hierarchical|-hier:** This option indicates that the command will get cells from all levels of the design hierarchy (optional).
  Without this argument, the command will only get cells from the top of the design hierarchy.
- **-regexp:** This option indicates that patterns are full regular expressions (optional).
- **-nocase:** This option indicates that matching is case-insensitive (optional).
- **-of_objects| -of <objects>:** This option indicates that the command will get cells of these objects (PINs or NETs) (optional).
  When this argument is effective, other arguments won't work.
- **-logical:** This option indicates that the command will return logical instances instead of physical instances (optional).
  Physical instances are converted to logical instances via cross-reference when available.
  Cannot be used together with -physical option**.**

- **-physical:** This option takes a logical cell name as input and returns the associated physical instances (optional).
  Uses cross-reference database to find physical instances mapped to the logical name.
  Only meaningful when current view is Physical; returns empty if current view is Logical.
  Cannot be used together with -logical option.
- **-placement:** This option displays placement information for a physical instance in table format (optional).
  Shows the site where the instance is placed and all logical instances packed into each primitive slot (e.g., REG0, REG1, K0, K1) via cross-reference database.
  Requires exactly one cell to be matched; error is reported if more than one cell matches.
  Can be combined with -physical option: first converts logical name to physical instance, then shows placement.
  Cannot be used together with -logical option.
- **-filter <expression>:** This option filters cells based on property values (optional). The expression format supports:
  - Simple comparisons: PROPERTY_NAME == "value" or PROPERTY_NAME != "value"
  - Comparison operators: ==, !=, <, >, <=, >=, =~ (pattern matching)
  - Logical operators: && (AND), || (OR), ! (NOT)
  - Parentheses for grouping: (PROPERTY1 == "value1") && (PROPERTY2 == "value2")

String values can be enclosed in quotes or used without quotes. Numeric values can be used without quotes.

Wildcard pattern matching is supported:

- *: matches any sequence of characters
- ?: matches any single character

The =~ operator always uses wildcard pattern matching (equivalent to == with wildcards).

Special properties: NAME (object hierarchical name), PRIMITIVE_TYPE (primitive cell model name).

Examples:

- -filter {ECO_MEM_ID == "RAM_ICROEF_LAT"}
- -filter {ECO_MEM_ID == "RAM_ICROEF_*"}   (matches any value starting with RAM_ICROEF_)
- -filter {ECO_MEM_TYPE == EBR && OUTREG == USED}
- -filter {(PROPERTY1 == "value1") || (PROPERTY2 == "value2")}
- -filter {!PROPERTY_NAME}   (checks if property does not exist)
- -filter {NAME =~ "c_*i1"}   (pattern matching on object name)

   - -filter {NAME =~ "c_i1*" && PRIMITIVE_TYPE == LUT} (combine NAME and PRIMITIVE_TYPE)
   - -filter {PRIMITIVE_TYPE == "LUT*"} (filter by primitive type with wildcards)
- **<patterns>:** This value specifies the patterns against which cell names are matched (optional). If no pattern is specified, set '*' as default.

Example

```
get_cells -hierarchical -nocase -regexp i\[10-9]+
get_cells -of_objects [get_pins {i*/q}]
get_cells -of_objects [get_nets {clk*}]
get_cells -filter {ECO_MEM_ID == "RAM_ICROEF_LAT"}
get_cells -filter {NAME =~ c_i1*}
get_cells -logical
```

See Also

- "Cell Object" on page 448

Net Object

SDC Collection	Description
get_net_info	Return net information.
get_nets	Return net objects.

get_net_info

Commands to return net info.

Syntax

```
get_net_info [-name] [-pin] <net_object>
```

Arguments

- **-name**: This option indicates that the command will output the name of the specified net (optional).
- **-pin**: This option indicates that the command will list pin names of specified net (optional).
- **<net_object>**: This value specifies one net object whose information will be listed.

Example

The following example lists pin names of net 'CLK':

```
get_net_info -pin [get_nets CLK]
```

See Also

- "Net Object" on page 450

get_nets

Commands to get a list of nets in the current design.

