Libero IDE v6.1 User's Guide

Actel®

Actel Corporation, Mountain View, CA 94043

Printed in the United States of America

Part Number: 5029120-10

Release: October 2004

Trademarks

Table Of Contents

What's New in Libero IDE v6.1

The following are highlights of new features for Libero IDE v6.1. For a more complete list, see the Libero IDE v6.1 release notes.

New help topics are highlighted by a red asterisk in the Table of Contents, as in the What's new in Libero IDE v6.1 topic.

Libero IDE v6.1 introduces software support for Actel's newest FPGA families, ProASIC3 and ProASIC3E. This release supports the following devices and packages:

A3PE600 600k Gate Enhanced Family Device:

- 208 PQFP
- 256 FBGA
- 484 FBGA

A3P250 250k Gate Device:

- 144 FBGA
- 208 PQFP

Software Features:

- ACTgen support for
- Clock Conditioning Capability (CCC) and PLL definitions via the Visual PLL Core Wizard
- Dual-Port RAM and FIFO configurations
- 1kbit user Flash ROM (FROM)
- FlashROM Simulation Flow
- Synplify Synthesis AE provides optimized performance and utilization
- Import of PDC (Physical Design Constraints) and SDC (Synopsys Design Constraints)
- Multiple Compile options: combine registers with I/Os, promote/demote global spines, automatic global promotion, local clock
- maximum shared instances and lock clock buffer fan out selection
- Enhanced Compile Report
- MultiView Navigator, ChipPlanner, PinEditor, and Netlist Viewer enhancements; I/O Attribute Editor supports the wide range of I/O choices in ProASIC3/E, Quadrant region creation
- Timing-Driven place-and-route results in optimized device performance in a push button manner; Incremental Place-and-Route
- Timer supports SDC constraints and pre/post-timing analysis
- Post layout power analysis
- Timing Back-Annotation flow

Synapticad WaveFormer Lite

- Reactive testbenches
- VCD support

Libero IDE Design Flow

The Libero Design Flow consists of six steps:

Step One - Design Creation

Plan out your design and enter it as either HDL (VHDL or Verilog), structural schematic, or mixed-mode (schematic and RTL).

Step Two - Design Verification - Functional Simulation

After you have defined your design, you must verify that it functions the way you intended. After creating a testbench using WaveFormer Lite, use the Model*Sim* VHDL or Verilog simulator to perform functional simulation on your schematic or HDL design.

Step Three - Synthesis/EDIF Generation

A design must be synthesized if the design was created using VHDL or Verilog. Use Synplify AE or Synplify Pro from Synplicity to generate your EDIF netlist. You can re-verify your design "post-synthesis" using the VHDL or Verilog Model*Sim* simulator used in step two.

While all RTL code must be synthesized, pure schematic designs are automatically "netlisted" out via the Libero tools to create a structural VHDL or structural Verilog netlist.

Step Four - Design Implementation

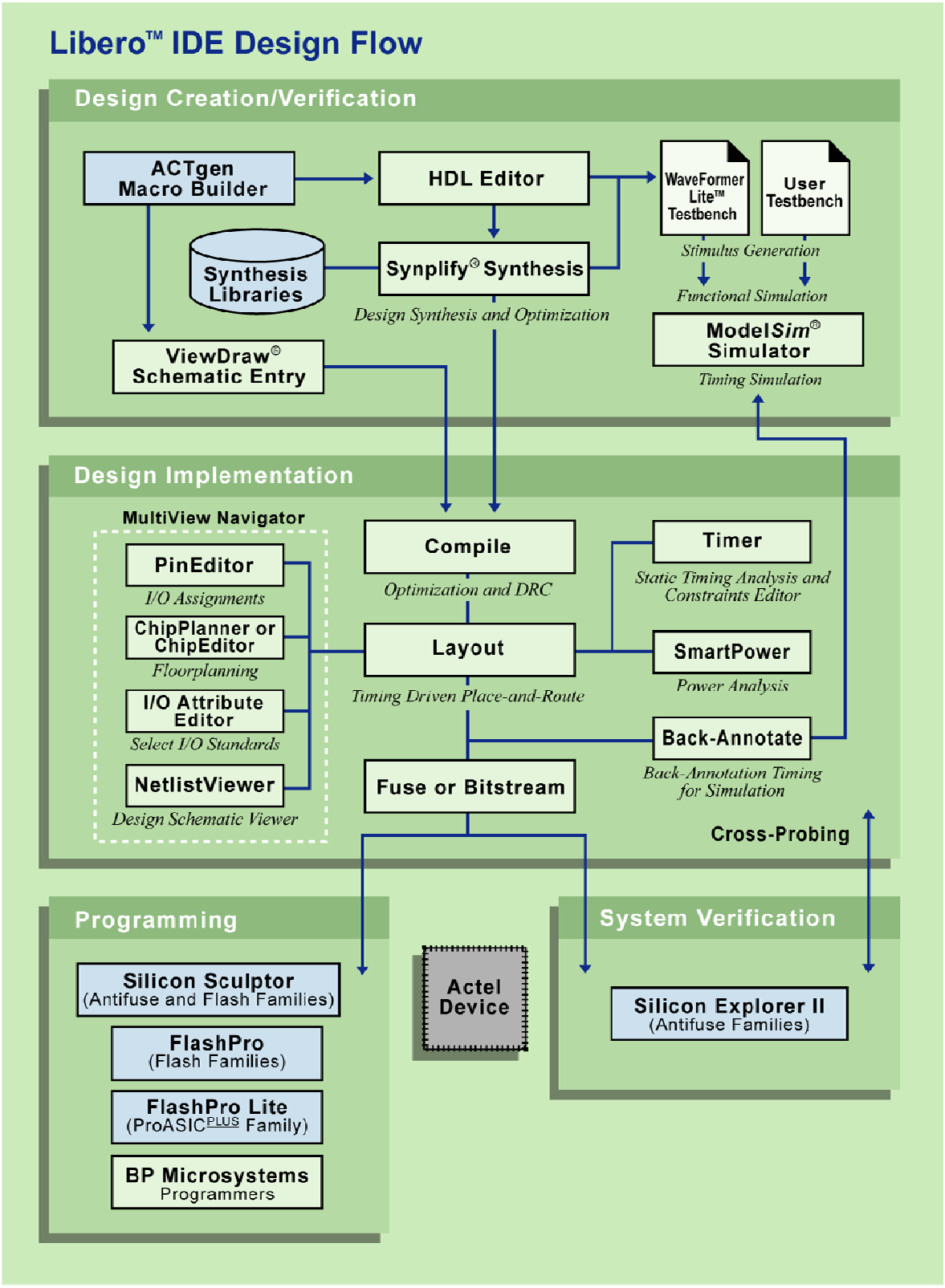
After you have functionally verified that your design works, the next step is to implement the design using the Actel Designer software. The Designer software automatically places and routes the design and returns timing information. Use the tools that come with Designer to further optimize your design. Use Timer to perform static timing analysis on your design, ChipEditor or ChipPlanner to customize your I/O macro placement, PinEditor for I/O customization, SmartPower for power analysis, and NetlistViewer to view your netlist.

Step Five - Timing Simulation

After you are done with design implementation, you can verify that your design meets timing specifications. After creating a test bench using WaveFormer Lite, use the Model*Sim* VHDL or Verilog simulator to perform timing simulation.

Step Six - Device Programming

Once you have completed your design, and you are satisfied with the timing simulation, create your programming file. Depending upon your device family, you need to generate a Fuse or Bitstream programming file.


Libero™ IDE Design Flow
Design Creation/Verification
ACTgen Macro Builder
HDL Editor
WaveFormer Lite™ Testbench
User Testbench
Stimulus Generation
Synthesis Libraries
Synplify® Synthesis
Functional Simulation
Design Synthesis and Optimization
ModelSim® Simulator
ViewDraw® Schematic Entry
Timing Simulation
Design Implementation
MultiView Navigator
Compile
Timer
PinEditor
I/O Assignments
Optimization and DRC
Static Timing Analysis and Constraints Editor
ChipPlanner or ChipEditor
Floorplanning
Layout
SmartPower
Power Analysis
I/O Attribute Editor
Select I/O Standards
Timing Driven Place-and-Route
Back-Annotate
NetlistViewer
Design Schematic Viewer
Fuse or Bitstream
Back-Annotation Timing for Simulation
Cross-Probing
Programming
System Verification
Silicon Sculptor (Antifuse and Flash Families)
Actel Device
Silicon Explorer II (Antifuse Families)
FlashPro (Flash Families)
FlashPro Lite (ProASICPLUS Family)
BP Microsystems Programmers

Creating a New Libero Project

Use the New Project Wizard to create a new Libero project.

To create a new project:

1. From the **File** menu, select **New Project**. The New Project Wizard starts.
2. Follow the instructions in the Wizard and click **Finish** when done.

You must select a new family in order to complete the New Project Wizard and create a new project.
You may set the die and package now, or do it later in the design flow.

Opening Your Libero Project

Libero IDE opens your most recent project automatically. You can change your default startup preferences in the Startup tab.

To open a different project in Libero IDE:

1. From the **File** menu, select **Close Project**.
2. From the **File** menu, select **Open Project** or **New Project**. If you create a new project Libero IDE starts the **New Project Wizard**.

Tip: The last five saved projects are available from the File menu. From the **File** menu, select **Recent Projects**, and then select the project to open.

Tip: You can open an existing project by double-clicking the .prj file or dragging the .prj file over the Libero IDE desktop icon.

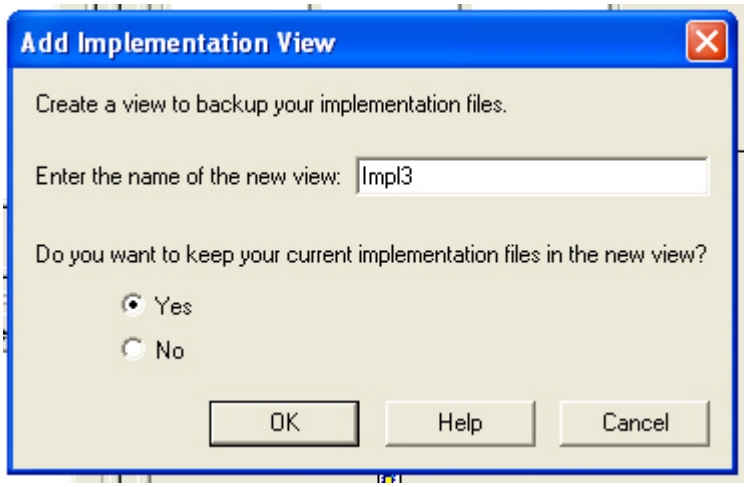
Project Implementations in Libero IDE

Implementations enable you to save different implementation files for individual projects. Use implementations to test different layout runs for a particular project.

You can check your layout results for each implementation view; to do so, create as many implementations as you wish, and select them from the Current implementation drop-down menu (available in the Implementation Toolbar) and run layout.

To create a new implementation:

Click the New Implementation button on the Implementation toolbar, or from the Implementations menu, select Add. The Add Implementations dialog box appears, as shown in the figure below.



You can use your new implementation view to backup your current implementation files. Enter the name of your new view, and choose to keep your current implementation files or revert to the Libero defaults. Click OK to continue.

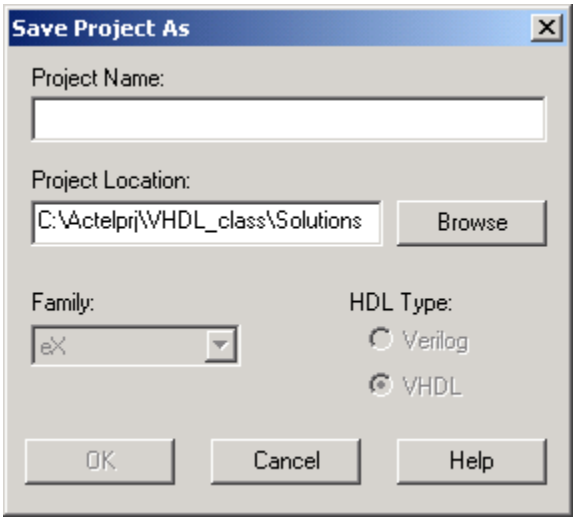
Your new implementation view appears in the list of Current Implementations on the Implementations toolbar.

Saving a Project with a New Name

Your project is saved when you close the project. To save the project with another name, use the **Save Project As** command.

To save the project with a new name:

1. From the **File** menu, select **Save Project As**. The Save Project As dialog box opens.



Save Project As Dialog Box

2. Enter a new project name.
3. Enter a new project location, or click **Browse** to specify a new location.

4. Click **OK**.

Closing and Exiting

Your project is automatically saved when closed. To explicitly save your project, use File -> Save Project. To save it with another name, use the Save Project As command.

To close a project, from the **File** menu, select **Close Project**.

To exit Libero IDE, from the **File** menu, select **Exit**.

Project Sources

Project sources are any design files that make up your design. These can include schematics, HDL files, simulation files, testbenches, etc. Anything that describes your design or is needed to program the device is a project source.

Source files appear in the Design Flow window. The Design Hierarchy tab displays the structure of the design modules as they relate to each other, while the File Manager tab displays all the files that make up the project.

The design description for a project is contained within the following types of sources:

Schematics

HDL Files (VHDL or Verilog)

One source file in the project is the top-level source for the design. The top-level source defines the inputs and outputs that will be mapped into the devices, and references the logic descriptions contained in lower-level sources. The referencing of another source is called an *instantiation*. Lower-level sources can also instatntiate sources to build as many levels of logic as necessary to describe your design.

Some project sources can be imported.

Sources for your project can include:

Source	File Extension
Schematic	*.1-9
Verilog Module	.v
VHDL Entity	.vhd
ACTgen Macro	.gen
Testbench	.vhd
Stimulus	.tim
Programming Files	.afm, .prb

New Files

You can create new files from Libero. New file types include:

Schematic

ACTgen macro

VHDL Entity

VHDL Package file

Stimulus

Stimulus HDL file

To create a new file:

1. From the **File** menu, select **New**.
2. Select the File type and type a name.
3. Click **OK**. The appropriate application starts. The saved file is added to your Libero project.

Importing Files

Anything that describes your design, or is needed to program the device, is a project source. These may include schematics, HDL files, simulation files, test benches, etc. Import these source files directly into your Libero project.

To import a file:

1. From the **File** menu, select **Import Files**.
2. In **Files of type**, select the file type.
3. In **Look in**, navigate to the drive/folder where the file is located.
4. Select the file to import and click **Open**.

Note:

Keep and import your VHDL package and behavioral and structural VHDL source files separately. Do not place your VHDL package into your source file.

You cannot import a Verilog File into a VHDL project and vice versa.

File Types for Import

File Type	File Extension
ViewDraw Symbol	*.1-9
ViewDraw Schematic	*.1-9
Behavioral and Structural VHDL	.vhd, .vhdl
VHDL Package	.vhd, .vhdl
ACTgen Macro	.gen
Verilog Include	.h
Behavioral and Structural Verilog	.v
Stimulus	.vhd, .vhdl, .v
EDIF Netlist	.edn

Libero IDE File types

When you create a new project in the Libero IDE it automatically creates new directories and project files.

Depending on your project preferences and the version of Libero IDE you installed, Libero IDE creates *actgen*, *constraint*, *designer*,*hdl*, *package*, *phy_synthesis*, *simulation*, *stimulus*, *synthesis*, and *viewdraw* directories for your project.

The toplevel directory (<project_name>) contains your PRJ file; only one PRJ file is enabled for each Libero IDE project.

***actgen* directory -** GEN files and LOG files from generated ACTgen cores

***constraint* directory -** All your constraint files (SDC, PDC, GCF, DCF, etc.)

***designer* directory -** ADB files (Actel Designer project files), -_ba.SDF, _ba.v(hd), STP, PRB (for Silicon Explorer), TCL (used to run designer), impl.prj_des (local project file relative to revision), designer.log (logfile)

***hdl* directory -** all hdl sources. *.vhd if VHDL, *.v and *.h if Verilog

package directory - VHD files

***phy_synthesis* directory -** _palace.edn, _palace.gcf, palace_top.rpt (palace logfile) and other files generated by PALACE

***simulation* directory -** meminit.dat, modelsim.ini files

***stimulus* directory -** BTIM and VHD stimulus files

***synthesis* directory -** *.edn, *_syn.prj (Synplify log file), *.psp (Precision project file), *.srr (Synplify logfile), precision.log (Precision logfile), exemplar.log (Leonardo logfile), *.tcl (used to run synthesis) and many other files generated by the tools (not managed by Libero IDE)

viewdraw directory - viewdraw.ini

Saving Files

Files and projects are saved when you close them.

To save an active file:

1. From the **File** menu, select **Save**, **Save As**, or **Save All**.
2. Click the **Save**  button in the toolbar.

Saving Files in Libero IDE

If you want to back up your Libero PRJ, Designer ADB, or other Libero IDE project files, create a new implementation of your design. Do not use the Save As function in Designer. Instead, use Add Implementation in the Libero IDE.

Deleting Files

Files can be deleted from the current project or from the disk.

To delete a file from the project:

3. Select the **File Manager** tab in the Design Explorer window.
4. **Right-click** the file and select **Delete from Project**. The file remains on your disk.

To delete a file from your project and the disk:

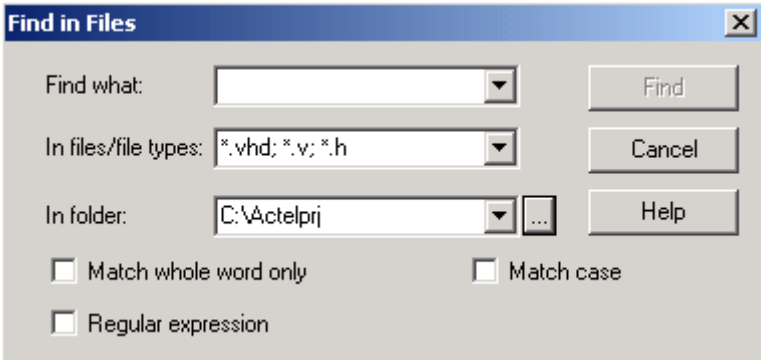
5. Select the **File Manager** tab in the Design Explorer window.
6. **Right-click** the file and select **Delete from Disk and Project**. The file is deleted from your disk and is no longer part of any project.

Finding Files

Use the Find In Files dialog box to search for files.

To find a file:

1. From the **Edit** menu, select **Find**. The Find In Files dialog box appears.



Find In Files

2. Select the properties for your search.

   Find what: Type a word string in the Find what text field.

   In files/file types: Select a file type.

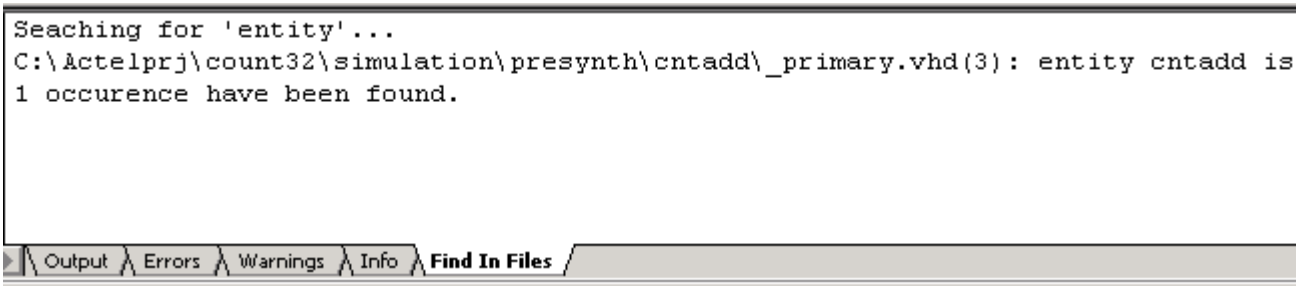
   In folder: Select a folder.

   Match whole word only: Select to match the whole word only.

   Regular expression: Select to recognize Microsoft Word-style expressions.

   Match case: Select to search for case-sensitive occurrences of a word or phrase. This limits the search so it only locates text that matches the upper- and lowercase characters you enter.

3. Click **Find**. The results appear in the Find In Files tab in the log window. Click the file name in the log window to open the file.



Found In Files Tab in Log Window

Finding Modules

You can search for modules in the Design Hierarchy with the Find Module feature.

To find a module in the Design Hierarchy:

1. Click the Find Module icon or from the **Edit** menu, select **Find Module**.
2. Enter the name of the module you wish to find.
   Libero IDE displays the module in the Design Hierarchy. If Libero does not display the module, you may have typed the module name incorrectly, or the module you are looking for may have a different name.

Reserved Actel keywords

See the online help for a complete list of reserved keywords.

Libero Project Options

Use the Options dialog box to specify your project settings for the currently open project.

From the **Options** menu, select **Project Settings** to open the Options dialog box. View and edit the preferences and Click **OK**. To revert to the default settings, click **Default**.

Options include:

Project Settings

Simulation

Programming

Libero Project Settings

Use the Project Settings tab in the Options window to change the die and package for your project.

To change the die and/or package:

1. From the **Tools** menu, select **Options**. The Options dialog box appears .
2. Select a die and package from the list and click **OK**.

Programming

If you did not install FlashPro as part of the Libero installation process, you can use the Programming tab in the Options dialog box to integrate FlashPro with your project.

To integrate FlashPro with your project:

1. From the **Options** menu, select **Project Settings**.
2. Click the Programming tab.
3. Fill in the options:

   Location: Specify the location of the FlashPro executable.

   Additional Parameters: Additional parameters of the command line.

   Jar file location: Specify the location of the FlashPro jar file.
4. Click **OK**.

Setting Your Project Profile

Each Libero IDE project can have a different profile, enabling you to integrate different tools with different Libero projects.

To set or change your project profile:

1. From the **Options** menu, select **Profile**.

   To add a tool: Click **Add** and select which type of tool (synthesis, stimulus, or simulation). Fill out the tool profile and click **OK**.

   To change a tool profile: After selecting the tool, click to **Edit** to select another tool, change the tool name, or change the tool location.

   To remove a tool from the project:After selecting a tool, click Remove.

   To restore the tool profiles shipped with Libero: Click Restore Defaults.

2. When you are done, click **OK**.

VHDL Package Files Organization

It is important to preserve the VHDL package file sequence if your VHDL package files are interdependent.