Syntax

```
get_nets [-hierarchical] [-regexp] [-nocase] [-logical]
[-physical] [-of_objects|-of <objects>]  [-filter <expression>]
[<patterns>]
```

Arguments

- **-hierarchical|-hier:** This option indicates that the command will get nets from all levels of the design hierarchy (optional). Without this argument, the command will only get nets from the top of the design hierarchy.
- **-regexp:** This option indicates that patterns are full regular expressions (optional).
- **-nocase:** This option indicates that matching is case-insensitive (optional).
- **-of_objects | -of <objects>:** This option indicates that the command will get nets of these objects (PINs or INSTANCEs) (optional). When this argument is effective, other arguments will not work.
- **-logical:** This option converts physical net/signal names to their corresponding logical net names using the cross-reference database. The returned nets will be logical net objects/names (optional). Cannot be used together with -physical.
- **-physical:** This option takes a logical net/signal name as input and returns the associated physical nets (optional).
  Uses cross-reference database to find physical signals mapped to the logical name.
  Only meaningful when current view is Physical; returns empty if current view is Logical.
  Cannot be used together with -logical.
- **-filter <expression>:** This option filters nets based on property values (optional). The expression format supports:
  - Simple comparisons: PROPERTY_NAME == "value" or PROPERTY_NAME != "value"
  - Comparison operators: ==, !=, <, >, <=, >=, =~ (pattern matching)
  - Logical operators: && (AND), || (OR), ! (NOT)
  - Parentheses for grouping: (PROPERTY1 == "value1") && (PROPERTY2 == "value2")

String values can be enclosed in quotes or used without quotes. Numeric values can be used without quotes.

Wildcard pattern matching is supported:

- *: matches any sequence of characters
- ?: matches any single character

The =~ operator always uses wildcard pattern matching (equivalent to == with wildcards).

Special properties: NAME (object hierarchical name).

Examples:

- -filter {PROPERTY_NAME == "value"}
- -filter {PROPERTY_NAME =~ "pattern"}
- -filter {PROPERTY1 == "val1" && PROPERTY2 == "val2"}
- -filter {NAME =~ "clk*"}   (pattern matching on net name)
- -filter {NAME =~ "*data*" && PROPERTY_NAME == "value"} (combine NAME with other properties)

- **<patterns>:** This value specifies the patterns against which net names are matched (optional)**.** If no pattern is specified, set '*' as default.

Example

```
get_nets -hierarchical -nocase -regexp clk[10-9]+
get_nets -of_objects [get_pins {i*/i}]
get_nets -of_objects [get_cells {i*}]
get_nets -filter {PROPERTY_NAME == "value"}
get_nets clk_net -physical
get_nets phy_net -logical
```

See Also

- “Net Object” on page 450

Pin Object

SDC Collection	Description
get_pin_info	Return pin objects.
get_pins	Return pin information.

get_pin_info

Commands to return pin info.

Syntax

```
get_pin_info [-name] [-is_clock] [-is_input] [-is_output]
[-net] [-parent_cell] <pin_object>
```

Arguments

- **-name**: This option indicates that the command will output the name of the specified pin (optional).
- **-is_clock**: This option indicates that the command will output if the specified pin is a clock pin (optional).
- **-is_input**: This option indicates that the command will output if the specified pin is an input pin (optional).
- **-is_output**: This option indicates that the command will output if the specified pin is an output pin (optional).
- **-net**: This option indicates that the command will output the net name of the specified pin (optional).
- **-parent_cell**: This option indicates that the command will output the instance name of the specified pin (optional).
- **<pin_object>**: This value specifies one pin object whose information will be output.

Example

The following example outputs instance name of pin 'fifo_inst/state_0':

```
get_pin_info -parent_cell [get_pins fifo_inst/state_0]
```

See Also

- "Pin Object" on page 452

get_pins

Commands to get a list of pins in the current design.

Syntax

```
get_pins [-hierarchical] [-regexp] [-nocase] [-logical]
[-physical] [-of_objects|-of <objects>] [-filter <expression>]
[<patterns>]
```

Arguments

- **-hierarchical|-hier:** This option indicates that the command will get pins from all levels of the design hierarchy (optional).
  Without this argument, the command will only get pins from the top of the design hierarchy.

- **-regexp:** This option indicates that patterns are full regular expressions (optional).
- **-nocase:** This option indicates that matching is case-insensitive (optional).
- **-logical:** This option converts physical pin names to their corresponding logical pin names using the cross-reference database. The returned pins will be logical pin objects/names (optional).
  Cannot be used together with -physical.
  Note: In Physical view, if neither -logical nor -physical is specified, -physical behavior is the default.
- **-physical:** This option takes a logical pin name as input and returns the associated physical pins (optional).
  Uses cross-reference database to find physical pins mapped to the logical pin.
  Only meaningful when current view is Physical; returns empty if current view is Logical.
  Cannot be used together with -logical.
- **-of_objects| -of <objects>:** This option indicates that the command will get pins of these objects (NETs or INSTANCEs). When this argument is effective, other arguments won't work (optional).
- **-filter <expression>:** This option filters pins based on property values. The expression format supports (optional):
  - Simple comparisons: PROPERTY_NAME == "value" or PROPERTY_NAME != "value"
  - Comparison operators: ==, !=, <, >, <=, >=, =~ (pattern matching)
  - Logical operators: && (AND), || (OR), ! (NOT)
  - Parentheses for grouping: (PROPERTY1 == "value1") && (PROPERTY2 == "value2")

String values can be enclosed in quotes or used without quotes. Numeric values can be used without quotes.

Wildcard pattern matching is supported:

- *: matches any sequence of characters
- ?: matches any single character

The =~ operator always uses wildcard pattern matching (equivalent to == with wildcards).

Special properties: NAME (object hierarchical name).

Examples:

- -filter {PROPERTY_NAME == "value"}
- -filter {PROPERTY_NAME =~ "pattern"}
- -filter {PROPERTY1 == "val1" && PROPERTY2 == "val2"}
- -filter {NAME =~ "clk*"}   (pattern matching on net name)

  - -filter {NAME =~ "*data*" && PROPERTY_NAME == "value"} (combine NAME with other properties)

- **<patterns>:** This value specifies the patterns against which pin names are matched. If no pattern is specified, set '*' as default (optional).

Example

```
get_pins -hierarchical -nocase -regexp clk\[0-9]+
get_pins -of_objects [get_nets {clk*}]
get_pins -of_objects [get_cells {i*}]
get_pins -filter {PROPERTY_NAME == "value"}
get_pins counter/D -physical
get_pins UI/CLK -logical
```

See Also

- "Pin Object" on page 452

Wildcard Support

Multiple 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.

Note

1. Wildcard * can be used in any position of object name such as a*, *b, a*b etc.
2. Wildcard * cannot cross hierarchical separator /. For example get_cells sub1/* only returns sub1/sub2 and does not return sub1/sub2/sub1 or sub1/sub2/sub1/sub2.
3. When -hierarchical is set, it will search all levels of the design hierarchy starting at the current instance.

Engineering Change Order Tcl Commands

This section provides Engineering Change Order (ECO) extended Tcl command syntax, command options, and usage examples.

ECO Tcl Command Descriptions

The following table provides a listing of all valid Radiant software ECO-related Tcl command options and describes option functionality.

Table 51: ECO Tcl Commands

Command	Description
eco_config_block	Config general IP.
eco_config_memory	Update memory initial value.
eco_config_sysio	Config sysio setting.
eco_save_design	Save design in memory to disk.

eco_config_block

Configures general IP.

Syntax

```
eco_config_block -comp <comp_name> (<key=value>)..
```

Arguments

- **-comp <comp_name>**: Specifies the IP instance name.
- **(<key=value>)...**: Specifies attribute key and value.

Example

```
eco_config_ip -comp DPHY1 {"Speed Select"="2.5Gbps"}
```

See Also

- "Engineering Change Order Tcl Commands" on page 455

eco_config_memory

Updates memory initial value.

Syntax

```
eco_config_memory -mem_id <memory_id> (-init_file <mem_file> -
format HEX|BIN|ADDR) | -all_0 | -all_1
```

Arguments

- **-mem_id**: Specifies the memory ID.
- **-init_file**: Specifies the memory file path.
- **-format**: Specifies memory file format. The valid format is HEX, BIN or ADDR.
- **-all_0**: Config memory all data to zero.
- **-all_1**: Config memory all data to 1. Option '-init_file' '-all_0' and '-all_1' are mutually exclusive.