To specify the VHDL package file order:

1. From the **Options** menu, select **Package File Organization**. The VHDL Package Files dialog box appears.
2. Use the up and down arrows to specify the order, or drag the files into order.
3. Select Simulation or Synthesis if you want the file included when simulation or synthesis is run.
4. Click **OK**.

Verilog Header File Organization

It is important to preserve the Verilog header file sequence if your Verilog header files are interdependent.

To specify the Verilog header file order:

5. From the **Options** menu, select **Header File Organization**. The Verilog Header Files dialog box appears.
6. Use the up and down arrows to specify the order, or drag the files into order.
7. Select Simulation or Synthesis if you want the file included when simulation or synthesis is run.
8. Click **OK**.

Setting Preferences

Use the Preferences dialog to customize Libero to your needs.

To set your preferences:

1. From the **File** menu, select **Preferences**.
2. Specify your preferences on each of the tabs.

Updates Tab

Proxy Tab

Startup Tab (File association)

Log Window

Text Editor

3. Click **OK**.

Note: These preferences are stored on a per-user basis. These preferences are not project specific.

Updates

Actel strongly recommends that you check at least once a week for fixes, updates, and enhancements for your Actel software.

Note: The version checking feature is not available for Linux.

The Updates tab in the Preferences dialog box allows you to set your automatic software update preferences.

To set your automatic software update preferences:

1. From the **File** menu, select **Preferences** and **Updates**.
2. Choose one of the following options:

   Automatically check for updates at startup: Select to be notified of updates when you start Designer.

   Remind me to check for updates at startup: Select to be asked if you want to check for a software update when you start Designer.

   Do not check for updates or remind me at startup: Select if you do not want to check for software updates at startup.

To manually check for software updates, from the **Help** menu, select **Check for Software Updates**.

3. Click **OK**.

Note: This feature requires an internet connection.

Setting Your Proxy

An FTP connection is used to update some data files. Use the Proxy tab in the Preferences dialog box to enter your proxy name if you use a proxy server.

1. From the **File** menu, select **Preferences.**
2. Click the **Proxy** tab.
3. If you use a Proxy server, select the check box and enter the name.
4. Click **OK** to dismiss the Preferences dialog box.

Startup Tab

Several programs, including Libero, create files with the .prj extension. If you want Libero to start whenever you double-click on a PRJ file, you need to set up Libero as the default editor for PRJ files.

To make Libero the default editor for .prj files:

1. From the File menu, select **Preferences**.

2. Click the **Startup** tab.
3. Select the check box to associate Libero with PRJ files.

To open the most recently used project at startup:

1. From the File menu, select **Preferences**.
2. Click the **Startup** tab.
3. Select the check box to open the most recently used project at startup.

Setting Your log Window Preferences

Errors, Warnings, and Informational messages are color-coded in the log window. You can change the default colors by using the log Window tab in the Preferences dialog box.

To change colors in the log window:

1. From the **File** menu, select **Preferences**.
2. Click the **Log Window** tab in the Preferences Dialog Box.
3. Select your new default colors and click **OK**.

The default color settings for the log window are:

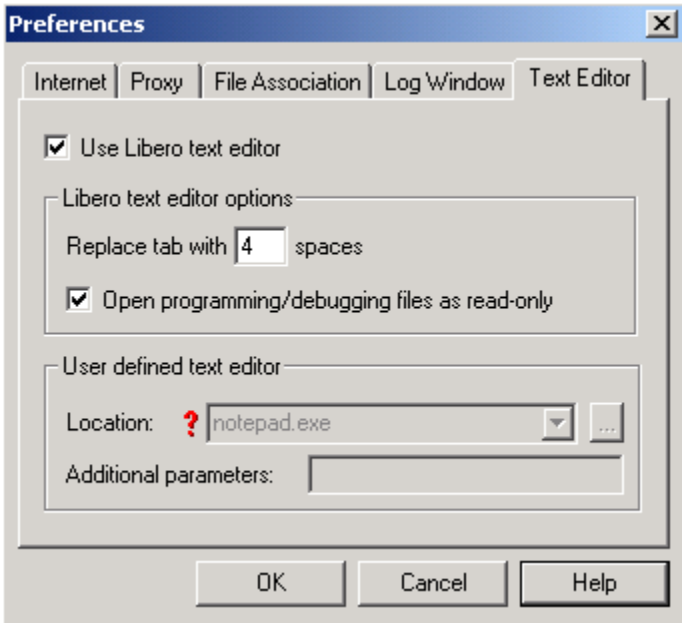
Message Type	Colors
Errors	Red
Warnings	Light Blue
Informational	Black
Linked	Dark Blue

Text Editor

You can use the Libero HDL text editor or another text editor.

To set your text editor preferences:

1. From the **File** menu, select **Preferences**.
2. Click Text Editor.



Preferences: Text Editor

3. Set your options and click **OK**.

Libero text editor options

Use Libero text editor: Select to use the Libero HDL text editor.

Replace tab with spaces: Enter the number of spaces you want entered when using the tab key.

Open programming/debugging files as read-only: Select to specify read-only permission to .stp and .prb files.

User defined text editor

User defined text editor: Deselect use Libero text editor to activate this area. Enter the .exe location of the text editor.

Additional parameters: Use to specify other settings to pass to the text editor. Typically, it is not necessary to modify this field.

Libero's Project Manager

Libero's Project Manager workspace integrates the needed design tools, streamlines the design flow, manages all design and log files, and passes necessary design data between tools.

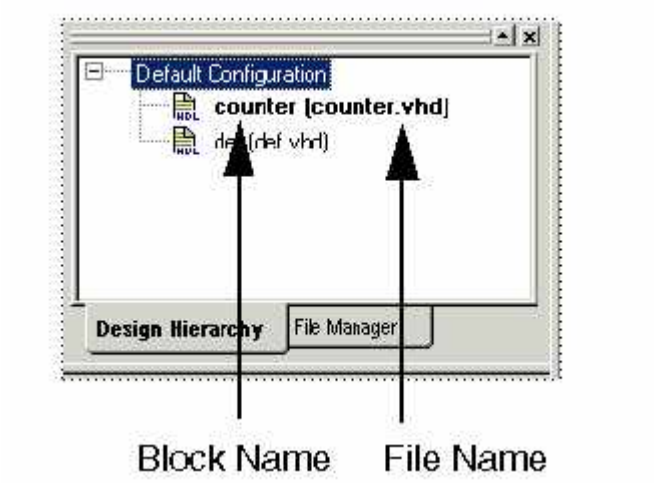
The Design Explorer Window, located in the upper left, consists of the Design Hierarchy and File Manager windows. Below is the Design Flow Window. The HDL Editor fills up the right side of the Project Manager, while the Log Window is found at the bottom.

Libero also includes toolbars and menus.

Design Hierarchy

The Design Hierarchy tab displays a hierarchical representation of the design based on the source files in the project. Libero IDE continuously analyzes and updates source files and updates the hierarchy. The Design Hierarchy tab displays the structure of the design modules as they relate to each other.

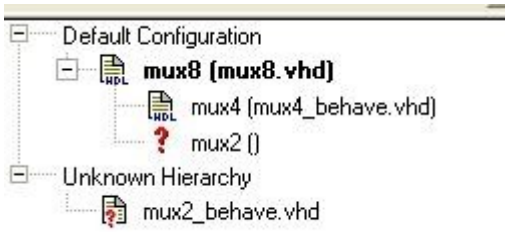
The file name (the file that defines the block) appears next the to block name in parentheses.



Design Hierarchy - Block and File Names

More information about the block can be found by right-clicking it and selecting **Properties**. The Block Properties dialog box displays block properties including, file path, created date, and last modified date.

Files that cannot be read by Libero are identified with red question marks.



Design Hierarchy - Unknown Hierarchy

All integrated source editors are linked with Libero's Project Manager. If a source is modified and the modification changes the hierarchy of the design, the Design Hierarchy automatically updates to reflect the change.

If you want to update the design hierarchy,from the **Edit** menu, select **Refresh** .

To open a source:

Double-click a source in the Design Hierarchy to open it. Depending on the block type and design state, several possible options are available from the right-click menus.

File Manager

The folders in the File Manager are like the ones in the Windows Explorer. These folders reside on your computer and support relative paths - paths from the project folder to files in your folders. This folder structure makes it easier to organize your files.

The File Manager window displays all the files associated with your project. Files are grouped by type.

block symbol files

schematic files

VHDL package

HDL

ACTgen cores

design implementation files

synthesis files

physical synthesis files

stimulus

Right-clicking a file in the File Manager provides a menu of available options specific to the file type. You can also delete files from the project by selecting **Delete from Project** from the right-click menu.

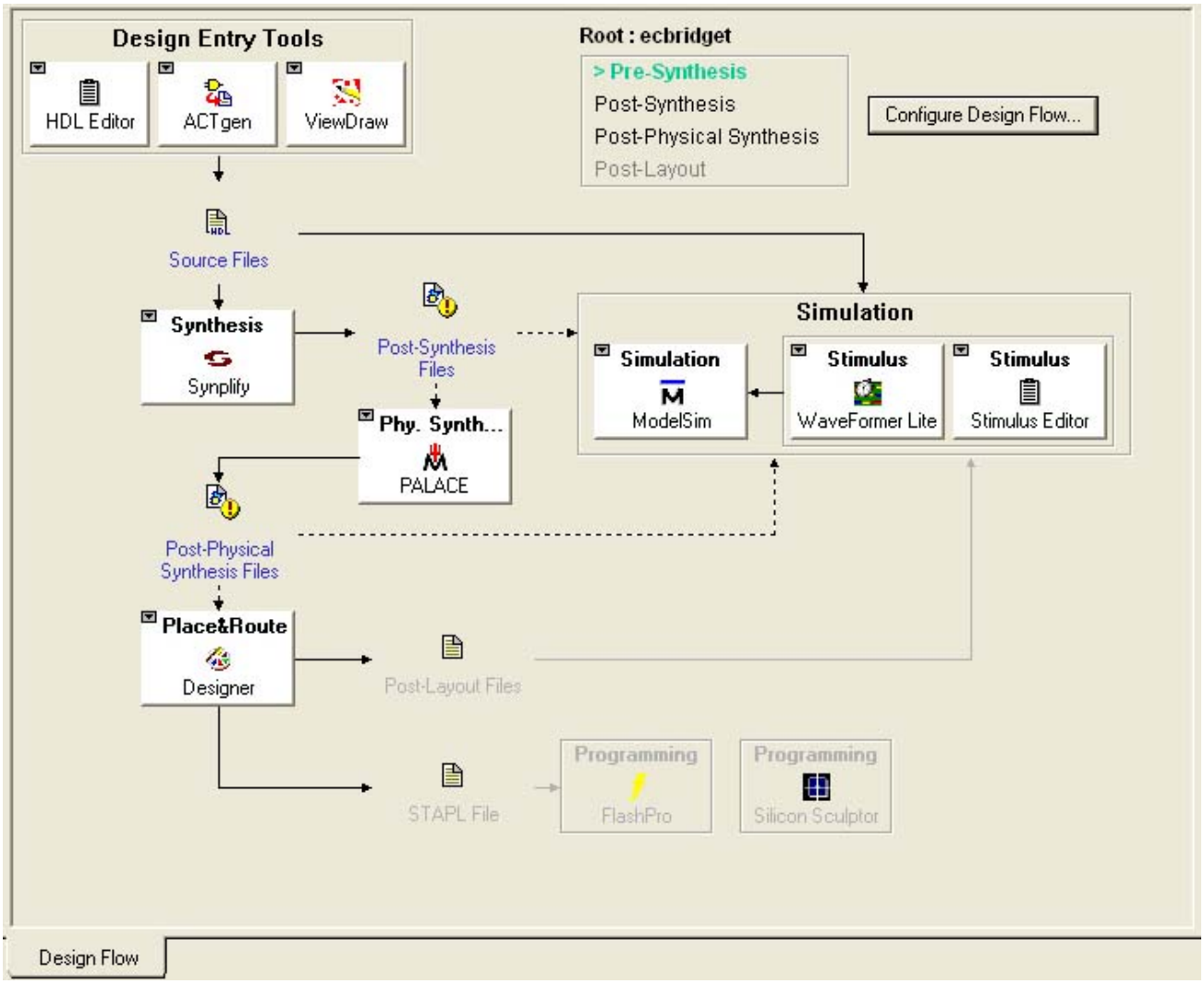
Tip: You can drag files in the File Manager to re-order them.

Design Flow Window

The Design Flow window displays all available tools involved in the design process (see the figure below).

This window shows you the current state of your design by activating and highlighting tools at appropriate times in the design process, while graying out tools that are not yet available. Green checks indicate successfully completed steps.

Click a tool to start it. Right-click a tool to access the right-click menu, which provides all the available processes you can start with the tool.



Design Flow Window

HDL Editor Window

The HDL Editor targets the creation of HDL code. It supports VHDL and Verilog with color, highlighting keywords for both HDL languages.

Note: To avoid conflicts between changes made in your HDL files, Actel recommends that you use one editor for all of your HDL edits.

Log Window

Colors and Symbols

For ProASIC and ProASIC^{PLUS} families, the log window displays notes and warnings. For Antifuse families, the log window displays Error, warning, and informational messages. Messages are represented by symbols and color-coded. The default colors are:

Type	Color
Error	Red
Warning	Blue
Information	Black

The colors can be changed by using the Preferences dialog box.

Output, Error, Warning, and Info Tabs

The Output tab displays all messages. Use the errors, warnings, or informational tabs to filter for just those messages. The views within the error, warnings, and info displays are reset when a new command is executed or a new design is opened. To see a complete history of your design session, click the output tab.

Linked Messages

Error and warning messages that are dark blue and underlined are linked to online help to provide you with more details or helpful workarounds. Click them to open online help.

Using the HDL Editor

The HDL Editor is a text editor designed for editing HDL source files. In addition to regular editing features, the editor provides a syntax checker.

You can have multiple files open at one time in the HDL Editor workspace. Click the tabs to move between files.

Editing

Editing functions are available in the Edit menu. Available functions include cut, copy, paste, find and replace. These features are also available in the toolbar.

Saving

You must save your file to add it to your Libero project. Click **Save** in the **File** menu, or click the Save icon in the toolbar.

Printing

Print and Print Preview functions are available from the File menu and the toolbar.

Note

To avoid conflicts between changes made in your HDL files, Actel recommends that you use one editor for all of your HDL edits.

Creating New HDL Files

To create an HDL file:

1. Open your project.
2. From the **File** menu, click **New**.
3. Click **VHDL** and type a file name in the Name field. Click **OK**. (Do not enter a file extension; Libero adds one for you.) The HDL Editor workspace opens.
4. After creating your HDL file, save your file to the project by clicking **Save** from the **File** menu. Your HDL file is saved to your project, appearing in the File Manager.

Opening an HDL Source File

To open an HDL source file:

1. From the **File** menu, click **New**.
2. From **Files of Type**, select **HDL File** (*.vhd, *.vhdl).
3. In **Look in**, navigate to the drive/folder where the .hdl file is located.
4. Select your file and click **Open**. Libero opens your file in the HDL Editor

Importing HDL Source Files

Import your HDL file into your project just as you would any source file.

To import an HDL source file:

1. From the **File** menu, click **Import Files**.
2. In **Files of type**, select the file type.
3. In **Look in**, navigate to the drive/folder where the file is located.
4. Select the file to import and click **Open**.

HDL Syntax Checker

After you are done creating your HDL file, use the HDL Syntax Checker to help validate an HDL file after editing the HDL code.

To run the syntax checker:

1. In the **Libero File Manager**, right-click an HDL file and click **Check HDL File**.
2. The syntax checker parses the selected HDL file and looks for typographical mistakes and syntactical errors. Warning and error messages for the HDL file appear in the Libero Log Window.

Commenting Text

You can comment text as you type in the HDL Editor, or you can comment out blocks of text by selecting a group of text and applying the Comment command.

To comment or uncomment out text:

1. Type your text.
2. Select the text.
3. From the **Edit** menu or right-click menu, click **Comment Out** or **Uncomment**.

Using ACTgen Cores

Use ACTgen to:

create high level modules, such as counters, multiplexers, multipliers, etc. that are optimized for Actel FPGAs.

create system level building blocks, such as filters, FIFOs and memories.

These can be instantiated into your schematic, Verilog design, or HDL design.

To use ACTgen with your HDL design:

1. Add the ACTgen core to your Libero project.

   From the Libero **File** menu, click **New**.

   In the New File dialog box, select ACTgen core, type a name, and click **OK**. ACTgen starts.

   Select your core type from the left Macro list box. The appropriate options appear. Select a tab and fill in the fields. Click **Generate**.

   In the Save As dialog box, leave the default selections and click **Save**. The file is added to your Libero project, appearing in the Design Hierarchy.

2. Instantiate the module in your HDL design.

ViewDraw AE

ViewDraw AE is a special version of ViewDraw. Use it for schematic entry with Libero IDE.

Note: The full online help system for ViewDraw AE can be accessed by opening ViewDraw AE and clicking the *Help* menu.

Importing Schematics

You can import any schematic created with ViewDraw AE.

To import a schematic file:

1. From the **File** menu, click **Import Files**.
2. In Files of type, select Schematics.
3. In **Look in**, navigate to the drive/folder where the file is located.
4. Select the file to import and click **Open**. The schematic is imported into your project and appears in the File Manager, under Schematic files.

To open the schematic, click ViewDraw AE in the Design Flow window, or right-click the file in the File Manager and select, **Open Schematic**.

Opening a Schematic Source File

Use ViewDraw AE to edit your schematic files.

To open your schematic file:

1. **Open** your project in Libero IDE.
2. Double-click the schematic file in the File Manager or Design Hierarchy windows. ViewDraw AE opens with the file loaded.
3. From the **File** menu, click **Save+Check** to create the required files for netlist generation. When Save + Check is complete, the Status Bar will say "Check complete, 0 errors and 0 warnings in project <name>." You must select Save +Check. Only selecting Save will not generate the needed WIR file for that block.
4. From the **File** menu, click **Exit**. The schematic is saved to the project, appearing in both the File Manager and Design Hierarchy tabs. Your schematic file is updated in Libero.

Using ACTgen Cores

Use ACTgen to:

create high level modul es, such as counters, multiplexors, multipliers, etc. that are optimized for Actel FPGAs.

create system level building blocks, such as filters, FIFOs and memories.

These can be instantiated into your schematic, Verilog design, or HDL design.

To generate cores for your schematic:

1. Add the ACTgen core to your Libero project.

   From the Libero **File** menu, click **New**.

   In the New File dialog box, select ACTgen core, type a name, and click **OK**. ACTgen starts.

   Select your core type from the left Macro list box. The appropriate options appear. Select a tab and fill in the fields. Click **Generate** to create an HDL representation of the core.

   In the Save As dialog box, leave the default selections and click **Save**. The file is added to your Libero project, appearing in the Design Hierarchy.

2. **Create the Symbol**. In the Design Hierarchy, right-click the ACTgen module and choose **Create Symbol**. The symbol is created, appearing in the File Manager, under Block Symbol files.

3. Use the Symbol.

   Start ViewDraw.

   From the **Add** menu, click **Component**.

   Select the new symbol, then drag and drop it onto your schematic.

Synthesis Overview

Libero IDE works with the following synthesis tools:

Synplify from Synplicity

LeonardoSpectrum from Mentor Graphics

Precision RTL from Mentor Graphics

While LeonardoSpectrum and Precision RTL are not part of the Libero IDE package, they can be integrated to work with Libero IDE. You can also integrate different versions of Synplify. To integrate tools, add them to your project profile.

Post-Synthesis Files

Post-synthesis files include your:

<top>.edn - EDIF netlist file; used for post-synthesis simulation in Libero, also used by Designer to generate back-annotated files and complete layout

<top>.adb - Designer project file, automatically generated by Libero; if you activate PALACE for physical synthesis, the <top>.adb file moves to Post-physical synthesis files.

<top>.vhd - simulation file, automatically generated by Libero

To generate your EDIF netlist, use your Synthesis tool. You can also import your EDIF netlist named <top>.edn.

Icons:

 (Green check) - Files are current; you can use the file for simulation or place-and-route.

 (Yellow exclamation) - Files are out-of-date;something has changed since you created your post-synthesis file (you may have modified the source, family, default synthesis tool, package, etc.). Re-run synthesis to update the file. **You may use the out-of-date file for simulation or place-and-route if you wish.**

Synplify

Libero's integrated synthesis tool, Synplify AE from Synplicity, takes your Verilog or VHDL Hardware Description Language source as input and outputs an optimized EDIF and HDL netlist.

Note: See the Actel Attribute and Directive Summary in the Synplicity online help for a list of attributes related to Actel devices.

Synthesizing Your Design with Synplify

1. In Libero, right-click the HDL file in the **File Manager**, or the top-level schematic for mixed schematic-HDL designs, in the **Design Hierarchy**, and select **Synthesize.** Synplify starts and loads the appropriate design files, with a few pre-set default values.
2. From Synplify's **Project** menu, click **Implementation Options**.
3. Set your specifications and click **OK**.
4. Deactivate synthesis of the defparam statement. The defparam statement is only for simulation tools and is not intended for synthesis. Embed the defparam statement in between **translate_on** and **translate_off** synthesis directives as follows :

```
/* synthesis translate_off */
defparam M0.MEMORYFILE = "meminit.dat"

/*synthesis translate_on */
// rest of the code for synthesis
```

5. Click the **RUN** button. Synplify compiles and synthesizes the design into an EDIF, *.edn, file. Your EDIF netlist is then automatically translated by Libero into an HDL netlist. The resulting *edn and *.vhd files are visible in the File Manager, under Implementation Files

Should any errors appear after you click the Run button, you can edit the file using the Synplify editor. Double-click the file name in the Synplify window showing the loaded design files. Any changes you make are saved to your original design file in Libero.

6. From the **File** menu, click **Exit** to close Synplify. A dialog box asks you if you would like to save any settings that you have made while in Synplify. Click **Yes**.

Note: See the Actel Attribute and Directive Summary in the Synplicity online help for a list of attributes related to Actel devices.

To add a clock constraint in Synplicity, you must add "n:<net_name>" in your SDC file. If you puts the net_name only, it does not work.

Integrating Precision RTL

Libero IDE supports Precision RTL from Mentor Graphics.