Example

```
eco_config_memory -mem_id {top} -all_0
eco_config_memory -mem_id {ram_dq} -all_1
eco_config_mem -mem_id {mem} -init_file {D:/mem/init_hex.mem} -
format HEX
eco_config_memory -mem_id
{large_ram_inst.lscc_lram_inst.genblk1.LRAM_0.LRAM_0.LRAM_inst}
-init_file {D:/memory/lram_eco/mem.bin} -format BIN
```

See Also

- "Engineering Change Order Tcl Commands" on page 455

eco_config_sysio

Config sysio setting.

Syntax

```
eco_config_sysio -comp <comp_name> (<key=value>)...
```

Arguments

- **-comp <comp_name>**: Specifies the instance name.
- **(<key=value>)...**: Specifies one or more sysio key and value settings.

Example

```
eco_config_sysio -comp {c[0]}
{diffdrive=NA;diffresistor=OFF;diff_invert=NA;drive=8;
early_io=ON;glitchfilter=OFF;hysteresis=NA;opendrain=OFF;
pullmode=NONE;slewrate=SLOW;termination=OFF;vref=OFF}
eco_config_sysio -comp {c[1]}
{diffdrive=NA;diffresistor=OFF;diff_invert=NA;drive=50RS}
```

See Also

- "Engineering Change Order Tcl Commands" on page 455

eco_save_design

Saves current design to disk.

Syntax

```
eco_save_design [-udb <udb_file>]
```

Arguments

- **-udb <udb_file>**: Specifies the output udb file (optional).

Example

```
eco_save_design -udb eco.udb
eco_save_design
```

See Also

"Engineering Change Order Tcl Commands" on page 455

sta_set_option

Commands to read database from udb file.

Syntax

```
sta_set_option [-name <logical or physical>] [-worst_delay]
```

Arguments

- **-name <logical or physical>:** This option controls the name used in reports (optional)
- **-worst_delay:** This option activates mixed temperature delay data (optional)

Example

```
sta_set_option -name physical
sta_set_option -worst_delay
```

See Also

- "Timing Analysis Tcl Commands" on page 432

Chapter 10

Advanced Topics

This chapter explains advanced concepts, features and operational methods for the Radiant software.

Shared Memory Environment

The Radiant software design environment uses a shared memory architecture. Shared memory allows all internal tool views to access the same image of the design at any point in time. Understanding how shared memory is being used can give you insight into managing the environment for optimum performance, especially when your design is large.

There is one shared database that contains the device, design, and constraint information in system memory.

Generating the hierarchy of your design uses an additional database separate from the primary shared memory database.

External tools referenced from within the Radiant software, such as those for synthesis and simulation, use their own memory in addition to what is used by the Radiant software.

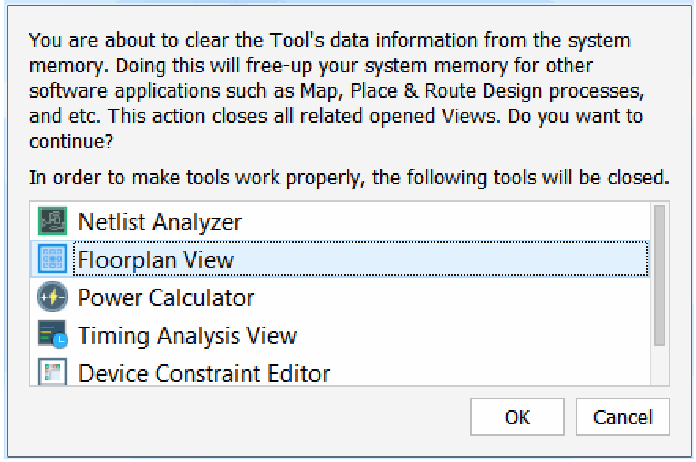
Because it is accessing shared memory, the initial tool view launch will take longer than the launch of subsequent views.

Clear Tool Memory

The “Clear Tool Memory command, available from the Tools menu, clears the device, design, and constraint 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 will be affected by clearing the tool memory, a confirmation dialog box will open to give you the opportunity to cancel the memory clear.