To integrate Precision RTL with your Libero IDE project:

1. From the **Options** menu, click **Profile**. The Project Profile dialog box appears.
2. Click **Add** and select **Synthesis**. The Add Tool dialog box appears.
3. Enter a name. This is the name that appears in the Project Profile dialog box.
4. Select Precision RTL.
5. Enter the location of Precision RTL and any additional parameters.
6. Click **OK**.
7. Select Precision RTL in the Project Profile dialog box and click **OK**.
8. Click **Precision RTL** in the Libero Design Flow window to start Precision RTL.

Starting Precision RTL

Before you can start Precision RTL you must add it to your project profile.

To start Precision RTL to run synthesis:

1. In Libero, right-click the HDL file in the File Manager or the top-level schematic for mixed schematic-HDL designs in the Design Hierarchy, and click **Synthesize**. Precision starts.
2. (Optional) Click **Setup Design** to enter clock frequency, input delays and output delays.
3. (Optional) Click **Constraint** if you want to import a constraint file (*.sdf).
4. Click **Compile**, if you want to compile the design first.
5. If compile runs without error, click **Synthesize** to optimize the design for your target technology. To investigate errors in the log window, click the red error icon next to the error. A HDL Text Editor opens and the part of the HDL text which is the source of the error is automatically highlighted for you to modify. Click **Save** to save the changes you have made to the HDL text. Rerun Synthesis to get a successful run.
6. Click **Synthesize**. Precision RTL runs compile and then synthesizes your design.
7. The synthesized netlist (EDIF format) and is visible under Implementation Files in the Libero File Manager tab.
8. From the **File** menu, click **Exit** to close Precision RTL. A dialog box asks you if you would like to save any settings that you have made while in Precision. Click **Yes** to save the Precision project file (*.psp).

Integrating LeonardoSpectrum

Libero IDE supports LeonardoSpectrum from Mentor Graphics.

To integrate LeonardoSpectrum with your Libero IDE project:

1. From the **Options** menu, click **Profile**. The Project Profile dialog box appears.
2. Click **Add** and select **Synthesis**. The Add Synthesis Tool dialog box appears.
3. Enter a name. This is the name that appears in the Project Profile dialog box.
4. Select LeonardoSpectrum.
5. Enter the location of LeonardoSpectrum and any additional parameters.
6. Click **OK**.
7. Select LeonardoSpectrum in the Project Profile dialog box and click **OK**.

8. Click **LeonardoSpectrum** in the Libero Design Flow window to start LeonardoSpectrum.

Synthesizing Your Design with LeonardoSpectrum

Before you can start Precision RTL you must add it to your project profile.

To synthesizing your design with LeonardoSpectrum:

1. In Libero, right-click the HDL file in the File Manager, or the top-level schematic for mixed schematic-HDL designs, in the Design Hierarchy, and select **Synthesize**. LeonardoSpectrum starts.
2. The Input Box is blank. Hit the Enter key on the keyboard and all the design files are loaded into the Input Box.
3. From Tools, click Options.
4. Deselect Automatically save and restore Current Work Directory option.
5. Enter the Clock Frequency if you want to constrain the design.
6. Use the slide bar to select the level of Optimize Effort.
7. Click **Run Flow.** Information about errors can be found by clicking the red error icon next to the error. The HDL Text Editor opens and the part of the HDL text which is the source of the error is automatically highlighted for you to modify.
8. Click **Save** to save the changes you have made to the HDL text. Rerun Synthesis to get a successful run. The synthesized netlist is in EDIF format and is visible under Implementation Files in the Libero File Manager tab.
9. From the **File** menu, click **Exit** to close LeonardoSpectrum. A dialog box asks you if you would like to save any settings that you have made while in LeonardoSpectrum. Click **Yes** to save the LeonardoSpectrum project file (*.lsp).

Integration Issues

Some workarounds are required when using LeonardoSpectrum with Libero IDE.

LeonardoSpectrum starts with empty windows: When you first open LeonardoSpectrum from Libero, the Input window is blank and the output file is not specified. Also, the wrong family appears in the Technology window. To fix this, simply place your cursor in the transcript window and press **Enter**. All windows are updated.

LeonardoSpectrum log files are misplaced: From the **Tools** menu, click **Options**, and **Session Settings**. Deselect Automatically save and restore current working directory. This box must be un-selected in order for your log files to be passed back to Libero properly.

Activating and Deactivating PALACE for Physical Synthesis

PALACE is available only for ProASIC^{PLUS} devices.

When you click Configure Design Flow and select the Use PALACE checkbox, PALACE is activated. You can choose to turn PALACE ON or OFF. You may turn PALACE ON only if you are using a ProASIC^{PLUS} device.

Activating PALACE creates a new implementation in the Libero IDE. Click the PALACE button in the Design Flow to set your PALACE options. When you turn PALACE OFF, it is hidden from the Design Flow window.

PALACE is useful as a performance enhancement tool; Actel recommends that you run PALACE after the design has passed Designer place-and-route. This enables you to compare your layout results with and without PALACE (using different implementations).

Physical Synthesis Files in Libero

Post-physical synthesis files include your:

<top>_palace.edn - PALACE EDIF netlist file; used for post-physical synthesis simulation in Libero, also used by Designer to generate back-annotated files and complete layout

<top>.adb - Designer project file, automatically generated by Libero

<top>_palace.vhd - simulation file, automatically generated by Libero

<top>_palace.gcf - PALACE generated GCF constraint file. Used as input for Designer.

<top>_palace.sdc - PALACE generated SDC constraint file. Used as input for Designer.

To generate your Post-Physical Synthesis files, use your PALACE tool.

Icons:

(Green check) - Files are current; you can use the file for simulation or place-and-route.

(Yellow exclamation) - Files are out-of-date;something has changed since you created your post-physical synthesis file (you may have modified the source, synthesis file, package, etc.). Re-run PALACE to update the file. You may use the out-of-date file for simulation or place-and-route if you wish.

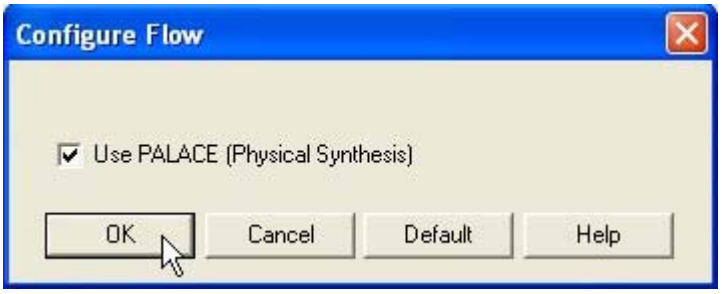
Using the PALACE Tool

PALACE is useful as a performance enhancement tool; Actel recommends that you run PALACE after the design has passed Designer place-and-route. This enables you to compare your layout results with and without PALACE.

Note: The current PALACE version only supports ProASIC^{PLUS}

To use PALACE with Libero:

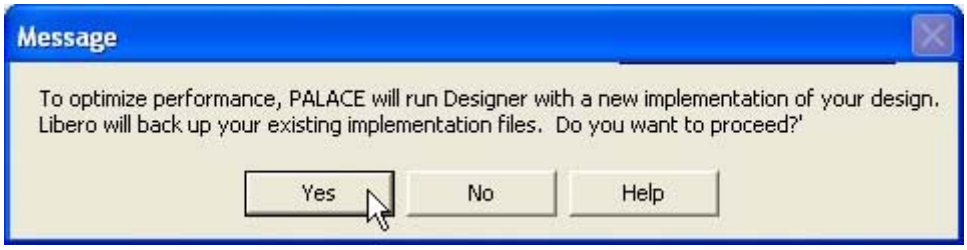
1. Open a ProASIC^{PLUS} project in the Libero IDE. If it is the first time you have opened your project in the latest version of the Libero IDE the software performs a conversion that brings your project up to date, as shown in the figure below.



Libero IDE opens new projects with a default implementation of 1 (impl1). If you have saved more than one implementation in your project, they are available in the Design Implementation drop-down menu.

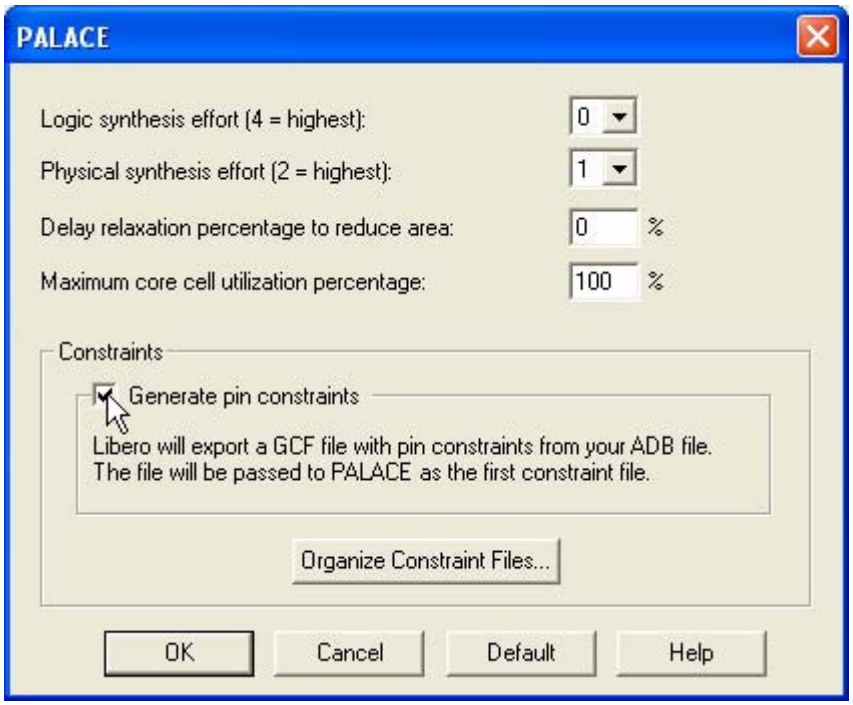
2. Click the Configure Design Flow button in Libero's Design Flow window to add PALACE to your design flow.

The Configure Flow dialog box appears. Select the Use PALACE (Physical Synthesis) to enable the PALACE flow. Click OK to continue. Libero displays a message explaining that it backs up your files and creates a new implementation of your design before it runs PALACE, and asks if you want to proceed, as shown in the figure below.



Create Implementation Message in Libero

3. Click Yes to continue (create a new implementation and complete physical synthesis with PALACE). If you click Yes, Libero creates a new implementation and adds PALACE to your Design Flow window. Click No to cancel and return to the main Libero IDE GUI.
4. Click PALACE in the Design Flow window to display the PALACE options dialog box, as shown in the figure below.



PALACE Options Dialog Box

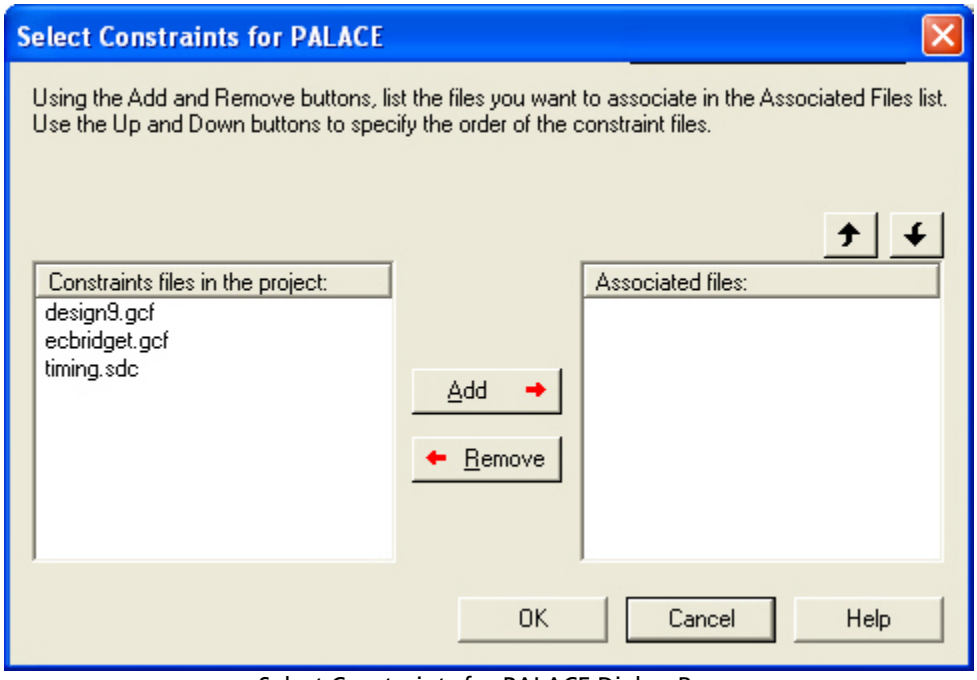
Logic synthesis effort
0 - No optimization, no change in area
1 - Combinatorial / sequential optimization with area focus
2 - Combinatorial optimization mode
3 - Both combinatorial / sequential optimization performed, register balancing, replicating etc
4 - Extensive optimization, exhausting all possible algorithms to achieve timing closure

Physical synthesis effort
1 - Minimum effort for physical placement

2 - Complete physical placement generated for design
Delay relaxation percentage to reduce area
0 - 100: Specifies the percent of delay relaxation on the best possible delay PALACE can achieve. Default is 0, where PALACE aims for maximum optimization
Max core cell utilization percentage:
0 - 100: Specifies the maximum core cell utilization before PALACE promotes the design to next die size. Default is 100 and allows full resource usage.

5. Set your options and click OK to continue.

You must organize your constraint files to guide the PALACE optimization effort. Click **Organize Constraint Files** to open the Select Constraints for PALACE dialog box (as shown in the figure below). The Select Constraints for PALACE dialog box enables you to associate your project's SDC (timing) and GCF (physical) constraints with PALACE. The list at left shows all the constraint files you have imported into your project. The list at right shows all the constraint files you wish to pass to PALACE (Constraint files for PALACE).



Select Constraints for PALACE Dialog Box

When you run PALACE, Libero passes all the constraint files associated with PALACE to the tool.

6. After you set all your options, click OK to run physical synthesis with PALACE.

PALACE completes physical synthesis and the PALACE box in the Design View window turns green.

7. Right-click the PALACE box and select Open Log File to view the PALACE log file.
8. Run place-and-route in Designer. Libero passes the PALACE-generated netlist and constraint files to Designer automatically.
9. If you are not satisfied with your results, return to your previous implementation, or modify your PALACE options and re-run physical synthesis.

WaveFormer Lite

WaveFormer Lite is a special version of WaveFormer Pro that can generate VHDL and Verilog stimulus-based testbenches for Libero IDE. WaveFormer Lite fits perfectly into Libero's design environment, automatically extracting signal information from your HDL design files and producing HDL test bench code that can be used for VHDL or Verilog simulation. WaveFormer Lite now supports VCD files.

WaveFormer Lite generates VHDL and Verilog testbenches from drawn waveforms. WaveFormer Lite can generate the following:

Reactive testbenches

VHDL transport testbench (*.vhd) that uses assignment statements

VHDL wait testbench (*.vhd) that uses wait statements

Verilog (*.v) file with Verilog stimulus statements

Note: WaveFormer Lite comes with its own online help. After starting WaveFormer Lite, click the **Help** menu.

Creating Your Testbench with WaveFormer Lite

WaveFormer Lite generates VHDL and Verilog testbenches from drawn waveforms. Create your test bench after you are done creating your design and wish to perform simulation.

There are five basic steps for creating testbenches using WaveFormer Lite and Libero IDE. These steps are described in detail in the following sections.

To create a testbench using WaveFormer Lite:

1. Click **WaveFormer Lite** in the **Design Flow Window**. WaveFormer Lite starts and your signal information is imported automatically .
2. Using WaveFormer Lite, draw the waveforms to describe the testbench.
3. From the **Export** menu, click **Export Timing Diagram** and choose the save type as *.tim or *.btim. This saves the waveforms.
4. (Optional) Add VHDL Libraries and Use Clauses for VHDL export. These libraries or packages can be included using the VHDL Libraries and Use Clauses dialog. From the **Options** menu, click the **VHDL Libraries and Use Clauses** menu item to open this dialog.
5. From the **Export** menu, click **Export Timing Diagram** and choose the type of file to generate. You can generate a test bench with a top-level module that automatically hooks up the model under test to the testbench, or you can generate just a testbench model. Below is a detailed description of the two methods:
6. To generate a Top-Level Model and a Testbench model choose one of the "top-level" scripts from the save as type drop-down list box. The top-level module instantiates the model under test and hook it up to the test bench. For this script to work the top-level module needs to be defined in the project. For Wave-Former Lite customers, the Actel Software should automatically set this option. Below is a list of top-level scripts:

   VHDL Wait with Top Level Testbench (*.vhd)s

   VHDL Transport with Top Level Testbench (*.vhd)s

   Verilog with Top Level TestBench (*.v)s
7. To generate a plain testbench model (which does not instantiate your model under test) then choose one of the VHDL or Verilog scripts. To simulate with the testbench model, you will need to write a top-level model that instantiates the testbench model and the model under test. This is the method used by Wave-Former Pro customers. Below is a list of VHDL and Verilog testbench generation scripts:

VHDL Wait (*.vhd)s

VHDL Transport (*.vhd)s

Verilog

8. From the **File** menu, click **Exit**.

Note: If you added extra signals to the testbench and do not want to export those signals, then double click the signal's names to open the Signals Properties dialog and uncheck the Export check box.

Synplify always changes the data type to std_logic or std_logic_vector in the post_synthesis netlist. If your top-level entity port is not std_logic or std_logic_vector and you need to run post-synthesis or post-layout simulation, you need to change the data type in WFL.

To create two testbenches:

1. Open the .tim file and select the special data type signal, right-click and choose **Edit Selected Signal**.
2. Choose std_logic or std_logic_vector under VHDL and click **Save.**

If you are running pre-synthesis simulation you do not need to change the data type in WFL.

ModelSim AE

ModelSim Actel Edition (AE) is a custom edition of ModelSim PE that is integrated into Libero's design environment. ModelSim for Actel is an OEM edition of Model Technology Incorporated's (MTI) tools. ModelSim for Actel supports VHDL or Verilog, but it can only simulate one language at a time. It only works with Actel libraries and is supported by Actel.

Other editions of ModelSim are supported by Libero. To use other editions of ModelSim with Libero, simply do not install ModelSim AE from the Libero CD.

Note: ModelSim for Actel comes with its own online help and documentation. After starting ModelSim, click the Help menu.

Setting Your Simulation Options

You can set a variety of simulation options for your project.

To set your simulation options:

1. From the **Options** menu, click **Project Settings**.
2. Click Simulation.
3. Select your options and click **OK**.

   Use automatic do file: Select if you do not want Libero to initialize ModelSim.

   User defined Do file: Enter the do file name or click the browse button.

   Compile VHDL Package files: Select to compile VHDL package files using ModelSim AE.

   Include Do file: Select to execute the wave.do or other specified Do file. Use the wave.do file to customize the ModelSim Waveform window display settings.

   Simulation Run Time: Specify how long the simulation should run in ns. If the value is 0, or if the field is empty, there won't be a run command included in the run.do file.

   Testbench entity name: Specify the name of your testbench entity name. Default is "testbench," the value used by WaveFormer Lite.

Top Level instance name in the testbench: Default is <top_0>", the value used by WaveFormer Lite. Libero replaces <top> by the actual top level macro when ModelSim is run.

Vsim Command Type:Select Min, Typical (Typ), or Max

Resolution: The default is family specific, but you can customize it to fit your needs.

Family	Default Resolution
ACT1, ACT2, ACT3	1 ns
MX	1 ns
DX	1 ns
SX, SX-A	1 ns
eX	1 ns
Axcelerator	1 ps
ProASIC	1 ps
ProASIC PLUS	1 ps
ProASIC3/E	1 ps

Vsim additional options: Text entered in this field is added to the vsim command.

Default: Restores factory settings.

Selecting a Stimulus File for Simulation

Before running simulation, you must associate a testbench. If you attempt to run simulation without an associated testbench, Libero IDE asks you to associate a testbench or open Model*Sim* without a testbench.

To associate a stimulus:

1. Run simulation or right-click the top level module in the Design Hierarchy Menu and click **Select a Stimulus File** from the right-click menu. The Select Stimulus dialog box appears.
2. Associate your testbench(es):

In the Select Stimulus dialog box, all the stimulus files in the current Libero project appear in the left *Stimulus Files in the Project* list box. Files already associated with the block appear in the *Associated Files* list box.

In most cases you will only have one testbench associated with your block. However, if you want simultaneous association of multiple testbench files for one simulation session, as in the case of PCI cores, add multiple files to the Associated Files dialog box.

To add a testbench: Select the testbench you want to associate with the block in the Stimulus Files in the Project list box and click **Add** to add it to the Associated Files list.

To remove a testbench: To remove or change the file(s) in the Associated Files list box, select the file(s) and click **Remove**.

To order testbenches: Use the up and down arrows to define the order you want the testbenches compiled. The top level entity should be in the bottom of the list.

3. When you are satisfied with the Associated File(s) list, click **OK**. A check mark appears next to WaveFormer Lite in the Design Flow window to let you know that a testbench has been associated with the block.

Selecting Additional Modules for Simulation

Libero IDE passes all the source files related to the top-level module to simulation .

If you need additional modules in simulation, right-click the module name in the Design Hierarchy and select **Properties** from the shortcut menu. Click the **Included for Simulation** checkbox to pass the file to simulation.

You must repeat this for each additional module you wish to pass to simulation.

Performing Functional Simulation

1. Create your testbench.
2. Right-click the top level module in the **Design Hierarchy** window.
3. Click **Select a Stimulus File** from the right-click menu.

In the Select Stimulus dialog box, all the stimulus files in the current Libero project appear in the left *Stimulus Files in the Project* list box. Files already associated with the block appear in the *Associated Files* list box.

In most cases you will only have one test bench associated with your block. However, if you want simultaneous association of multiple test bench files for one simulation session, as in the case of PCI cores, add multiple files to the *Associated Files* dialog box.

To add a test bench: Select the test bench you want to associate with the block in the *Stimulus Files in the Project* list box and click **Add** to add it to the Associated Files list.

To remove a testbench: To remove or change the file(s) in the Associated Files list box, select the file(s) and click **Remove**.