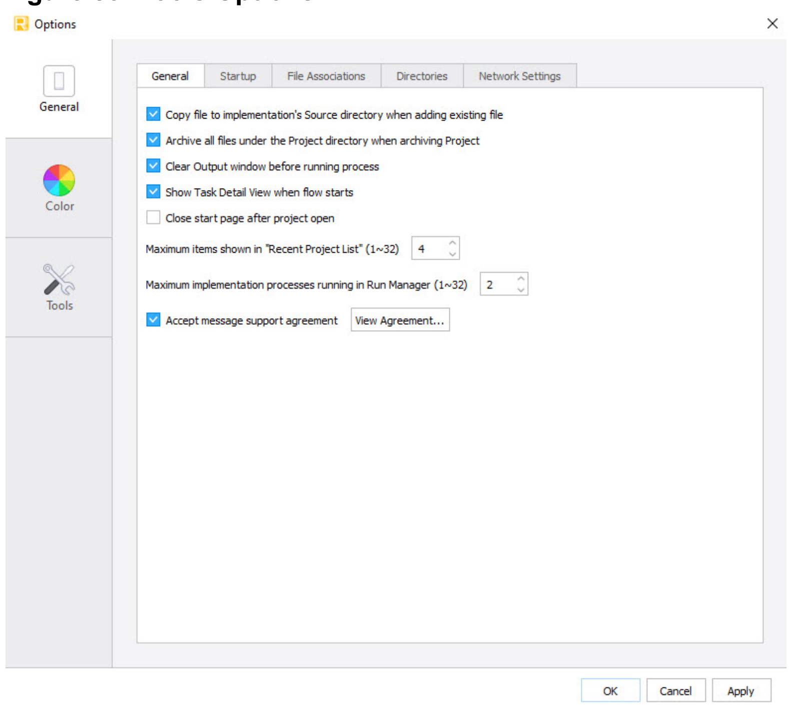
Figure 82: Clear Tool Memory Dialog



Environment and Tool Options

The Radiant software provides many environment control and tool view display options that enable you to customize your settings. Choose **Tools > Options** to access these options.

Figure 83: 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. You can also change the Theme color of the 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 set the constraint design rule check (DRC) time to real time, prior to saving or when launching a tool.

Batch Tool Operation

The core tools in the FPGA implementation design flow can all be run in batch mode using command-line tool invocation or scripts. For detailed information, refer to the Radiant Software Help. See **Reference Guides > Command Line Reference Guide**.

Tcl Scripts

The Radiant Extended Tcl language enables you to create customized scripts for tasks that you perform often in the Radiant software. Automating these operations through Tcl scripts not only saves time, but also provides a uniform approach to design. This is especially useful when you try to find an optimal solution using numerous design implementations.

Creating Tcl Scripts from Command History

A good first step in Tcl programming is to create a Tcl script by saving some command history and then modifying it as needed. This allows you to get started by using existing command information.

To create a Tcl command script using command history:

1. In the Tcl Console window, perform a *reset* command so that your script won't contain any of the actions that may have already been executed.

   ```
   reset
   ```

2. Open the project and perform the commands that you want to save as a script.
3. Optionally, enter the history command in the Tcl Console window to ensure that the commands you wish to save are in the console's memory. In the Tcl Console window type

   ```
   save_script <filename.tcl>
   ```

   The <filename.tcl> can be 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. A file name with an extension of .ext will not work.

   The **<filename.ext>** entry can be preceded with an absolute or relative path. Use the **"/"** (i.e. forward slash) symbol to delimit the path elements. If the path is not specified, the script is saved in the current working directory. The current working directory can be determined by using the TCL *pwd* command.
4. Navigate to your script file and use the text editing tool of your choice to make any necessary changes, such as deleting extraneous lines or invalid arguments.

In most cases, you will need to edit the script you saved and take out any invalid arguments or any commands that cannot be performed in the Radiant software environment due to a 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.

Creating Tcl Scripts from Scratch

Tcl commands can be written directly into a script file. You can use any text editor, such as Notepad or vi, to create a file and type in the Tcl commands.