To order testbenches: Use the up and down arrows to define the order you want the testbenches compiled.

4. When you are satisfied with the Associated File(s) list, click **OK**. A check mark appears next to WaveFormer Lite in the Design Flow window to let you know that a testbench has been associated with the block.
5. Start **Model*Sim*** AE by doing one of the following:
6. Right-click the top level module in the Design Hierarchy window and select **Run Pre-Synthesis Simulation** or **Run Post-Synthesis Simulation**.
7. Click **Model*Sim*** Simulation in the **Design Flow Window**.

ModelSim starts and compiles the appropriate source files. When the compilation completes, the simulator runs for 1 S and the Wave window opens to display the simulation results.

8. Scroll in the Wave window to verify that the logic of your design functions as intended. Use the zoom buttons to zoom in and out as necessary.
9. From the **File** menu, click **Quit**.

Performing Timing Simulation

The steps for performing functional and timing simulation are nearly identical. Functional simulation is performed before place-and-route and simulates only the functionality of the logic in the design. Timing simulation is performed after the design has gone through place-and-route and uses timing information based on the delays in the placed and routed designs.

Timing simulation includes much more detailed timing information for the targeted device. Timing simulation requires a testbench.

To perform timing simulation:

1. If you have not done so, back-annotate your design and create your testbench.
2. Right-click the top level module in the **Design Hierarchy Menu**.
3. Click **Select a Stimulus File** from the right-click menu.

In the Select Stimulus dialog box, all the stimulus files in the current Libero project appear in the left *Stimulus Files in the Project* list box. Files already associated with the block appear in the *Associated Files* list box.

In most cases you will only have one testbench associated with your block. However, if you want simultaneous association of multiple testbench files for one simulation session, as in the case of PCI cores, add multiple files to the *Associated Files* dialog box.

To add a testbench: Select the testbench you want to associate with the block in the *Stimulus Files in the Project* list box and click **Add** to add it to the Associated Files list.

To remove a testbench: To remove or change the file(s) in the Associated Files list box, select the file(s) and click **Remove**.

To order testbenches: Use the up and down arrows to define the order you want the testbenches compiled.

4. When you are satisfied with the Associated File(s) list, click **OK**. A check mark appears next to WaveFormer Lite in the Design Flow window to let you know that a testbench has been associated with the block.
5. Click **ModelSim Simulation** in the Design Flow window. The ModelSim simulator starts and compiles the source files. When the compilation completes, the simulator runs for 1 S and a Wave window opens to display the simulation results.
6. Scroll in the Wave window to verify the logic works as intended. Use the cursor and zoom buttons to zoom in and out and measure timing delays. If you didn't' create a testbench with WaveFormer Lite, you might get error messages with the vsim command if the instance names of your testbench don't follow the same conventions as WaveFormer Lite. Ignore the error message and type and the correct vsim command.
7. When you are done, from the **File** menu, click **Quit**.

Welcome to Designer

The Designer interface offers both automated and manual flows, with the push-button flow achieving the optimal solution in the shortest cycle.

Actel's Designer software is integrated with Libero IDE. Use the Designer software to implement your design.

To implement your design:

1. **Start Designer**. Right-click the top level module in the **Design Hierarchy** and select **Run Designer**, or click **Designer** in the Design Flow window. Designer starts and loads your files from Libero.
2. **Set up your device.** From the Tools menu, click Device Selection. In the Device Selection Wizard, select the die, package, speed grade, voltage, and operating conditions. Make your selections and click Next to complete the steps
3. **Compile your design**. In Designer, click **Compile** in the design flow window. The log window displays the utilization of the selected device. When compile has completed, the Compile box in the Design Flow window turns green.
4. **Designer's User Tools.** Once you have successfully compiled your design, you can use Designer's User's tool to optimize your design. To start a tool, simply click it in the flow tree. The tools include:

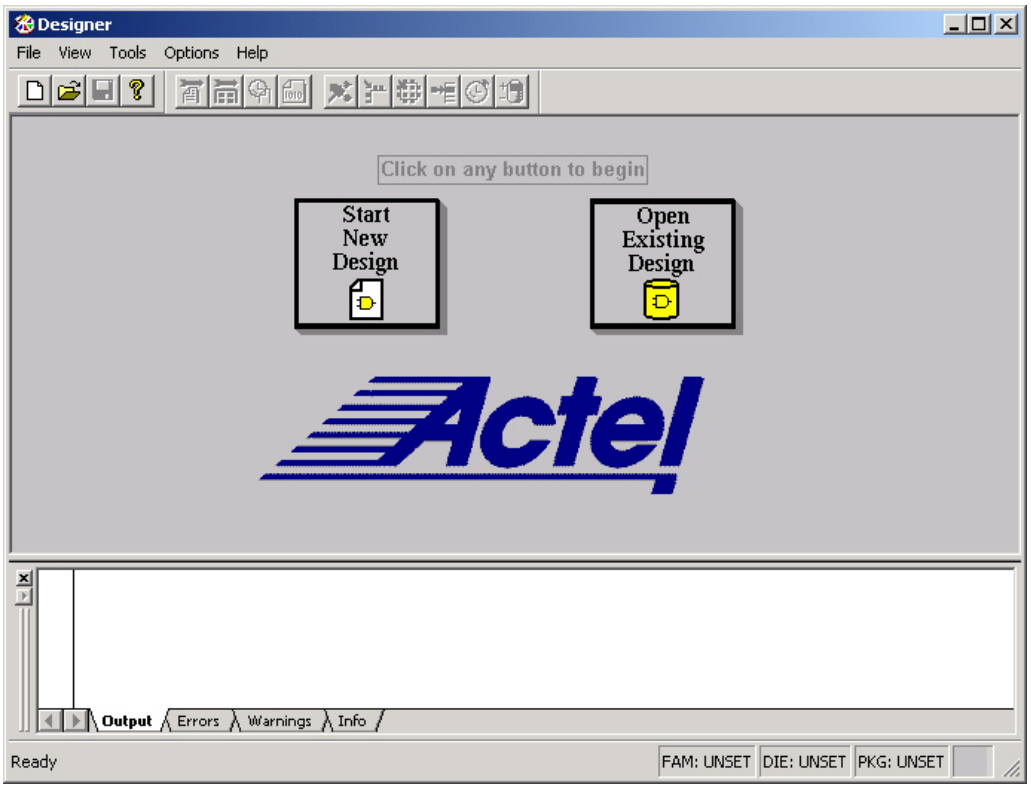
Tool	Function	Supported Families
PinEditor	Package level floorplanner and I/O attribute editor	All
ChipPlanner	Logic viewer, placement and floorplanning tool	Axcelerator,ProASIC, ProASIC^{PLUS}, ProASIC3, ProASIC3E
ChipEditor	Logic viewer and placement tool	All
NetlistViewer	Design schematic viewer	All
SmartPower	Power analysis tool	Axcelerator, ProASIC, ProASIC^{PLUS}, ProASIC3, ProASIC3E
Timer	Static timing analysis and constraints editor	All
I/O Attribute Editor	Edit I/O attributes, layout	Axcelerator, ProASIC, ProASIC^{PLUS}, ProASIC3, ProASIC3E

5. **Layout**. Click Layout in the Design Flow Window to place-and-route your design.
6. **Back-Annotate your design**. Click **Back-Annotate** in the Design Flow Window. Choose SDF as CAE type and appropriate simulation language. Select Netlist in the Export Additional Files area and Click **OK**. If you are exporting files post-layout, Designer exports <top>_ba.vhd and <top>_ba.sdf to your Libero project. The "_ba" is added by Libero to identify these for back-annotation purposes. <top> is the top root name. Pre-layout exported files do not contain "_ba" and are exported simply as *.vhd and *.sdf. The files are visible from the File Manager, under Implementation Files.
7. **Generate a programming file**. Click **Fuse** or **Bitstream** in the design flow tree if you wish to create a programming file for your design. This step can be performed later after you are satisfied with the back-annotated timing simulation.
8. **Save and Exit**. From the **File** menu, click **Exit**. Select Yes to save the design before closing Designer. Designer saves all of the design information in an *.adb file. The <project>.adb file is visible in Libero's File Manger, in the Implementation Files folder. To re-open this file at any time, simply double-click it.

Starting Designer

To start Designer from Libero IDE:

In the Design Flow window, click Designer Place-and-Route.



Starting a New Design

To begin a new design session, you must start a new design or open an existing design.

Starting a new design creates an Actel ADB file. ADB files are proprietary Actel project files.

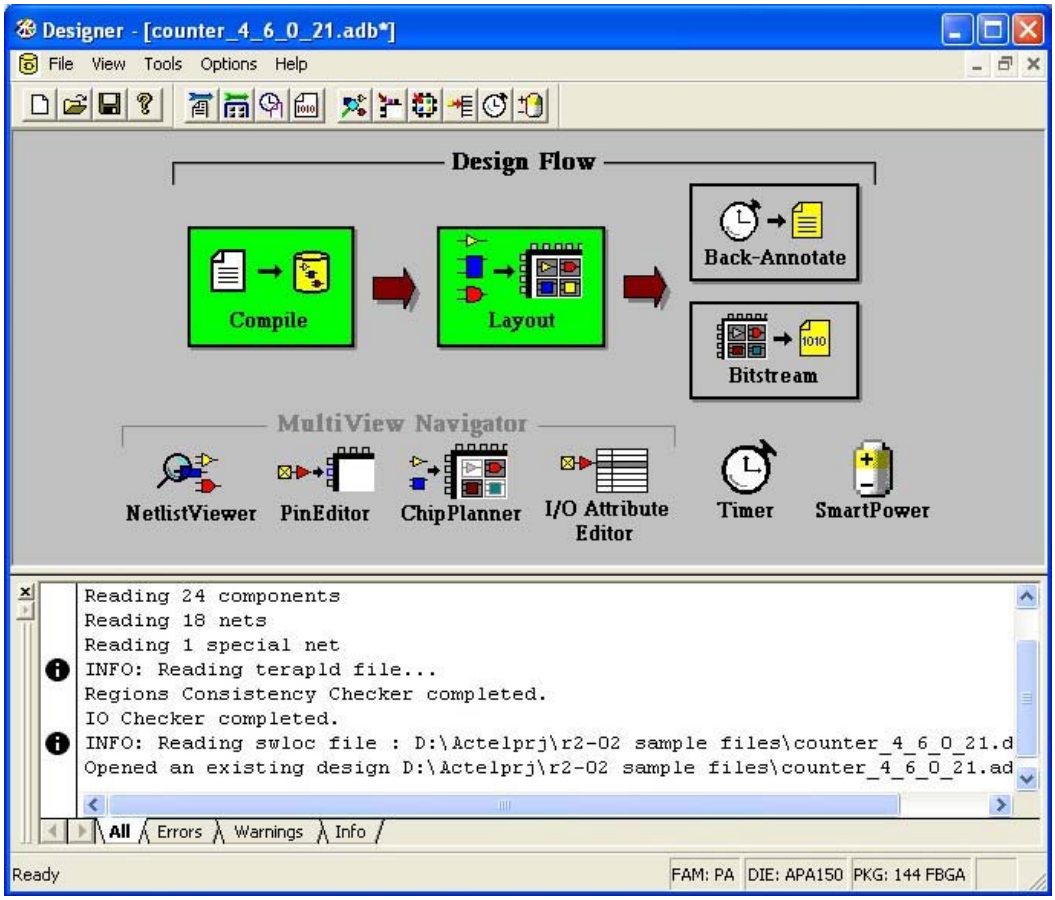
To start a new design:

1. Click **Start New Design** in the Designer main window, or in the **File** menu, click **New**. This displays the Setup Design dialog box.
2. Setup Design:

   Enter a **Design Name**. The design name is used in reports and as the default name when saving or exporting files.

   Select an **Actel Family** from the drop down menu list.

   Specify a **working directory**. Click Browse to locate a directory.
3. Click **OK**. The Designer custom design flow window appears. All tools and commands are activated.



Designer:New Design

Opening an Existing Design

To open an existing design:

1. Click **Open Existing Design** or in the **File** menu, click **Open**. This displays the Open dialog box.
2. Select **File**. Type the full path name of the .adb file in the File Name box, or select the file from the list.
3. Click **Open**. Designer's custom design flow window appears and all tools and commands are activated. When you open an existing design, Designer checks to see if you have modified your netlist since the last time you imported the netlist into this design. If you have, Designer prompts you to re-import your netlist.

Opening Designs Created in Previous Versions

Designer can directly open designs created with previous versions of the Designer software.

If your design was created in version 3.1 or earlier, contact Applications or go to http://www.actel.com/support for information on converting your design.

All existing die, package, pin assignments, and place-and-route information is read and maintained. Designs created in previous versions of software may need library conversions when loaded into the Designer environment. If your design requires this

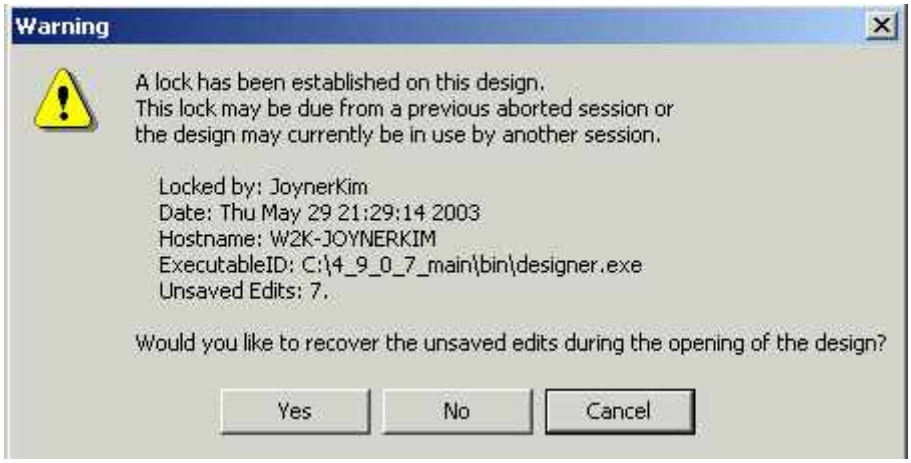
conversion, Designer prompts you to allow the software to update the design to the new library before you attempt to start any of the Designer features.

Opening Locked Files

Designer notifies you if a lock has been established on your file. You might get a warning or an error message when opening a design with a lock.

Warning

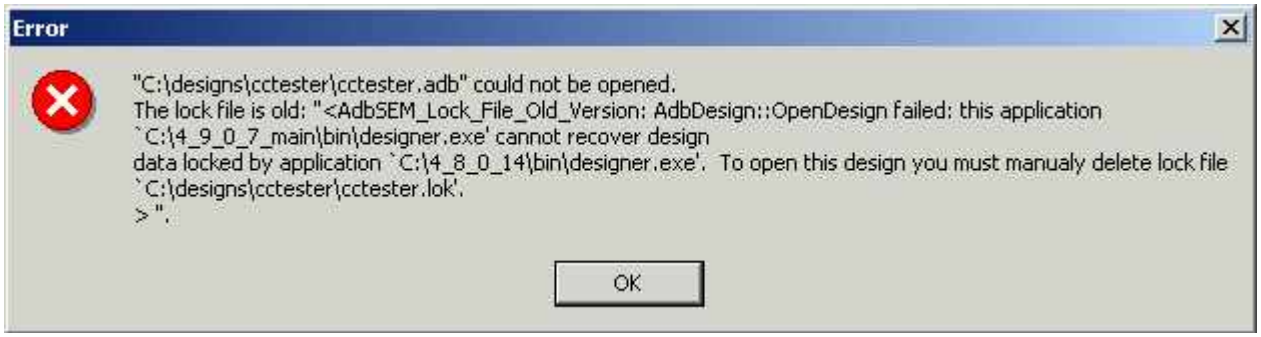
Designer warns you when opening a design that was not closed properly or may be open somewhere else. You can choose to recover the unsaved edits.



Warning: Locked File

Error

When opening a design, an error might notify you that the file can't be opened because the lock file is old. You can't recover any unsaved edits.

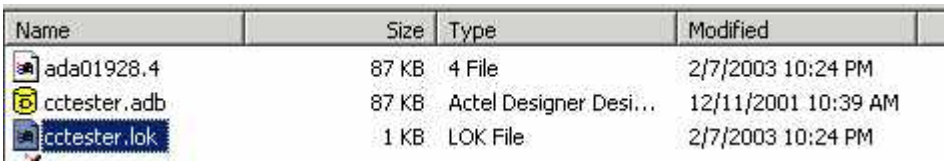


Error: Locked File

To open a design with an old lock file:

1. Go to the design directory.
2. Locate the design .adb file and corresponding .lok file.

3. Delete the .lok file.
4. Return to Designer and open the design.



Starting other applications from Designer (PC only)

You can start any application from Designer that you have added to the Tools menu.

To add an application to the Tools menu:

1. From the **Tools** menu, click **Customize**.
2. Enter the application name in the Menu Text area. This text will appear in the Tools command menu.
3. Enter the command to execute, or click the Browse button to select an executable filename. If the location of the command to execute is not in your path, you must include the absolute path when specifying the command.
4. In the Arguments text box, enter the command-line arguments that will be passed to the command when executing.
5. In the Initial Directory field, type the absolute path of the directory in which the application will initially be executed.
6. Click **Add**.
7. When you are finished adding tools, click **OK**. The application name you added appears in the Tools menu.

To remove an application from the Tools menu:

1. From the **Tools** menu, click **Customize**.
2. Select the application to remove and click **Remove**.
3. When you are finished removing applications, click **OK**.

To order applications in the Tools menu:

1. From the **Tools** menu, click **Customize**.
2. Reorder the tools by selecting one at a time and clicking the **Move Up** or **Move Down** buttons.
3. Click **OK** when you are finished. The tools will appear in the Tools menu in the same order as they do in the Menu Contents list box.

About Your Installation

To display information about your license:

From the Start menu, click Programs, and then click the Actel software folder and select About Your Installation. The software displays your complete license configuration, all Actel-installed software and versions, as well as your HostID and disk volume serial number.

You can also select License Details from the Help menu in Designer to view your license information.

Directory Preferences

When executing a command or function such as Open or Save, Designer uses the directory you specify as the start-up directory.

To specify your directory preferences:

1. From the **File** menu, click **Preferences**.
2. Click the **Directory** tab.
3. Specify your Startup directory.
4. Select your working directory options:

   To design file's directory when opening design: Select to automatically change directories when opening a design.

   To design file's directory when saving design: Select to automatically change directories when saving a design.

   To script file's directory when executing script: Select to automatically change directories when executing a script.

   Add design name to working directory when creating design: Select to enable a design name folder to be automatically created in the working directory when creating a new design.
5. Click **OK**.

Updates

The Updates tab in the Preferences dialog box allows you to set your automatic software update preferences.

To set your automatic software update preferences:

1. From the **File** menu, click **Preferences** and **Updates**.
2. Choose one of the following options and click **OK**.

   Automatically check for updates at startup: Select to be notified of updates when you start Designer.

   Remind me to check for updates at startup: Select to be asked if you want to check for a software update when you start Designer.

   Do not check for updates or remind me at startup: Select if you do not want to check for software updates at startup.

To manually check for software updates, from the **Help** menu, click **Check for Software Updates**.

Note: This feature requires an internet connection.

Proxy

A Proxy improves access to the Actel server.

To enable the proxy:

1. Select I use a proxy.
2. Type the proxy name in the text field.
3. Click **OK**.

File Association (PC Only)

Several programs, including Designer, create files with the .adb extension.

Use the File Association tab in the Preferences dialog box to specify Designer as the default program for files with the .adb extension. Doing so starts Designer whenever a file with the .adb extension is double clicked.

To associated .adb files with the Designer application:

1. From the **File** menu, click **Preferences**.
2. Select Check the default file association (.adb) at startup to Check the box to associate .adb files with the Designer application. Un-check the box if you do not want Designer to start when clicking a file with the .adb extension.
3. Click **OK**.

Setting Your Log Window Preferences

Errors, Warnings, and Informational messages are color coded in the log window. You can change the default colors by using the log Window tab in the Preferences dialog box.

To change colors in the log window:

1. From the **File** menu, choose **Preferences**.
2. Click the **Log Window** tab in the Preferences Dialog Box.
3. Select your new default colors and click **OK**.

The default color settings for the log window are:

Message Type	Colors
Errors	Red
Warnings	Light Blue
Informational	Black
Linked	Dark Blue

The default preference is to Clear log window automatically. This clears the Designer log window each time you close or open a new design in Designer. Uncheck the box if you want Designer to leave the log information after you close a design.

PDF Reader (UNIX Only)

Use the PDF Reader tab to bring up the Designer online manuals. Enter the default reader's name with the full path or click browse.

Web Browser (UNIX Only)

Specify the default web browser you wish to use on the UNIX platform. The web browser displays the online help for Designer.

Device Selection Wizard

After you import your source files, the Device Selection Wizard helps you specify the device, package, and other operating conditions. You must complete these steps before your netlist can be compiled.

The wizard steps include:

Selecting die, package, speed, and voltage

Selecting variations (reserve pins and I/O attributes)

Setting operating conditions

Setting Die, Package, Speed, and Voltage

The first screen in the Device Selection Wizard allows you to set die, package, speed, and voltage.

1. In the **Tools** menu, click **Device Selection** to start the Device Selection Wizard.
2. Select **Die** and **Package**. Select a die from the Die list. Available packages are listed for each die.
3. Specify **Speed.**
4. Select **Die Voltage**. Select from the available settings in Die Voltage drop-down menu. Two numbers separated by a "/" are shown if mixed voltages are supported. If two voltages are shown, the first number is the I/O voltage and the second number is the core (array) voltage.
5. Click **Next** to set reserve pins and I/O Attributes.

Device Variations

The second screen in the Device Selection Wizard enables you to set reserve JTAG and probe pins and the default I/O standard.

To select reserve pins and default I/O standard:

1. Select your reserve pins:

   Check the Reserve JTAG box to reserve the JTAG pins "TDI," "TMS," "TCK," and "TDO" during layout.

   Check the Reserve JTAG Reset box to reserve the JTAG reset Pin "TRST" during layout.

   Check the Reserve Probe box to reserve the Probe pins "PRA," "PRB," "SDI," and "DCLK" during layout.

Reserve Pins are not selectable for the Axcelerator, ProASIC, and ProASIC Plus families.