Sample Tcl Script

The following Tcl example shows a simple script that opens a project, runs the entire design flow through the Place & Route process, and then closes the project. A typical script would probably contain more steps, but you can use this example as a general guideline.

```
prj_project open "C:/lscc/Radiant/counter/counter.rdf"
prj_run PAR -impl counter -forceAll
prj_project close
save_script c:/lscc/radiant/examples/counter/myscript2.tcl
```

Running Tcl Scripts

You can run scripts from the Radiant software integrated Tcl Console whether your project is opened or not. You can also run scripts from the external Tcl Console prompt window. The following example procedure uses the integrated Tcl Console and the sample Tcl script from the previous section:

To run a Tcl script that opens a project, runs processes and closes the project:

1. 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. If there are previously issued commands in the console, type **`reset`** in the console command line to refresh your session and clear out all previous commands.

   ```
   reset
   ```
4. Use the TCL *source* command to load and run your TCL script. Since it's the only argument, the *source* command requires 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 the following:

   ```
   source C:/lscc/radiant/<version_number>/examples/counter/
   myscript2.tcl
   ```
5. 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.

Project Archiving

A Radiant software project archive is a single compressed file (.zip) of your entire project. The project archive can contain all of the files in your project directory, or it can be limited to source-related files. When you use the **File > Archive Project** command, the dialog box provides the option to "Archive all files under the Project directory." When you select this option, the entire project is archived. When you clear this option, only the project's source-related files, including strategies, are archived. Many of these source-related files must be archived in order to achieve the same bitstream results for a fully implemented design.

Whichever archiving method you select, if your project contains source files stored outside the project folder, the remote files will be compressed under the remote_files subdirectory in the archive; for example:

```
<project_name>/remote_files/sources
<project_name>remote_files/strategies
```

When unarchiving, you must manually move the archived remote files to the original locations or the project will not work.

File Descriptions

This section provides a list of the file types used in the Radiant software, including those generated during design implementation. The Archive column indicates the files that must be archived in order to achieve the same bitstream results.

Table 52: Source Files

File Type	Definition	Function	Archive?
.fdc	FPGA Design Constraint file	Used for specifying design constraints for Synplify-Pro synthesis tool.	✓
.ipk	Radiant Software IP Package file.	Package file for Radiant software Soft IP.	
.ipx	Manifest file generated by IP Catalog.	Enables changes to be made to a module or IP Catalog.	✓
.ldc	Lattice Design Constraint file	Used for specifying timing constraints for LSE synthesis flow. The .ldc contents will be combined into design database file .udb.	✓
.pcf	Power Calculator project file	Stores power analysis results from information extracted from the design project.	✓
.pdc	Post-Synthesis constraint file.	Used for specifying post-synthesis constraints (timing/physical) for Lattice engines such as MAP and PAR.	✓
.rdf	Radiant Software Project file	Used for managing and implementing all project files in the Radiant software.	✓
.rva	Reveal Analyzer file	Defines the Reveal Analyzer project and contains data about the display of signals in Waveform View.	✓
.rvl	Reveal Inserter debug file	Defines the Reveal project and its modules, identifies the trace and trigger signals, and stores the trace and trigger options.	✓
.sdc	SDC constraints file	Used for specifying design-specific constraints for Synplify-Pro. SDC is replaced by the FDC format in but is still supported in the Radiant software.	✓
.spf	Simulation project file, a script file produced by the Simulation Wizard	Used for running the simulator for your project from the Radiant software.	✓

Table 52: Source Files (Continued)

File Type	Definition	Function	Archive?
.sty	Strategy file	Defines the optimization control settings of implementation tools such as logic synthesis, mapping, and place and route.	✓
.sv	SystemVerilog		
.v	Verilog source file	Contains Verilog description of the circuit structure and function.	✓
.vhd	VHDL source file	Contains VHDL description of the circuit structure and function.	✓
.xcf	Configuration chain file	Used for programming devices in a JTAG daisy chain.	✓

Table 53: IP Files

File Type	Definition	Function	Archive?
<*instName*>_bb.v	Verilog IP black box file	Provides the Verilog synthesis black box for the IP core and defines the port list.	✓
<*instName*>.cfg	IP parameter configuration file	Used for re-creating or modifying the core in the IP Catalog tool.	✓
<*instName*>.svg	Scalable Vector Graphics file	A graphic file used to display module/IP schematic and ports.	✓
<*instName*>_tmpl.v	Verilog template file	A template for instantiating the generated module. This file can be copied into a user Verilog file.	✓
<*instName*>_tmpl.vhd	VHDL module template file	A template for instantiating the generated module. This file can be copied into a user VHDL file.	✓
<*instName*>.v	Verilog module file	Verilog netlist generated by IP Catalog to match the user configuration of the module.	✓