2. Select an I/O attribute. The I/O Attributes section notifies you if your device supports the programming of I/O attributes on a per-pin basis. For the Axcelerator family, the I/O Attribute section allows you to set the default I/O standard for the I/O banks.
3. Click **Next** to set operating conditions.

Setting Operating Conditions

Operating Conditions, step 3 of the Device Selection Wizard, enables you to define the voltage and temperature ranges a device encounters in a working system. The operating condition range entered here is used by Timer, the timing report, and the back-annotation function. These tools enable you to analyze worst, typical, and best case timing.

Junction Temperature

Select a junction temperature. Supported ranges include:

Commercial (COM)

Industrial (IND)

Military (MIL)

Automotive

Custom

Consult the Actel Data Sheet, available at http://www.actel.com/techdocs/ds/index.html to find out which temperature range you should use.

If you select Custom, edit the Best, Typical, and Worst fields. Modify the range to the desired value (real) such that Best < Typical < Worst.

You can calculate junction temperature from values in the Actel Data Sheet, available at http://www.actel.com/techdocs/ds/index.html.

The temperature range represents the junction temperature of the device. For commercial and industrial devices, the junction temperature is a function of ambient temperature, air flow, and power consumption.

For military devices, the junction temperature is a function of the case temperature, air flow, and power consumption. Because Actel devices are CMOS, power consumption must be calculated for each design. For most low power applications (e.g. 250mW), the default conditions should be adequate.

Performance decreases approximately 2.5% for every 10 degrees C that the temperature values increase. Refer to the SmartPower online help for more information about power consumption.

Voltage

Select a voltage:

Commercial (COM)

Industrial (IND)

Military (MIL)

Automotive

Custom

If you select Custom, you may choose from Best > Typical > Worst in the drop-down menu.

Radiation Derating

Conservative post radiation performance estimates are available for some radiation tolerant devices based upon the number of KRads the device is expected to be subjected to. Radiation effects vary by device lot and may not be completely representative of the lot you are using. Post radiation timing numbers are only meant to be a guide and are not a guarantee of performance. Customers must consult the specific radiation performance report for the specific lot used. Post radiation exposure estimates currently only affect timing numbers. The SmartPower power analysis tool is not affected by changing the radiation exposure value.

RTSX-S and RTAX-S ONLY - Radiation Derating

This option is only available for RTSX-S and RTAX-S devices. The valid range is integer values from 0 to 300, and the units are in KRads. Modifying this selection impacts the timing derating in Timer and back-annotating SDF files, so when you modify this value, you must extract a new SDF file from Designer and re-evaluate the timing of your design. It does not affect the device configuration.

Changing Design Name and family

Design name and family are set when you create a new design. However, you can change this information for existing designs. If you change the family, you must re-import the netlist. Use the following procedure to change the name of a design and the targeted Actel family for the design.

To change the design name or family:

1. In the **Tools** menu, click **Setup Design**. This displays the Setup Design dialog box.
2. Specify the design name and family.

3. Click **OK**. Refer to the Actel datasheet for your device for family specifications.

You may wish to migrate your SX device to an SX-A device. The SX to SX-A compatible family change option is available in the Device Selection wizard.

To migrate your SX design to SX-A:

1. Open your SX design.
2. From the **Tools** menu, select **Device Selection**.
3. Select "SXA" from the **Change to** drop down menu, and proceed in the Device Selection Wizard to complete the migration. You must re-compile and layout your design to run the Designer User Tools.

Changing Device Information

Device and package information, device variations, and operating conditions are set when you import a netlist and compile a new design. However, you can change this information for existing designs.

To change device information for existing designs:

1. In the **Tools** menu, click **Device Selection**. The Device Selection Wizard appears.
2. Select Die, Package, and Speed Grade and click **Next**. (You must select die and package to continue.)
3. Select Device Variations and click **Next**.
4. Select Operating Conditions and click **Finish**.

Refer to the Actel FPGA Data Book or call your local Actel Sales Representative for information about device, package, speed grade, variations, and operating conditions.

Compatible Die Change

When you change the device, some design information can be preserved depending on the type of change.

Changing Die Revisions

If you change the die from one technology to another, all information except timing is preserved. An example is changing an A1020A (1.2um) to an A1020B (1.0um) while keeping the package the same.

Device Change Only

Constraint and pin information is preserved, when possible. An example is changing an A1240A in a PL84 package to an A1280A in a PL84 package.

Repackager Function (Non-Axcelerator families only)

When the package is changed (for the same device), the Repackager automatically attempts to preserve the existing pin and Layout information by mapping external pin names based on the physical bonding diagrams. This always works when changing from a smaller package to a larger package (or one of the same size). When changing to a smaller package, the Repackager determines if any of the currently assigned I/Os are mapped differently on the smaller package. If any of the I/Os are mapped differently, then the layout is invalidated and the unassigned pins identified.

Importing Source Files

Source files include your netlist and constraint files.

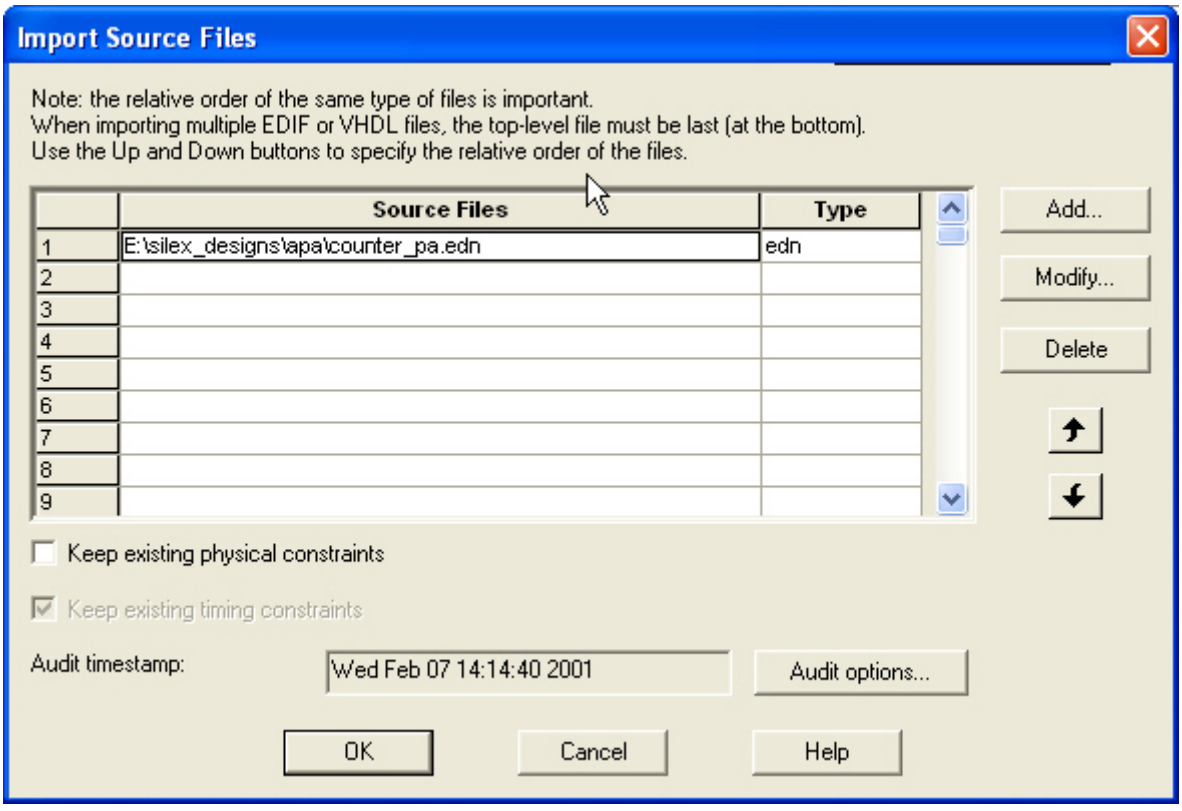
Source files are files created by outside tools that will be tracked (audited) to better coordinate the design changes. If you wish, you may import some files as auxiliary files. Auxiliary files are not audited, but you do not have to re-compile your design after you import them.

Source Files	File Type Extension
EDIF	*.ed*
Verilog	*.v
VHDL	*.vhd
Actel ADL Netlist	*.adl
Criticality	*.crt
ProASIC Constraint File (ProASIC^{PLUS}, ProASIC)	*.gcf
Physical Design Constraint File (ProASIC3/E, Axcelerator)	*.pdc
SDC	*.sdc

The choice of source files is family dependent. Only supported source files are displayed in the Import Source dialog box. If you are working on a new design or if you have changed your netlist, then you must re-import your netlist into Designer.

To import a source file:

1. From the **File** menu, choose **Import Source Files**. This displays the Import Source Files dialog box, as shown in the figure below.



Import Source Files Dialog Box

2. Click **Add**. The Add Source Files dialog appears.
3. Select the file you want to import and click **Import**. The File is added to the Import Source Files dialog box.
4. Add more source files to the list. All files added to the Import Source Files dialog box are imported at the same time. To modify a file, select the file and click **Modify**. To delete a file, select the file and click **Delete**.
5. Specifying a priority is useful if you are importing multiple netlist files, GCF files, or PDC, or SDC files. When importing multiple EDIF or structural HDL files, the top-level file must appear last in the list (at the bottom). Drag your files to specify the import order.
6. (Axcelerator, ProASIC, ProASIC PLUS, ProASIC3/E designs only) Select Keep existing physical constraints to preserve all existing physical constraints that you have made using ChipPlanner, PinEditor, or the I/O Attribute Editor.
7. (SX-A, eX, Axcelerator, ProASIC PLUS, ProASIC3/E) Select Keep existing timing constraints to preserve all the existing timing constraints already in your design, whether coming from the Timer tool or previously imported file. If you import an SDC file and select this option, Designer merges the existing constraints and the constraints contained in the SDC file. In case of a conflict, the new constraint has priority over the existing constraint.
8. To set the audit options for these source files, click **Audit options** and follow the directions in the Audit Options dialog box.
9. When you are done adding all your source files, click **OK**. Your source files are imported. Any errors appear in the Designer log window.

Note: Designer may not import file names or paths with spaces. Rename the file or path to remove the spaces, and re-import.

Importing SDC and GCF Files Simultaneously

If you specify SDC and GCF files in the Import source file dialog, GCF files are imported before SDC files regardless of the order you specify. This means that any timing constraints in GCF are converted before the SDC constraints are applied. For a cleaner flow, Actel recommends that you do not use both SDC and GCF timing constraints. SDC is the recommended format.

Auditing Files

Designer audits your source files to ensure that your imported source files are current. All imported source files are date and time stamped. Designer notifies you if the file is changed. When notified, select the appropriate action and click **OK**.

To change your audit settings:

1. From the **File** menu, click **Audit Settings**. The Audit Settings dialog box appears. Audit Timestamp reflects the last time and day that the import source or audit update was successfully done.
2. Select the audit check box next to the file to enable auditing.
3. Click **Change Location** to open the "Modify File Location" dialog. The Modify File Location dialog box enables you to specify the correct path so that the design can find the source file(s)..
4. Click **Reset to Current Date Time** to associate the file with the current day and time.

Importing Auxiliary Files

Auxiliary files are not audited and are treated more as one-time data-entry or data-change events, similar to entering data using one of the interactive editors (e.g. PinEditor or Timer).

Some timing constraints (such as multi_cycle) are not supported in the Timer GUI and must be implemented by importing the SDC file. If you import the SDC file as an auxiliary, you do not have to re-compile your design. However, auditing is disabled when you import auxiliary files, and Designer cannot detect the changes to your SDC file(s) if you import them as auxiliary files.

Auxiliary Files	File Type Extension	Family
Criticality	*.crt	ACT1, ACT2, ACT3, MX, XL, DX
PIN	*.pin	ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX
SDC	*.sdc	ProASIC3/E, SX-A, eX, Axcelerator, ProASIC PLUS
Physical Design Constraint	*.pdc	ProASIC3/E and Axcelerator
Value Change Dump	*.vcd	ProASIC3/E, Axcelerator, ProASIC, ProASIC PLUS
Switching Activity Intermediate File/Format	*.saif	ProASIC3/E, Axcelerator, ProASIC, ProASIC PLUS
Design Constraint File	*.dcf	Axcelerator, ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX

To import an auxiliary file:

1. From the **File** menu, select **Import Auxiliary Files**. The Import Auxiliary Files dialog appears.
2. Click the **Add** button. The Add Auxiliary Files dialog box appears.
3. Select your file and click **Import**. The file is added to the Import Auxiliary Files dialog box. Continue to add more auxiliary files to the list. Some formats (like DCF and SDC) are not allowed to be imported in multiple auxiliary files.

   Modifying:If you need to modify a selection, select the file row and click **Modify**

   Deleting:If you need to delete a file, select the file row and click **Delete**.

   Ordering:Ordering your auxiliary files. Select and drag your files to specify the import order. Specifying a priority is useful if you are importing multiple PDC files.
4. After you are done adding all your Auxiliary files, click **OK**. Your auxiliary files are imported. Any errors appear in Designer's Log Window.

Note:

.vcd and .saif are used by SmartPower for power analysis.

.crt for backwards compatibility with existing designs only.

File names or paths with spaces may not import into Designer. Rename the file or path, removing the spaces, and re-import.

Keep Existing Timing Constraints in SDC Files

The Keep Existing Constraints check box is designed to support an additional "merge or replace" functionality when you import SDC files.

Select **Keep existing timing constraints** to preserve all existing timing constraints that you have made using the Timer GUI or previously imported file. If you import a SDC file and you have this box selected, Designer merges the existing constraints and the constraints existing in the SDC file. In case of a conflict, the new constraint has priority over the existing constraint.

The Keep Existing Constraints option is **On** by default. With this option **On**, your timing constraints from the imported SDC files are merged with the existing constraints. When this option is **Off**, all the existing timing constraints are replaced by the constraints in the newly imported SDC files.

Keep Existing Physical Constraints

The Keep Existing Physical Constraints option in the Import Source Files dialog box enables you to merge or replace existing constraints when you import new or modified GCF or PDC files.

Select **Keep existing physical constraints** to preserve all existing physical constraints that you have entered either using one of the MVN tools (ChipPlanner, PinEditor, or the I/O Attribute Editor) or a previous GCF or PDC file. The software will resolve any conflicts between new and existing physical constraints and display the appropriate message.

The Keep Existing Constraints option is **Off** by default. When this option is **Off**, all the physical constraints in the newly imported GCF or PDC files are used. All pre-existing constraints are lost. When this option is **On**, the physical constraints from the newly imported GCF or PDC files are merged with the existing constraints.

Compiling Your Design

After you import your netlist files and select your device, you must compile your design. Compile contains a variety of functions that perform legality checking and basic netlist optimization. Compile checks for netlist errors (bad connections and fan-out problems), removes unused logic (gobbling), and combines functions to reduce logic count and improve performance. Compile also verifies that the design fits into the selected device.

There are three ways to select the compile command:

In the Tools menu, select **Compile**.

Click the **Compile** button in the Design Flow.

Click the C**ompile** icon in the toolbar.

If you have not already done so, Designer's Device Selection Wizard prompts you to set the device and package.

During compile, the message window in the Main window displays information about your design, including warnings and errors. Designer issues warnings when your design violates recommended Actel design rules. Actel recommends that you address all warnings, if possible, by modifying your design before you continue.

If the design fails to compile due to errors in your input files (netlist, constraints, etc.), you must modify the design to remove the errors. You must then re-import and re-compile the files.

After you compile the design, you can run Layout to place-and-route the design or use the User Tools (PinEditor, ChipEditor, ChipPlanner, Timer, SmartPower, or NetlistViewer) to perform additional optimization prior to place-and-route.

Compile Options

Setting Compile Options

To set compile options

1. From the **Options** menu, click **Compile**. The Compile Options dialog box opens. The Options available are family specific.
2. Select your options, and click **OK**.

Note: ProASIC3/E Compile options appear by default each time you compile the design. If you have disabled this feature, you can click the Options menu and choose Compile to review/change/reset your Compile options.

Compile options vary according to family.

MX, SX, SX-A, eX

Axcelerator

ProASIC, ProASICPLUS

ProASIC3/E

MX, SX, SX-A, eX Compile Options

Netlist Pin Properties Overwrite Existing Properties

During the Compile process, Designer checks the netlist properties. If the netlist file specifies a pin assignment for a pin that was also assigned in PinEditor session, there is a conflict. How this conflict is resolved is determined by your selection in this box.

If this option is off, or unchecked, then Designer uses the assignment made in PinEditor, and the assignment in the netlist file for the conflicting pin is ignored. If this option is on, or checked, then Designer uses the assignment in the netlist file for that pin, and the PinEditor assignment is ignored. If you edit pin assignments in PinEditor, this option is automatically set to "off."

Fanout messages

Use the control slider in the Messages area to control the warning level. Use the control slider to specify the fanout limit that the Compile step checks against. Setting the control slider to '0' informs the system to use the system defaults. Any non-zero value replaces the system default value for the fanout limit with the user-specified value. Typically, this value range is 1 to 24.

This does not adjust the fanout of the design and it has no effect on the netlist. This only adjusts the warning level, by controlling what level of fanout checking you want to be warned about during Compile. Changing this fanout limit option does not invalidate the Compile design state.

Axcelerator Compile Options

Combine registers into I/Os where possible

The Axcelerator, ProASIC3, and ProASIC3E families includes an optional register on the input path, an optional register on the output path, and an optional register on the 3-state control pin. Select the option Combine Registers into I/Os where possible to take advantage of these registers.

Abort on PDC error

Setting Abort on PDC Error aborts the PDC import when an error is encountered. When this box is checked, the PDC file is either imported fully or the design is left untouched.

ProASIC and ProASICPLUS Compile Options

Include RAM and I/O in Spine and Net Regions

This option affects the behavior of the following:

The use_global constraint

The set_net_region constraint

The creation of spines in the MultiView Navigator

Selecting "Include RAM and I/O in Spine and Net Regions" enables you to assign memory and I/O to spine (LocalClock) and net regions.

When this option is checked, Designer will apply the use_global and set_net_region constraints to core cells, memory, and I/O. When unchecked, Designer will apply the use_global and set_net_region constraints to core cells only. For new designs, this box is automatically checked. For designs created with v5.1 or earlier, this option is unchecked by default. If you change this default setting, you must recompile your design.

This option also determines whether memory and I/O are included in a LocalClock region that you create with the ChipPlanner tool. If checked, memory and I/O are included. If not checked, they are excluded.

ProASIC3/E Compile Options

This interface lets you do the following:

Verify Physical Design Constraints

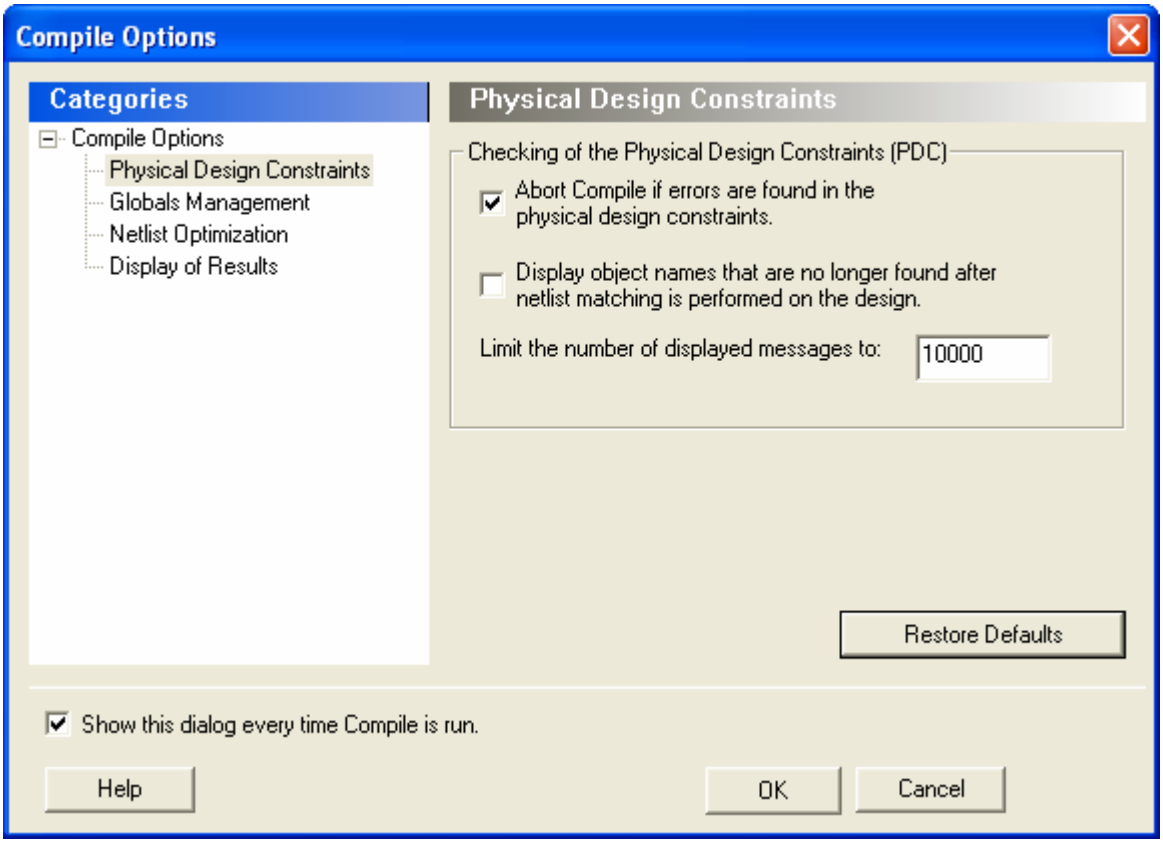
Perform Globals Management

Netlist Optimization

Generate Compile report

Physical Design Constraints

This interface enables you to verify the Physical Design Constraints (PDC) file.



Checking the Physical Design Constraint (PDC)

Abort Compile if errors are found in the physical design constraints: Changes the "Abort on PDC error" behavior. Select this option to stop the flow if any error is reported in reading your PDC file. If you deselect this option, the tool skips errors in reading your PDC file and just reports them as warnings. The default is ON.