Table 54: Implementation Files

File Type	Definition	Function	Archive?
.bgn	Bitstream generation report file	Reports results of a bit generation (bitgen) run and displays information on options that were set.	
.bin	Bitstream file	Used for SRAM FPGA programming.	
.ibs	Post-Route I/O buffer information specification file (IBIS)	Used for analyzing signal integrity and electromagnetic compatibility (EMC) on printed circuit boards.	
.mcs	PROM file	Used for SRAM FPGA programming.	
.mrp	Map Report file	Provides statistics about component usage in the mapped design.	

Table 54: Implementation Files (Continued)

File Type	Definition	Function	Archive?
.pad	Post-Route PAD report file	Lists all programmable I/O cells used in the design and their associated primary pins.	
.par	Post-Route Place & Route report file	Summarizes information from all iterations and shows the steps taken to achieve a placement and routing solution.	
.sso	Post-PAR SSO analysis file	Reports the noise caused by simultaneously switching outputs.	
.tw1	Post-Map Timing analysis file	Estimates pre-route timing.	
.twr	Post-PAR Timing analysis file	Reports post-route timing.	
.vo	Post-Route Verilog simulation file	Used for post-route simulation.	
<*Design name*>_vo.sdf	Post-Route SDF simulation file for Verilog	Used for timing simulation.	
.vm	Synthesized netlist file	Netlist file generated by the Radiant software Synthesis tools.	
.udb	Unified Design Database file	Compiled from HDL design source. It may contain both design netlist and constraints.	

Revision History

The following table gives the revision history for this document.

Date	Version	Description
06/26/2026	2026.1	Updated to reflect changes in Radiant 2026.1 software.
12/11/2025	2025.2	Updated to reflect changes in Radiant 2025.2 software.
06/26/2025	2025.1	Updated to reflect changes in Radiant 2025.1 software.
03/31/2025	2024.2.1	Updated to reflect changes in Radiant 2024.2.1 software.
12/20/2024	2024.2	Updated to reflect changes in Radiant 2024.2 software.
06/28/2024	2024.1	Updated to reflect changes in Radiant 2024.1 software.
03/29/2024	2023.2.1	Updated to reflect changes in Radiant 2023.2.1 software.
11/24/2023	2023.2	Updated to reflect changes in Radiant 2023.2 software.
06/26/2023	2023.1	Updated to reflect changes in Radiant 2023.1 software.
02/9/2023	2022.1.1	Updated to reflect changes in Radiant 2022.1.1 software.
11/21/2022	2022.1	Updated to reflect changes in Radiant 2022.1 software.
06/8/2022	3.2	Updated to reflect changes in Radiant 3.2 software.
03/29/2022	3.1.1 Update	Updated to reflect changes in Radiant 3.1.1 MachXO5-NX Device Update.
03/22/2022	3.1.1	Updated to reflect changes in Radiant 3.1.1 software.
12/7/2021	3.1	Updated to reflect changes in Radiant 3.1 software.
06/14/2021	3.0	Updated screen captures. Updated Chapter 5, Pre-Synthesis Constraint File.
04/5/2021	3.0	Updated to reflect changes in Radiant 3.0 software.
10/20/2020	2.2	Updated to reflect changes in Radiant 2.2 software, including replacing the Active-HDL simulator with Mentor ModelSim.
07/10/2020	2.1.1	Updated to reflect changes in Radiant 2.1.1 software.
05/7/2020	2.1	Updated to reflect changes in Radiant 2.1 software.
11/27/2019	2.0	Updated to reflect changes in Radiant 2.0 software. Removed Appendix A, Reveal User Guide. Starting with Radiant software v2.0, this is now a stand-alone user guide. Removed Appendix B, Programming Tools User Guide. Starting with Radiant software v2.0, this is now a stand-alone user guide.

Date	Version	Description
03/25/2019	1.1	Updated to reflect changes in Radiant 1.1 software.
02/08/2018	1.0	Initial release.