Note: The flow always stops even if this option is deselected in the following two cases:

If there is a Tcl error (For example, the command does not exist or the syntax of the command is incorrect)

The assign_local_clock command for assigning nets to LocalClocks fails. This may happen if any floor planning DRC check fails, such as, region resource check, fix macro check (one of the load on the net is outside the local clock region). If such an error occurs, then the Compile command fails. Correct your PDC file to proceed.

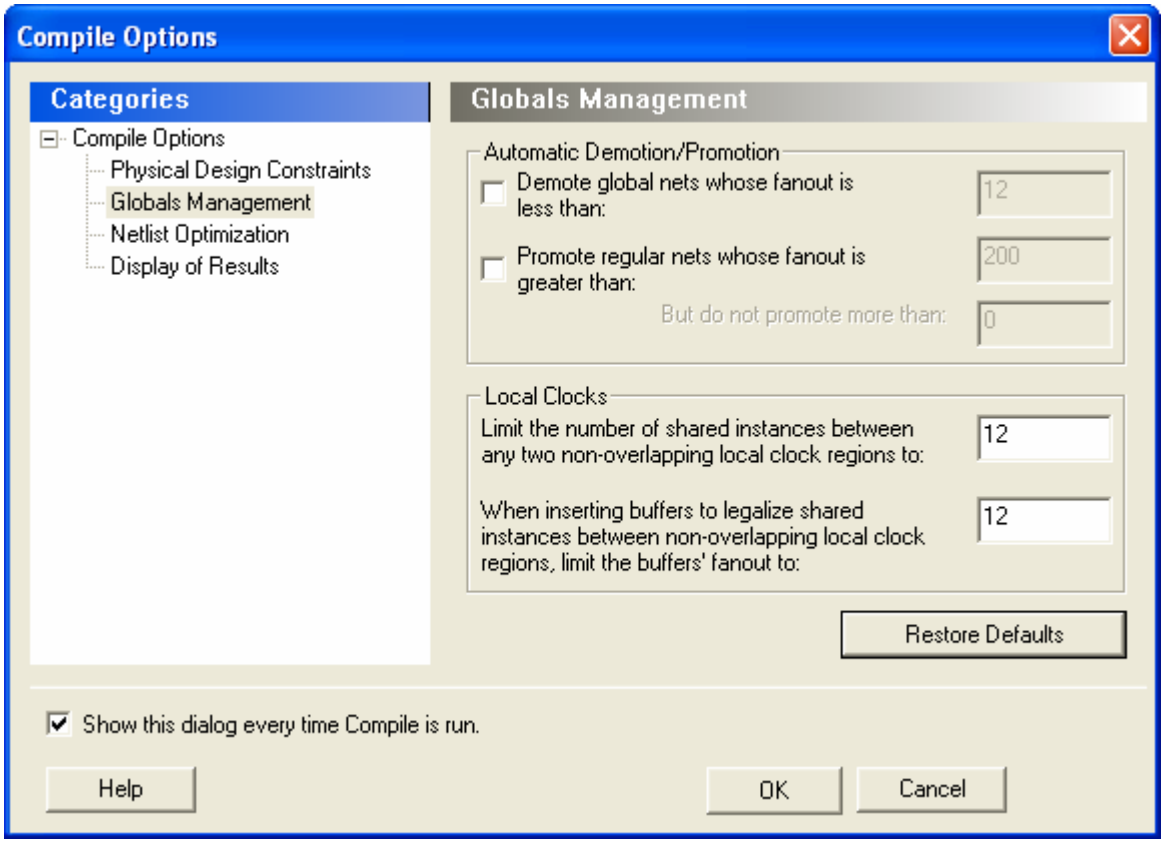
Note: Every time you invoke this dialog box, this option is reset to its default value ON. This is to ensure that you have a correct PDC file.

Display object names that are no longer found after netlist matching is performed on the design: Displays netlist objects in the PDC that are not found in the imported netlist during the Compile ECO mode. Select this option to report netlist objects not found in the current netlist when reading the internal ECO PDC constraints. The default is OFF.

Limit the number of displayed messages to: Defines the maximum number of errors/warnings to be displayed in the case of reading ECO constraints. The default is 10000 messages.

Globals Management

The interface provides a global control to the Compile component of the design flow.



Automatic Demotion/Promotion

Demote global nets whose fanout is less than: Enables the global clock demotion of global nets to regular nets. By default, this option is OFF. The maximum fanout of a demoted net is 12.

Note: A global net is not automatically demoted (assuming the option is selected) if the resulting fanout of the demoted net is greater than the max fanout value. Actel recommends that the automatic global demotion only act on small fanout nets. Actel recommends that you drive high fanout nets with a clock network in the design to improve timing and routability.

Promote regular nets whose fanout is greater than: Enables global clock promotion of nets to global clock network. By default, this option is OFF. The minimum fanout of a promoted net is 200.

But do not promote more than: Defines the maximum number of nets to be automatically promoted to global. The default value is 0. This is not the total number as nets need to satisfy the minimum fanout constraint to be promoted. The promote_globals_max_limit value does not include globals that may have come from either the netlist or PDC file (quadrant clock assignment or global promotion)

Note: Demotion of globals through PDC or Compile is done before automatic global promotion is done.

You may exceed the number of globals present in the device if you have nets already assigned to globals or quadrants from the netlist or by using a PDC file. The automatic global promotion adds globals on what already exists in the design.

Local clocks

Limit the number of shared instances between any two non-overlapping local clock regions to: Defines the maximum number of shared instances allowed to perform the legalization. It is also for quadrant clocks.

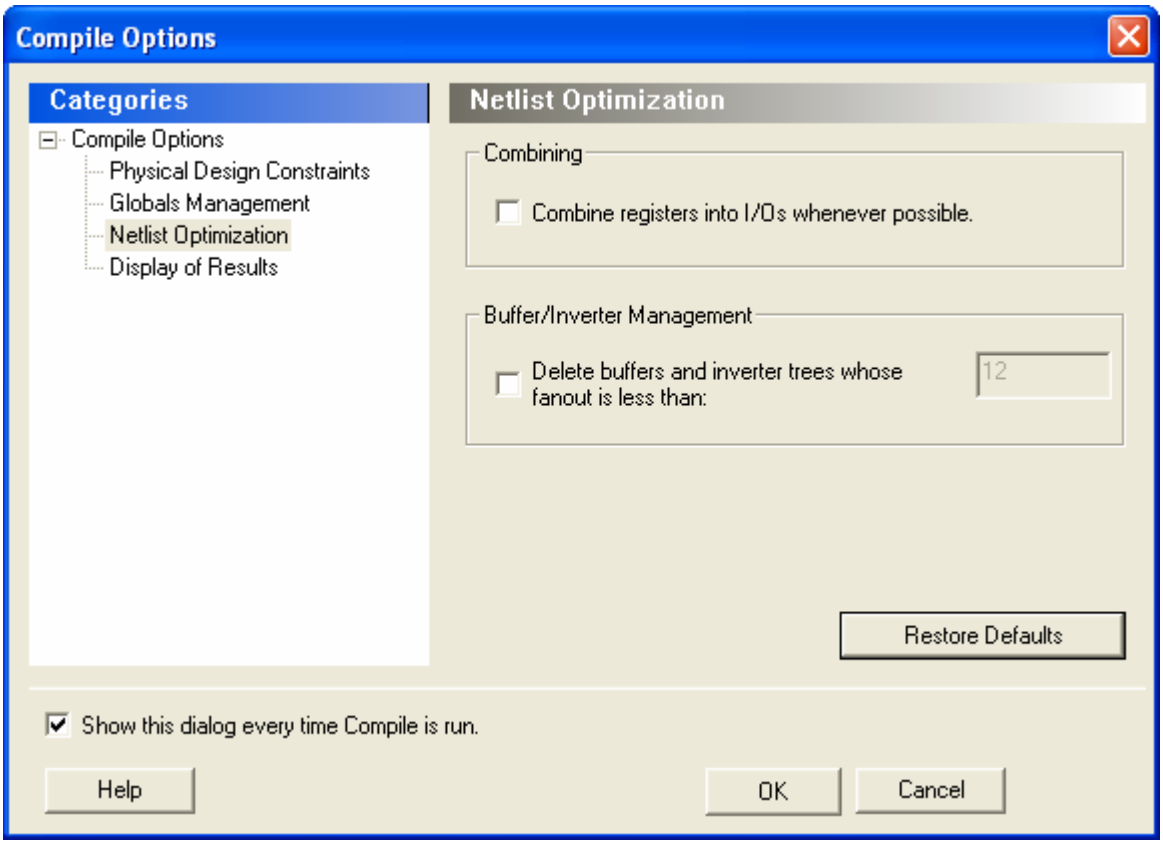
The maximum number of instances allowed to be shared by 2 local clock nets assigned to disjoint regions to perform the legalization (default is 12, range is 0-1000). If the number of shared instances is set to 0, no legalization is performed.

When inserting buffers to legalize shared instances between non-overlapping local clock regions, limit the buffers' fanout to: Defines the maximum fanout value used during buffer insertion for clock legalization. Set the value to 0 to disable this option and prevent legalization (default value is 12, range is 0-1000). If the value is set to 0, no buffer insertion is performed. If the value is set to 1, there will be one buffer inserted per pin.

Note: If you assign quadrant clock to nets using MultiView Navigator, no legalization is performed.

Netlist Optimization

This interface allows you to perform netlist optimization.



Combining

Combine registers into I/O wherever possible: Combines registers at the I/O into I/O-Registers. Select this option for optimization to take effect. By default, this option is OFF.

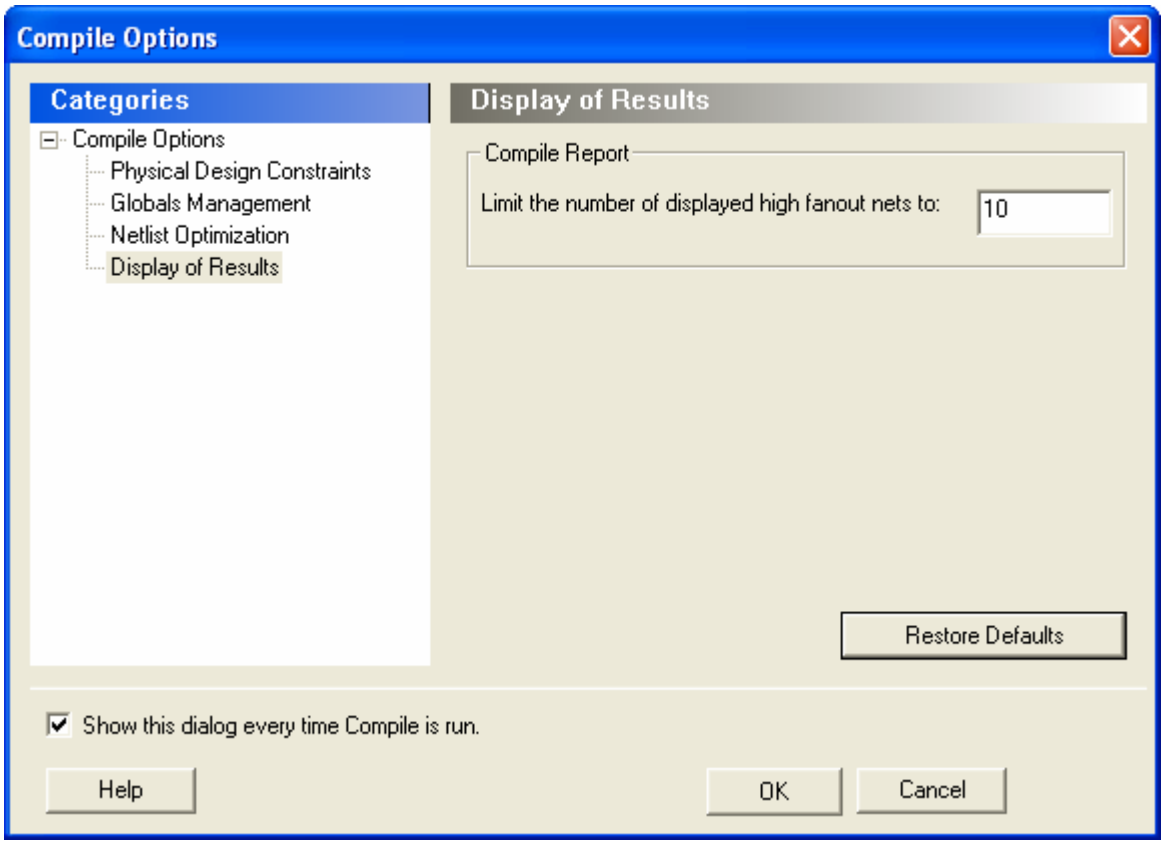
Buffer/Inverter Management

Delete buffers and inverter trees whose fanout is less than: Enables buffer tree deletion on the global signals from the netlist. The buffer and inverter are deleted. By default, this option is OFF. The maximum fanout of a net after buffer tree deletion is 12.

Note: A net does not automatically remove its buffer tree (assuming the option is on) if the resulting fanout of the net (if the buffer tree was removed) is greater than the max fanout value. It is recommended that the automatic buffer tree deletion should only act on small fanout nets. From a routability and timing point of view, it is not recommended to have high fanout nets not driven by a clock network in the design.

Display of Results

This interface lets you generate a Compile report.



Compile Report

Limit the number of displayed high fanout nets to: Enables flip-flop net sections in the compile report and defines the number of nets to be displayed in the high fanout. The default value is10

About Design Constraints

Design constraints are specifications for placing, implementing, naming, and timing considerations of physical and logical assignments. They are usually either restrictions or properties in your design. There are several types of constraints: routing, timing, area, mapping, and placement constraints.

Timing constraints

ocation and region assignment constraintsLocation and region assignment constraints (placing and routing)

I/O assignment constraints (pin location and I/O attributes)

Attributes

You use constraints to ensure that a design meets timing performance and required pin assignments.

Designer supports both physical and timing constraints. You can set constraints by either using Actel's interactive tools or by importing constraint files directly into Designer.

About Location and Region Assignments

You can set constraints for locations and regions using the ChipPlanner or ChipEditor tool as well as via constraint files.

The tool you use depends on which product family you are designing for:

For ProASIC3E, ProASIC3, ProASIC PLUS, Axcelerator, and ProASIC devices, use ChipPlanner.

For other design families, use ChipEditor.

If you choose to use constraint files to set your location and region assignments, the exact constraint file you use depends on your device family. See Types of physical constraints for more information.

When you open your design (.adb) file, Designer automatically presents you with the appropriate tools in the Design Flow window.

About Physical Constraints and Attributes

Physical Constraints

Physical constraints are the placement and routing constraints that apply to a specific architecture and device. You can either import a constraint file or enter them in your design using one of several tools.

If you are designing for the ProASIC3E, ProASIC3, ProASIC PLUS, Axcelerator, and ProASIC families, you will use the tools available from within the MultiView Navigator. These tools are:

ChipPlanner - Sets location and region assignments

PinEditor in MVN - Sets the pin location constraints

I/O Attribute Editor - Sets I/O attributes

For all other families, you will use the standalone versions of the following tools:

ChipEditor - Sets location and region assignments

PinEditor - Sets I/O attributes and pin location constraints

Physical constraints may also be specified in three types of files. The supported file types depend on the family. See Types of physical constraints for more information.

Attributes

Attributes are the characteristics of logic macros or nets in your design. They indicate placement, implementation, naming, directionality, and other characteristics. This information is used by the design implementation software during placement and routing of a design.

Input and output attributes are described in the documentation for the I/O Attribute Editor. Attributes applicable to a specific tool are described in the documentation for that tool.

Types of Physical Constraints

Designer supports three types of physical constraints:

I/O assignments

Location and region assignments

Clock assignments

I/O Assignments

Use PinEditor to manually place and configure your I/Os. Or, assign I/O locations automatically by importing one of the following constraint files into Designer:

GCF (ProASIC PLUS and ProASIC families)

PDC (ProASIC3E, ProASIC3, and Axcelerator families)

PIN (all families except ProASIC3E, ProASIC3, ProASIC PLUS, Axcelerator, and ProASIC)

Location and Region Assignments

Use ChipPlanner to view the placement and routing for ProASIC3E, ProASIC3, Axcelerator, ProASIC, and ProASIC PLUS designs. Use ChipEditor to view and manually change location assignments for other design families.

You can also assign location constraints and enter region constraints by importing one of the following constraint files into Designer:

GCF (ProASIC and ProASIC PLUS families)

PDC (ProASIC3E, ProASIC3, and Axcelerator families)

Clock Assignments

Use ChipPlanner to assign nets to local clocks in ProASIC and ProASIC PLUS.

You can also assign nets to local clocks and global clocks (with the exception noted below) by importing the following constraint files into Designer:

GCF (ProASIC and ProASIC PLUS families)

PDC (ProASIC3/E and Axcelerator families)

Note: You cannot assign global clocks for Axcelerator.

Timing Constraints

Timing constraints can be entered using the interactive Timer tool or by importing a constraint file.

Constraint File Type	Supported Families
SDC	Axcelerator, ProASIC PLUS, ProASIC3/E, SX-A, and eX
DCF	SX, SX-A, MX, eX, ACT1, ACT2, and ACT3
GCF	ProASIC ONLY

To understand the complexity of a design and its performance, perform place-and-route with no constraints to see if routing can complete without constraints. If routing completes successfully, you can open Timer to see if the physical design meets timing requirements.

ProASIC only: If you are using a synthesis tool such as Synopsys Design Compiler, Actel recommends that you use it to generate a forward SDF file containing path constraints only.

Over constraining a design may result in increased place-and-route run times, while not improving design performance.

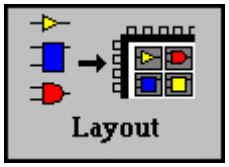
Layout

Running Layout

Use Layout to place and route your design.

To run Layout:

1. Click the **Layout** button in the Design Flow Window.



2. **Layout Options**. Select your Layout options and Click **OK**. Layout options are family specific.

Layout options

Axcelerator Layout Options

When running Layout, use the Layout Options dialog box to set your Layout options.

Timing-driven

Select this option to run timing-driven Layout. The primary goal of timing-driven layout is to meet timing constraints, with a secondary goal of producing high performance for the rest of the design. Timing-driven Layout is more precise and typically results in higher performance.

Standard layout is selected when the Timing-driven checkbox is unchecked. Standard layout maximizes the average performance for all paths. Each part of a design is treated equally for performance optimization. Standard layout uses net weighting (or criticality) to influence the results. Delay constraints that have been set for a design during place-and-route are not considered, however a delay report based on delay constraints entered in Timer can still be generated for the design. This is helpful to determine if timing-driven Layout is required.

Run Place

Select this option to run the placer during Layout. If you have not run Layout before, Run Place is checked by default. If your design has already been placed but not routed, this box is not checked. You can also select the following incremental placement options.

Incrementally: Select to use previous placement data as the initial placement for the next placement run.

Lock Existing Placement (fix): Select to use and lock previous placement data for the next incremental placement run.

Effort Level

Use the Effort Level slider to increase the effort Layout uses to place and route your design. The range is 1 to 5 with a default of 3. A higher level of effort generally improves the quality of results, but runs longer.

Run Route

Select to run the router during Layout. By default, it reflects the current Layout state. If you have not run Layout before, Run Route is checked. Run Route is also checked if your previous Layout run completed with routing failures. If your design has been routed successfully, this box is checked.

Incremental routing is available for Axcelerator devices. When activated, the option sets the previous routing information as the initial starting point. To use the incremental routing option in the script mode, see the Advanced Tcl Layout options for Axcelerator (in the Tcl Scripting section).

Use Multiple Passes

Select to run layout multiple times with different placement seeds. Multiple Pass Layout attempts to improve layout quality by selecting from a greater number of layout results. Click **Configure** to set your Multiple Pass Configuration.

Note: To run Multiple Passes, you must check both Run Place and Run Route.

ProASIC3/E, ProASICPLUS, and ProASIC Layout Options

When running layout, use the Layout Options dialog box to set your layout options.

Timing-driven

Select this option to run timing-driven Layout. The primary goal of timing-driven layout is to meet timing constraints, with a secondary goal of producing high performance for the rest of the design. Timing-driven Layout is more precise and typically results in higher performance.

When not checked, standard layout runs. Standard layout maximizes the average performance for all paths. Each part of a design is treated equally for performance optimization. Standard layout uses net weighting (or criticality) to influence the results. Delay constraints that have been set for a design during place-and-route are not considered, however a delay report based on delay constraints entered in Timer can still be generated for the design. This is helpful to determine if timing-driven Layout is required.

Run Place

Select this option to run the placer during Layout. By default, it reflects the current Layout state. If you have not run Layout before, Run Place is checked by default. If your design has already been placed but not routed, this box is not checked by default. You can also select the following incremental placement options.

Incrementally: Select to use previous placement data as the initial placement for the next place run.

Lock Existing Placement (fix): Select to preserve previous placement data during the next incremental placement run.

Run Route

Select to run the router during Layout. By default, it reflects the current Layout state. If you have not run Layout before, Run Route is checked. Run Route is also checked if your previous Layout run completed with routing failures. If your design has been routed successfully, this box is checked.

Incrementally: Select to fully route a design when some nets failed to route during a previous run. You can also use it when the incoming netlist has undergone an E.C.O. (Engineering Change Order). Incremental routing should only be used if a low number of nets fail to route (less than 50 open nets or shorted segments). A high number of failures usually indicates a less than optimal placement (if using manual placement through macros, for example) or a design that is highly connected and does not fit in the

device. If a high number of nets fail, relax constraints, remove tight placement constraints, or select a bigger device and rerun routing.

Use Multiple Passes

Select to run layout multiple times with different seeds. Multiple Pass Layout attempts to improve layout quality by selecting from a greater number of layout results. Click **Configure** to set your Multiple Pass Configuration.

Click the Advanced button to set Timing-Driven options.

ProASIC3/E, ProASICPLUS, and ProASIC Layout advanced Options

To set these advanced options during Layout, click the Advanced button in the Layout dialog box.

Placer Timing Weight

Setting this option to values within a recommended range of 1-4 changes the weight of the timing objective function, thus influencing the results of timing-driven place-and-route in favor of either routability or performance. This option is available only when timing constraints have been defined.

Note: If you change the Timing Weight you must re-run the placer to complete routing. Changing the Timing weight has no effect if you do not re-run the placer.

Restore Defaults

Click **Restore Defaults** to run the factory default settings for advanced options.

eX, SX, SX-A Layout Options

When running layout, use the Layout dialog box to set your layout options.

Timing-Driven

Select to run Timing-Driven Layout. The primary goal of Timing-Driven layout is to meet timing constraints, while still producing high performance for the rest of the design. Timing-Driven Layout is more precise and typically results in higher performance. This option is available only when timing constraints have been defined.

When not checked, standard layout runs. Standard layout maximizes the average performance for all paths. Each part of a design is treated equally for performance optimization. Standard layout uses net weighting (or criticality) to influence the results. Delay constraints that have been set for a design during place-and-route are not considered, however a delay report based on delay constraints entered in Timer can still be generated for the design. This is helpful to determine if Timing-Driven Layout is required.

Place Incrementally

Select to use previous placement data as the initial placement for the next place run.

Lock Existing Placement: Select to preserve previous placement data during the next incremental placement run.

Use Multiple Passes (eX and SX-A only)

Select to run layout multiple times with different seeds. Multiple Pass Layout attempts to improve layout quality by selecting from a greater number of layout results. Click **Configure** to set your Multiple Pass Configuration.

Advanced

Click the Advanced button to set Extended Run and Timing-Driven options.

eX, SX, and SX-A Advanced Layout Options

To set these advanced options during Layout, click the Advanced button in the Layout dialog box.

Extended Run

Select this to run a greater number of iterations during optimization within a single layout pass. An extended run layout can take up to 5 times as long as a normal layout.

Effort Level

This setting specifies the duration of the timing-driven phase of optimization during timing-driven Layout. Its value specifies the duration of this phase as a percentage of the default duration. This option is available only when timing constraints have been defined

The default value is 100 and the selectable range is within 25 - 500. Reducing the effort level also reduces the run time of timing-driven place-and-route (TDPR). With an effort level of 25, TDPR is almost four times faster. With fewer iterations, however, performance may suffer. Routability may or may not be affected. With an effort level of 200, TDPR is almost two times slower. This variable does not have much effect on timing.

Timing Weight

Setting this option to values within a recommended range of 10-150 changes the weight of the timing objective function, thus influencing the results of timing-driven place-and-route in favor of either routability or performance. This option is available only when timing constraints have been defined

The timing weight value specifies this weight as a percentage of the default weight (i.e. a value of 100 has no effect). If you use a value less than 100, more emphasis is placed on routability and less on performance. Such a setting would be appropriate for a design that fails to route with TDPR. In case more emphasis on performance is desired, set this variable to a value higher than 100. In this case, routing failure is more likely. A very high timing value weight could also distort the optimization process and degrade performance. A value greater than 150 is not recommended.

Restore Defaults

Click **Restore Defaults** to run the factory default settings for advanced options.

ACT, MX, and DX Layout Options

Timing-driven

Select this option to run Timing-Driven Layout. The primary goal of timing-driven layout is to meet timing constraints, with a

secondary goal of producing high performance for the rest of the design. timing-driven Layout is more precise and typically results in higher performance. This option is available only when timing constraints have been defined.

When not checked, standard layout runs. Standard layout maximizes the average performance for all paths. Each part of a design is treated equally for performance optimization. Standard layout uses net weighting (or criticality) to influence the results. Delay constraints that have been set for a design during place-and-route are not considered, however a delay report based on delay constraints entered in Timer can still be generated for the design. This is helpful to determine if Timing-Drive Layout is required.

Place Incrementally

Select to use previous placement data as the initial placement for the next place run.

Lock Existing Placement: Select to preserve previous placement data during the next incremental placement run.

Advanced

Click Advanced to set Extended Run options.

ACT, MX, and DX Advanced Layout Options

To set these advanced options during Layout, click the Advanced button in the Layout dialog box.

Extended Run

Select this to run a greater number of iterations during optimization. An extended run layout can take up to 5 times as long as a normal layout.

Restore Default

Click **Restore Defaults** to run the factory default settings for advanced options.

Incremental Placement

In either standard or timing-driven mode, use incremental placement to preserve the timing of a design after a successful place-and-route, even if you change part of the netlist. Incremental placement has no effect the first time you run layout. During design iteration, incremental placement attempts to preserve the placement information for any unchanged macros in a modified netlist.

As a result, the timing relationships for unchanged macros approximate their initial values, decreasing the execution time to perform Layout. By forcing Designer to retain the placement information for a portion of the design, some flexibility for optimal design layout may be lost. Therefore, do not use incremental placement to place your design in pieces. You should only use it if you have successfully run Layout and you have minor changes to your design.

Incremental placement requires prior completion of place. Do not use incremental placement if the previous Layout failed to meet performance goals.

Locking Existing Placement (Fix)

When the **Lock Existing Placement** option is selected in the Layout dialog box, all unchanged macros are treated as locked (fixed) placements during an incremental placement. This is the strongest level of control, but it may be too restrictive for the new placement to successfully complete. The default ON setting treats unchanged macro locations as placement hints, but alters their locations as needed to successfully complete placement. Refer to ChipEditor for details on locking macros.

ProASIC and ProASICPLUS Placement Constraint File (GCF)

For ProASIC and ProASIC^{PLUS} designs, you can export a GCF constraint file to get all of the constraint information. From the **File** menu, choose **Export>Constraint Files**, type a file name and click **Save**, and then select **All GCF constraints** in the **Export GCF File** dialog box. Blocks with locked placement constraints generate locked placement constraints, while the others generate initial placement constraints. You can edit a GCF file to remove existing constraints or add new constraints. You must then import the modified GCF file as well as the netlist back into Designer. See Importing Source Files for more information about importing files.

Multiple Pass Layout

Multiple Pass Layout attempts to improve layout quality by selecting from a greater number of Layout results. This is done by running individual place and route multiple times with varying placement seeds and measuring the best results with a specified criteria.

Note:

Before running Multiple Pass Layout, you need to save your design.

Multiple Pass Layout is supported in the following families: ProASIC3/E, Axcelerator, ProASIC^{PLUS}, ProASIC, SX-A, and eX.

Multiple Pass Layout saves your design file with the pass that has the best layout results. A corresponding timing report file for the best result, named design-name_timing.rpt is also saved to disk. If you want to preserve your existing design state, you should save your design file with a different name before proceeding. To do this, from the File menu, click Save As.

A timing report for each pass will be written out to the working directory to assist you in later analysis. The report files will be named design-name_timing.rpt.pass-number. Look at the design-name _iteration_summary.rpt for details of the saved files.

To configure your multiple pass options:

1. When running Layout, se;ect **Use Multiple Passes** in the Layout Options dialog box.
2. Click **Configure**. The Multi-Pass Configuration dialog box appears.
3. Set the options and click **OK**.

Maximum Number of Passes: Set the number of passes (iterations) using the slider. 3 is the minimum and 25 is the maximum. The recommended number of passes is 5.

Measurement: Select the measurement criteria you want Layout to meet. If Slowest Clock or Specific Clock is selected as your criterion, then the Layout runs all passes. If Timing Violations is selected as your criterion, Layout stops once the timing constraints are met. If the constraints are not met, then all of the Layout passes run.

Slowest Clock	Select to use the slowest clock in the design in a given pass as the performance reference for the layout pass.
Specific Clock	Select to use a specific clock as the performance reference for all Layout passes.
Timing Violations	Select to use the pass that best meets the slack or timing-violations constraints. NOTE: You must enter your own timing constraints through the Timer or SDC.The ‘best’ case is calculated by determining the total negative slack for all constraints.

Save Results from All Passes: Select to save the design file (.adb) for each pass. By default, only the best result is saved to your design. With this option, for every pass, the individual .adb is stored as filename_pass-number.adb in the name. The ‘best’ pass design will still also be written back to the original .adb filename. Saving all results does take more disk space, but allows you to later analyze the result of each pass in more detail. Look at the design-name _iteration_summary.rpt for details of the saved files.

The extended_run_shell Tcl script enables you to run the multiple pass layout in batch mode from a command line. See the extended_run_shell script for more information.

Analyzing Timing in Your Design

Use the Timer tool to analyze the timing in your design. Using Timer, you can:

Determine your clock frequency (a common yardstick for measuring the timing of your design)

Calculate delays in paths

Analyze and modify critical paths

Set timing constraints to meet your timing requirements

Incorporate the ChipPlanner/ChipEditor tools

Analyzing Power Consumption in Your Design

Use the SmartPower tool to analyze your designs power consumption. Use the SmartPower tool to:

Define clock domains

Specify individual pin frequencies

View detailed hierarchical analysis of your design

View global power consumption at the design level

If you wish, you may also view the equations that SmartPower uses to calculate your power consumption.

Viewing Your Netlist

The NetlistViewer tool displays the contents of the design as a schematic, making it easier for you to debug your design. With NetlistViewer, you can view nets, ports, and instances in the schematic view. You can also isolate specific sections of your netlist to simplify your analysis and cross probe with other tools.

There are two versions of the NetlistViewer tool: NetlistViewer in MultiView Navigator (MVN) and NetlistViewer Standalone. Which version you use depends on which family you are designing for.

NetlistViewer in MultiView Navigator supports ProASIC3E, ProASIC3, ProASIC PLUS, Axcelerator, and ProASIC families.

NetlistViewer Standalone supports MX, SX-A, eX, RTSX, and RTSX-S families.

Used with PinEditor in MultiView Navigator, ChipPlanner, or Timer, NetlistViewer in MultiView Navigator assists you in meeting area and timing goals by helping you with critical path identification. NetlistViewer Standalone can also be used alone or with PinEditor Standalone, ChipEditor, or Timer.

When you open your design (.adb) file, Designer will automatically present you with the appropriate tools in the Design Flow window. You must compile your design before you can open it in NetlistViewer.

Back-Annotation

The back-annotation functions are used to extract timing delays from your post layout data. These extracted delays are put into a file to be used by your CAE package's timing simulator. If you wish to perform pre-layout back-annotation, select Export and Timing Files from the File menu.

The Back-Annotation command creates the files necessary for back-annotation to the CAE file output type that you chose. Refer to Actel Interface Guides or the documentation included with your simulation tool for information about selecting the correct CAE output format and using the back-annotation files.

To back-annotate your design:

1. From the **Tools** menu, click **Back-Annotate**, or click the Back-Annotate button in the Design Flow window.
2. Make your selections in the Back-Annotate dialog box and click **OK**.

Extracted Files Directory: The file directory is your default working directory. If you wish to save the file elsewhere, click Browse and specify a different directory.

Extracted File Names: This name is used as the base-name of all files written out for back-annotation. Do not use directory names or file extensions in this field. The file extensions will be assigned based on your selection of which file formats to export. The default value of this field is <design>_ba.

Output Formats: Select the file format of the timing file. One of SDF or STF.

(STF is only supported for ACT1, ACT2, ACT3, DX, MX, SX).

Simulator Language: Select either Verilog or VHDL93.

Export Additional Files: Check Netlist or Pin to export these files at the same time. For Axcelerator, ProASIC3, and ProASIC3E, you must export and use the 'flattened' netlist (AFL-style) with the back-annotated timing file (SDF) in timing simulation.

Note: For Axcelerator, ProASIC3, and ProASIC3E, you cannot select SDF format using File -> Export Files -> Timing.

You have no choice regarding exporting the netlist from the back-annotate command for Axcelerator, ProASIC3, and ProASIC3E. This selection is hard-coded to be 'on'. For all other families, the export-netlist and back-annotate generates equivalent netlist files. So the back-annotate command does not enforce the writing out of the netlist during back-annotate.

Available Report Types

Select from Report Type & Options on the dialog box when invoking a report.

Report Type	Supported Families	Report Contents
Status Report	All	Provides information about Designer, Device Data, and variable settings for the design.
Timer Report	All	Displays summarized timing delays for paths.
Pin Report	All	Allows you to create a text list of the I/O signal locations on a device. You can generate a pin report sorted by I/O signal names or by package number.
FlipFlop Report	All	Creates a report that lists the number and type of flip-flops (sequential or CC, which are flip-flops made of 2 combinatorial macros) used in a design. The flip-flop report can be of two types: Summary or Extended. Both types of reports include the Flip-Flop type, sequential (Seq) or combinatorial (CC), the Library name, and the Total number of Seq and CC Flip-Flops in the design. The Summary Report also includes the Number of instances of each unique type. The Extended Report provides the Macro name. All Reports are output to an editable window for viewing, modifying, saving, and printing.
Power Report	ProASIC^{PLUS}, Axcelerator, ProASIC and ProaSIC3/E	Enables you to quickly determine if any power consumption problems exist in your design. The power report lists the following information: - Global device information and SmartPower Preferences selection information - Design level static power summary - Dynamic power summary - Hierarchical detailed power report (including gates, blocks, and nets), with a block by block, gate by gate, and net by net power summary SmartPower results
Timing Violations Report	All	Enables you to obtain constraint results sorted by slack. You can now view Max Delay violations as well as Min Delay violations in the report.
I/O Bank Report	Axcelerator and ProASIC3/E	Provides information on the I/O functionality, I/O technologies, I/O banks and I/O voltages.

Status Reports

The status report enables you to create a report containing device and design information, such as die, package, percentage of the logic and I/O modules used, etc.

To generate a status report:

1. In the **Tools** menu, choose **Reports**.
2. Select **Status** from the drop-down list in the **Report Types** dialog box. The status report opens in a separate window. You can save or print the report.

Timing Reports

The timing report enables you to quickly determine if any timing problems exist in your design. The timing report lists the following information about your design:

maximum delay from input I/O to output I/O

maximum delay from input I/O to internal registers

maximum delay from internal registers to output I/O

maximum delays for each clock network

maximum delays for interactions between clock networks

To generate a timing report:

1. In the **Tools** menu, click **Reports**.
2. Choose **Timing** from the Report Type drop-down list. This displays the Timing Report dialog box.
3. **Specify the Slack Threshold**. If you select “Slack” as the sort method, you can limit the number of delays displayed based upon a slack threshold. For example, if you only want to see delays that have a slack less than 5ns, enter 5 in the Slack Threshold box.
4. **Setup-hold Timing Check**. Selection of this box enables you to configure the timing report to calculate external setup and hold information for device inputs in addition to the standard information.
5. **Expand Failed Paths**. If a path does not meet your timing specifications, and you would like to see the incremental delay of each macro within that path, select the Expand Failed Paths box.
6. **Options**. Clicking Options brings up the Timing Preferences dialog box, where you can set additional display and report options.

   Sort by Actual Delay

   Sort by Slack Delay

   Path Selection

   Break Path at Register
7. Click **OK**. This displays a timing report based upon your timing and display preferences. The format and content of the report is determined by the family

Pin Reports

The pin report allows you to create a text list of the I/O signal locations on a device. You can generate a pin report sorted by I/O signal names or by package number.

To generate a pin report:

1. In the **Tools** menu, click **Reports**.
2. Choose **Pin** from the drop-down list in the Report Type dialog box. This displays the Pin Report dialog box.
3. Specify the type of report to generate. Select Number or Name from the List By pull-down menu, then click **OK**. This displays a pin report.

Flip-Flop Reports

The flip-flop report enables you to create a report that lists the number and type of flip-flops (sequential or CC, which are flip-flops made of 2 combinatorial macros) used in a design.

There are two types of reports you can generate, Summary or Extended:

A Summary report displays whether the flip-flop is a sequential, I/O sequential, or CC flip-flop, the macro implementation of the flip-flop, and the number of times the implementation of the flip-flop is used in the design.

An Extended report individually lists the names of the macros in the design.

To generate a flip-flop report:

1. In the **Tools** menu, click **Reports**. This displays the Reports dialog box.
2. Select Flip-Flop from the drop-down menu. The Flip-Flop Report dialog box appears.
3. Specify the type of report to generate. Select Summary or Extended from the Type pull-down menu, then click **OK**. This displays the report in a separate window.

Power Reports

The power report enables you to quickly determine if any power consumption problems exist in your design. The power report lists the following information:

Global device information and SmartPower Preferences selection information

Design level static power summary

Dynamic power summary

Hierarchical detailed power report (including gates, blocks, and nets), with a block by block, gate by gate, and net by net power summary SmartPower results

To create a power report:

1. In the **Tools** menu, click **Reports**. This displays the Reports dialog box.
2. Choose **Power** in the Report list and click **OK**. The Power Report dialog appears.
3. Choose from the following options:

   Static Power: Returns static power information

   Dynamic Power: Returns dynamic power information

   Report Style: Specifies report style

Select analysis preferences:

Units: Sets units preferences for power and frequency

Operating Conditions: Sets preferences for operating conditions

Block Expansion Control: Filters reported power values returned in the report. This box does not control which values are included, rather it specifies which blocks are detailed/expanded. You may specify which blocks are expanded using a minimum power value, a minimum power ratio (with regards to the total power of the design) and a maximum hierarchical depth; a filtered value is not include in displayed lists, but still counted for upper hierarchical levels.

4. Once you are satisfied with your selections, click **OK** in the Preferences dialog box and then click **OK** in the Power Report dialog box. SmartPower displays the report in a separate window.

Timing Violations Reports

For families that use the pin-to-pin timing model, the Violations report enables you to obtain constraint results sorted by slack. You can now view Max Delay violations as well as Min Delay violations in the report.

To generate a timing violations report:

1. From **Tools** menu, click **Reports**.
2. In the Report Types dialog box, select **Timing Violations.**
3. Click **OK**.

I/O Bank Reports

The I/O Bank report provides information about the I/O functionality, I/O technologies, I/O banks and I/O voltages.

The following section shows an excerpt from the I/O Bank report:

```
I/O Function:
    Type                          | w/o register  | w/ register   | w/ DDR register
    ------------------------------|---------------|---------------|----------------
    Input I/O                     | 20            | 0             | 0
    Output I/O                    | 17            | 0             | 0
    Bidirectional I/O             | 1             | 0             | 0
    Differential Input I/O Pairs  | 0             | 0             | 0
    Differential Output I/O Pairs | 0             | 0             | 0

I/O Technology:
                                  |   Voltages    |             I/Os
    ------------------------------|-------|-------|-------|--------|--------------
    I/O Standard(s)               | Vcci  | Vref  | Input | Output | Bidirectional
    ------------------------------|-------|-------|-------|--------|--------------
    LVTTL                         | 3.30v | N/A   | 20    | 17     | 1

I/O Bank Resource Usage:

          |   Voltages    | Single I/Os   | Diff I/O Pairs |        Vref I/Os
          |-------|-------|------|--------|-------|--------|------|-------|----------
          | Vcci  | Vref  | Used | Total  | Used  | Total  | Used | Total | Vref Pins
    ------|-------|-------|------|--------|-------|--------|------|-------|----------
    Bank0 | 3.30v | N/A   | 0    | 25     | 0     | 12     | N/A  | N/A   | N/A
    Bank1 | 3.30v | N/A   | 0    | 15     | 0     | 7      | N/A  | N/A   | N/A
    Bank2 | 3.30v | N/A   | 0    | 17     | 0     | 6      | N/A  | N/A   | N/A
    Bank3 | 3.30v | N/A   | 0    | 16     | 0     | 7      | N/A  | N/A   | N/A
    Bank4 | 3.30v | N/A   | 0    | 15     | 0     | 7      | N/A  | N/A   | N/A
    Bank5 | 3.30v | N/A   | 0    | 22     | 0     | 10     | N/A  | N/A   | N/A
    Bank6 | 3.30v | N/A   | 0    | 19     | 0     | 9      | N/A  | N/A   | N/A
    Bank7 | 3.30v | N/A   | 0    | 18     | 0     | 7      | N/A  | N/A   | N/A

I/O Voltage Usage:

       Voltages    |     I/Os
    ---------------|--------------
     Vcci  | Vref  | Used | Total
    -------|-------|------|-------
     3.30v | N/A   | 38   | 147
```

I/O Functionality

This section of the report indicates the total number of regular input, output, and bidirectional signals. This also shows the differential input and output in the design. The I/O categories are: regular I/Os, registered I/Os or DDR I/Os in the current design.

I/O Technologies

This section of the report specifies the VCCI and VREF voltage requirements for each I/O standard and the number of user I/Os per I/O standards used in the current design.

Note: For voltage referenced I/O standards, input and bidirectional I/Os require both a VCCI and a VREF power supply whereas output I/Os only require a VCCI power supply. This is why these two categories are shown separately in this section.

I/O Banks

This section of the report specifies the following I/O bank characteristics:

VCCI and VREF voltages assigned for each bank.

Total number of single-ended I/Os available and used for each bank.

Total number of differential I/O pairs available and used for each bank.

Total number of VREF pins assigned for each bank. This information is only relevant if a bank has been assigned a VREF voltage.

Total number of voltage referenced I/Os available and used for each bank. Voltage referenced I/Os are available only if VREF pins have been assigned.

I/O Voltages

This section of the report indicates the current design voltage requirements.

For each VCCI and VCCI/VREF user I/O demand in the current design, this table reports the total number of bonded I/Os available on the device that satisfy this demand.

Note: For an I/O bank assignment (VCCI and VREF assignment) to be valid for the current design, the I/O voltage table must show no violation. Violations are indicated with an asterix ("*") when the number of user I/Os that need a given VCCI or VCCI/VREF assignment is less than the total number of bonded I/Os that can satisfy this demand.

To generate IO bank report:

1. In the **Tools** menu, choose **Reports**. This displays the **Report Types** dialog box.
2. Select **IOBank** in the **Report types** and click **OK**. The IOBank report opens in a separate window. You can save or print the report.

Exporting Files

Designer supports different types of files to export.

Supported file types

How to export a file?

Export a GCF file

Export a PDC file

Supported file types

The following table shows a complete list of files that you can export along with the supported family.

Note: Designer does not support exporting VHDL 87 format.

Files	File Extension	Family
Actel Flattened Netlist	.afl	All
Actel Internal Netlist	.adl	All
Standard Delay Format	.sdf	All
STAMP	.mod, .data	SX-A, eX, Axcelerator, ProASIC3/E, ProASIC PLUS
Tcl script file	.tcl	All
Verilog Netlist	.v	All
VHDL Netlist	.vhd	All
EDIF Netlist file	.edn	All
Log File	.log	All

STAPL	.stp	ProASIC3/E, ProASIC, ProASIC PLUS
Bitstream	.bit	ProASIC3/E, ProASIC, ProASIC PLUS
Programming file (legacy)	.fus	ACT1, ACT2, ACT3, MX, XL, DX
Actel programming file	.afm	ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX, Axcelerator
Routing Segmentation file	.seg	ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX
Silicon Explorer Probe file	.prb	ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX, Axcelerator
Placement Location file	.loc	ProASIC3/E, ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX
ProASIC Constraints file	.GCF	ProASIC
ProASIC PLUS Constraints file	.GCF	ProASIC PLUS There is no timing constraint information when you export a GCF file in ProASIC PLUS; constraints are moved to an SDC file. The SDC file is not automatically exported when the GCF is exported; you must explicitly export the SDC file.
Combiner Info	.cob	ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX, Axcelerator, ProASIC3/E
BSDL file	.bsd	DX, 42MX, SX, SX-A, eX, Axcelerator, ProASIC3/E, ProASIC, ProASIC PLUS
Criticality	*.crt	ACT1, ACT2, ACT3, MX, XL, DX
PIN	*.pin	ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX
SDC	*.sdc	SX-A, eX, Axcelerator, ProASIC PLUS, ProASIC3/E
Physical Design Constraint	*.pdc	Axcelerator and ProASIC3/E
Value Change Dump	*.vcd	Axcelerator, ProASIC3/E, ProASIC, ProASIC PLUS
Switching Activity Intermediate File/Format	*.saif	Axcelerator, ProASIC3/E, ProASIC, ProASIC PLUS
Design Constraint file	*.dcf	ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX

How to export a file

To export a file:

1. From the **File** menu, select **Export** and then select the type of file you wish to export.
2. Specify file name and file type and click **OK**.

Note: You must compile your design before you export an SDC file.

To export a GCF file:

1. From the File menu, select Export > Constraint Files.
2. Enter the file name and click **OK**.
3. Choose the type of information you want to export:

   Pin locations: Select to export the I/O placement and region constraints related to the I/Os only.

   Placement constraints: Select to export all placement constraints, including I/O and core constraints.

   All GCF constraints: Select to export all constraint information.

4. Click **OK**.

To export a PDC file:

1. From the File menu, select Export > Constraint Files.
2. Type a file name and choose a directory for it. Click **Save**. The Export Physical Design Constraints (PDC) dialog box appears.
3. Choose the type of information you want to export:

   Pin locations and attributes: Select to export information about the pin locations and attributes only.

   Placement constraints: Select to export all user-defined constraints.

   Complete placement information: Select to export all the information about the I/O locations, I/O attributes, and logic placement.
4. Click **OK**.

A message appears in the Log window telling you if the Export command succeeded.

Note: The content of the exported PDC file depends on the state of your design.

Pre-Compile: The exported PDC file will contain constraints that are stored in the database from the last valid compile state. Export may fail if you did not run Compile at least once before running Export.

Post-Compile: The exported PDC file will contain constraints that are currently stored in the database.

Saving Your Design

Once you have imported a netlist and compiled a design, you can save the design as an ADB file.

To save your design as an ADB file:

1. In the **File** menu, click **Save** or click the save icon in the toolbar.
2. Enter the File name and click **Save**. The default file name is the name you previously entered in the setup dialog box. The default format is adb. Make sure your save in the ".adb" format.

Once you have saved your compiled design as an ADB file, during any future Designer sessions, you can open the ADB file, skipping the compile step, and perform optimization on the design, including updating netlist and auxiliary file information.

Exiting Designer

To end a Designer session, from the **File** menu, click **Exit**.

If the information has not been saved to disk, you are asked if your want to save the design before exiting. If you choose YES, the "<design_name>.adb" file is updated with information entered the current session. If you choose NO, the information is not saved and the "<design_name>.adb" file remains unchanged.

Generating Programming Files

Once you have completed your design, and you are satisfied with the back-annotated timing simulation, create your programming file. Depending upon your device family, you need to generate a Fuse, Bitstream or STAPL programming file.

ProASIC3/E devices use the FlashPoint program file generator to create a programming file. The FlashPoint interface enables the advanced security features in ProASIC3/E.

Programmer	Antifuse Programming File	Flash Programming File
FlashPro	N/A	.stp
Silicon Sculptor I	.afm (Non-Axcelerator families)	.bit
Silicon Sculptor II	.afm	.bit .stp (Windows only)

Starting Silicon Sculptor from Libero IDE

Before starting Silicon Sculptor, generate your programming file.

To start the programming tool software:

3. Right-click the design root file in the **Design Hierarchy** window.
4. Click **Run Silicon Sculptor.** Refer to the Silicon Sculptor User's Guide for information on using the programming tool.

Generate a Programming File

FlashPoint enables you to program security settings, FPGA Array, and FlashROM features for ProASIC3/E devices. You can program these features separately using different programming files or you can combine them into one programming file. Each feature is listed as a silicon feature in the GUI.

You can generate a programming file with one, two, or all of the silicon features from the **Generate Programming File** page.

To generate a programming file:

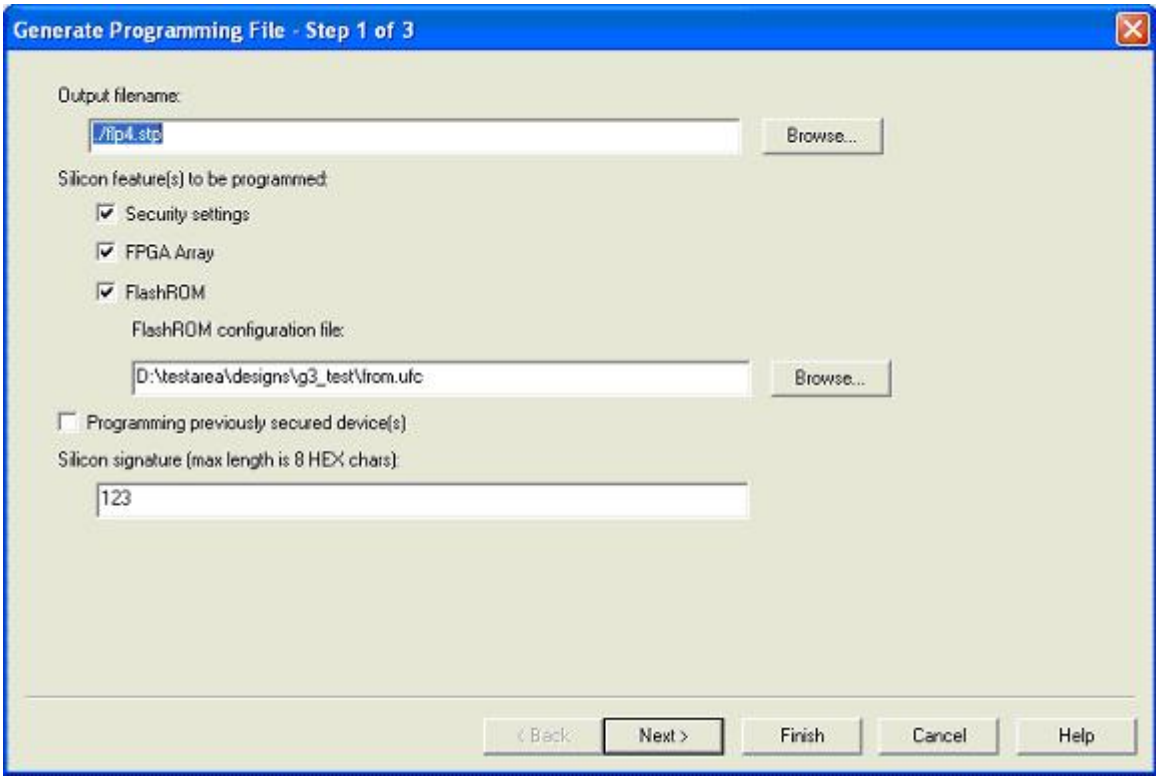
1. Enter the **Output file name**.
   Click the **Browse** button if you need to find your file or select your directory, and then enter the file name to save your output file.

2. Select the **Silicon feature(s)** you want to program.

   Security settings

   FPGA Array

   FlashROM



3. Click the **Programming previously secured device(s)** check box if you are reprogramming a device that has been secured.

   Because the ProASIC3/E family enables you to program the Security Settings separately from the FPGA Array and/or FlashROM, you must indicate if the Security Settings were previously programmed into the target device. This requirement also applies when you generate programming files for reprogramming.

4. Enter the silicon signature(0-8 HEX characters). See Silicon Signature for more information.

5. Click **Next**.

Silicon Signature

With Designer tools, you can use the silicon signature to identify and track Actel designs and devices. When you generate a programming file, you can specify a unique silicon signature to program into the device. This signature is stored in the design database and in the programming file, and programmed into the device during programming.

The silicon signature is accessible through the USERCODE JTAG instruction.

NOTE: If you set the security level to high, medium, or custom, you must program the silicon signature along with the Security Setting. If you have already programmed the Security Setting into the target device, you cannot reprogram the silicon signature without reprogramming the Security Setting.

Note: The previously programmed silicon signature will be erased if:

You have already programmed the silicon signature, and

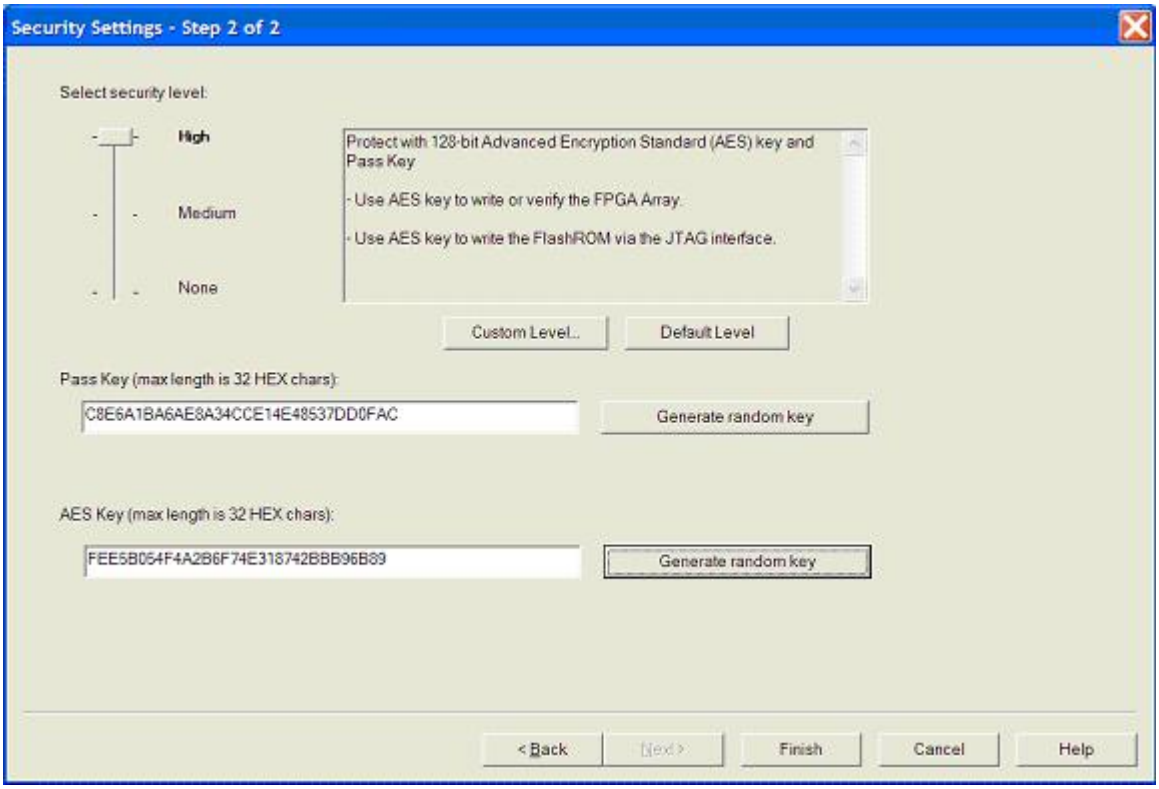
You are programming the security settings, but you do not have an entry in the silicon signature field

Programming Security Settings

FlashPoint allows you to set a security level of high, medium or none.

To program Security Settings on the device:

1. If you choose to program Security Settings on the device from the **Generate Programming File** page, the wizard takes you to the **Security Settings** page (see figure below).



2. Move the sliding bar to select the security level for FPGA and FlashROM (see table for a description of the security levels).

Security Level	Security Option	Description
High	Protect with a 128-bit Advanced Encryption Standard (AES) key and a Pass Key	Access to the device is protected by an AES Key and the Pass Key. The Write and Verify operations of the FPGA Array use a 128-bit AES encrypted bitstream. From the JTAG interface, the Write and Verify operations of the FlashROM use a 128-bit AES encrypted bitstream. Read back of

		the FlashROM content via the JTAG interface is protected by the Pass Key. Read back of the FlashROM content is allowed from the FPGA Array.
Medium	Protect with Pass Key	The Write and Verify operations of the FPGA Array require a Pass Key. From the JTAG interface, the Read and Write operations on the FlashROM content require a Pass Key. You can Verify the FlashROM content via the JTAG interface without a Pass Key. Read back of the FlashROM content is allowed from the FPGA Array.
None	No security	The Write and Verify operations of the FPGA Array do not require keys. The Read, Write, and Verify operations of the FlashROM content also do not require keys.

3. Enter the **Pass Key** and/ or the **AES Key** as appropriate. You can generate a random key by clicking the **Generate random key** button.

   The **Pass Key** protects all the Security Settings for the FPGA Array and/or FlashROM.

   The **AES Key** decrypts FPGA Array and/or FlashROM programming file content. Use the AES Key if you intend to program the device at an unsecured site or if you plan to update the design at a remote site in the future.

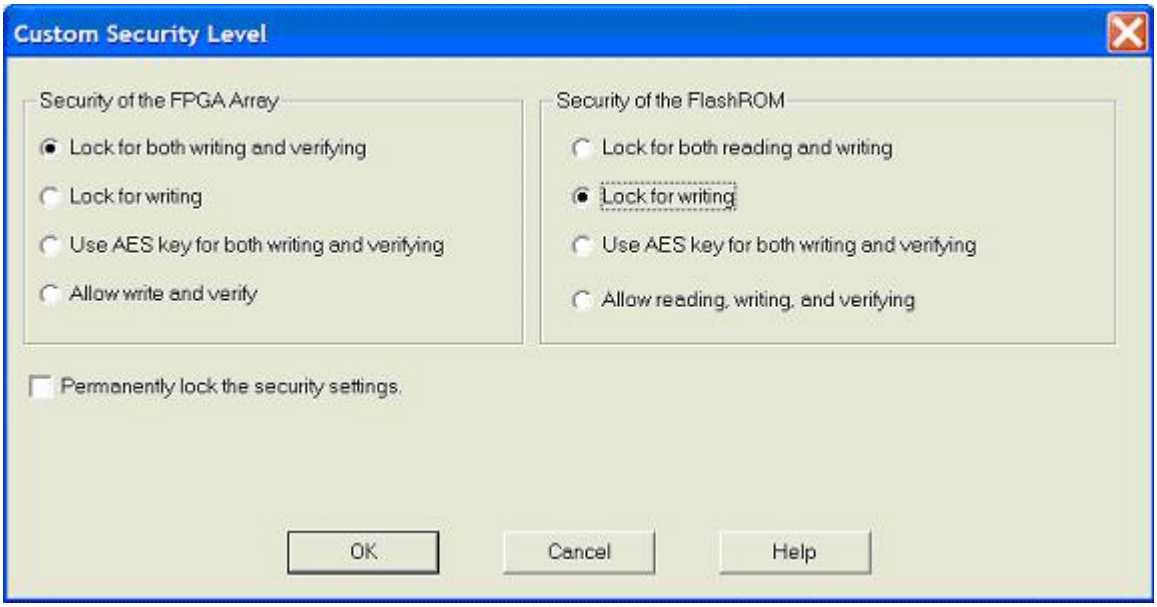
You can also customize the security levels by clicking the **Custom Level** button. For more information, see the Custom Security Levels section.

Custom Security Levels

For advanced use, you can customize your security levels.

To set custom security levels:

1. Click the **Custom Level** button in the **Setup Security** page. The **Custom Security** dialog box appears (see figure below).



2. Select the **FPGA Array Security** and the **FlashROM Security** levels.

   The FPGA Array and the FlashROM can have different Security Settings. See the tables below for a description of the custom security option levels for FPGA Array and FlashROM.

FPGA Array

Security Option	Description
Lock for both writing and verifying	Allows writing/erasing and verification of the FPGA Array via the JTAG interface only with a valid Pass Key.
Lock for writing	Allows the writing/erasing of the FPGA Array only with a valid Pass Key. Verification is allowed without a valid Pass Key.
Use the AES Key for both writing and verifying	Allows the writing/erasing and verification of the FPGA Array only with a valid AES Key via the JTAG interface. This configures the device to accept an encrypted bitstream for reprogramming and verification of the FPGA Array. Use this option if you intend to complete final programming at an unsecured site or if you plan to update the design at a remote site in the future. Accessing the device security settings requires a valid Pass Key.
Allow write and verify	Allows writing/erasing and verification of the FPGA Array with plain text bitstream and without requiring a Pass Key or an AES Key. Use this option when you develop your product in-house.

Note: The ProASIC3/E family FPGA Array is always read protected regardless of the Pass Key or the AES Key protection.

FlashROM

Security Option	Description
Lock for both reading and writing	Allows the writing/erasing and reading of the FlashROM via the JTAG interface only with a valid Pass Key. Verification is allowed

	without a valid Pass Key.
Lock for writing	Allows the writing/erasing of the FlashROM via the JTAG interface only with a valid Pass Key. Reading and verification is allowed without a valid Pass Key.
Use the AES Key for both writing and verifying	Allows the writing/erasing and verification of the FlashROM via the JTAG interface only with a valid AES Key. This configures the device to accept an encrypted bitstream for reprogramming and verification of the FlashROM. Use this option if you complete final programming at an unsecured site or if you plan to update the design at a remote site in the future. The bitstream that is read back from the FlashROM is always unencrypted (plain text).
Allow writing and verifying	Allows writing/erasing, reading and verification of the FlashROM content with a plain text bitstream and without requiring a valid Pass Key or an AES Key.

Note: The FPGA Array can always read the FlashROM content regardless of these Security Settings.

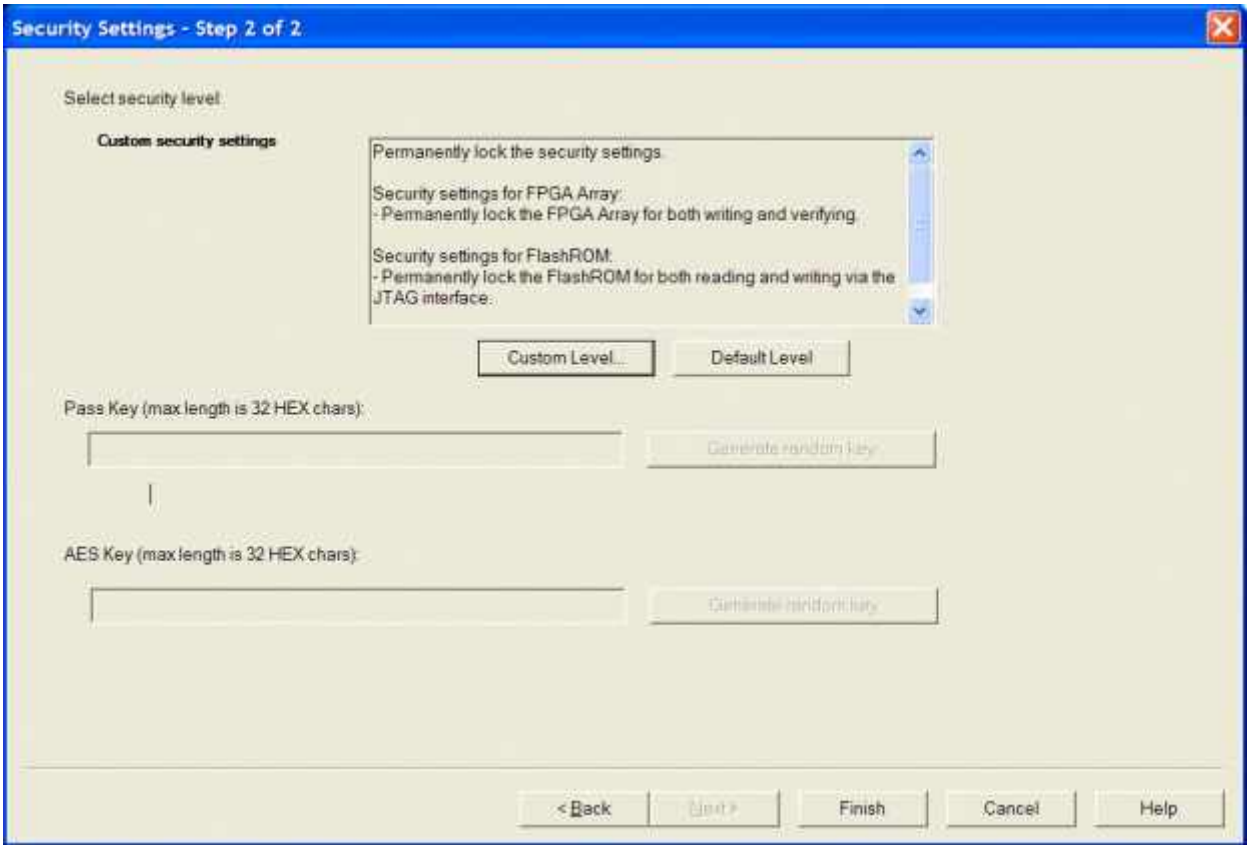
Note: To make the Security Settings permanent, select the **Permanently lock the security settings** check box. This option prevents any future modifications of the Security Setting of the device. A Pass Key is not required if you use this option.

When you make the Security Settings permanent, you can never reprogram the Silicon Signature. If you Lock the write operation for the FPGA Array or the FlashROM, you can never reprogram the FPGA Array or the FlashROM, respectively. If you use an AES key, this key cannot be changed once you permanently lock the device.

To use the Permanent FlashLock™ feature, select Disable Write and Verify for **FPGA Array** and Disable Read, Write and Verify for **FlashROM** and select the **Permanently lock the security settings** checkbox as shown in the figure below. This will make your device one-time-programmable.

3. Click the **OK** button.

The **Security Settings** page appears with the **Custom security setting** information as shown in the figure below.

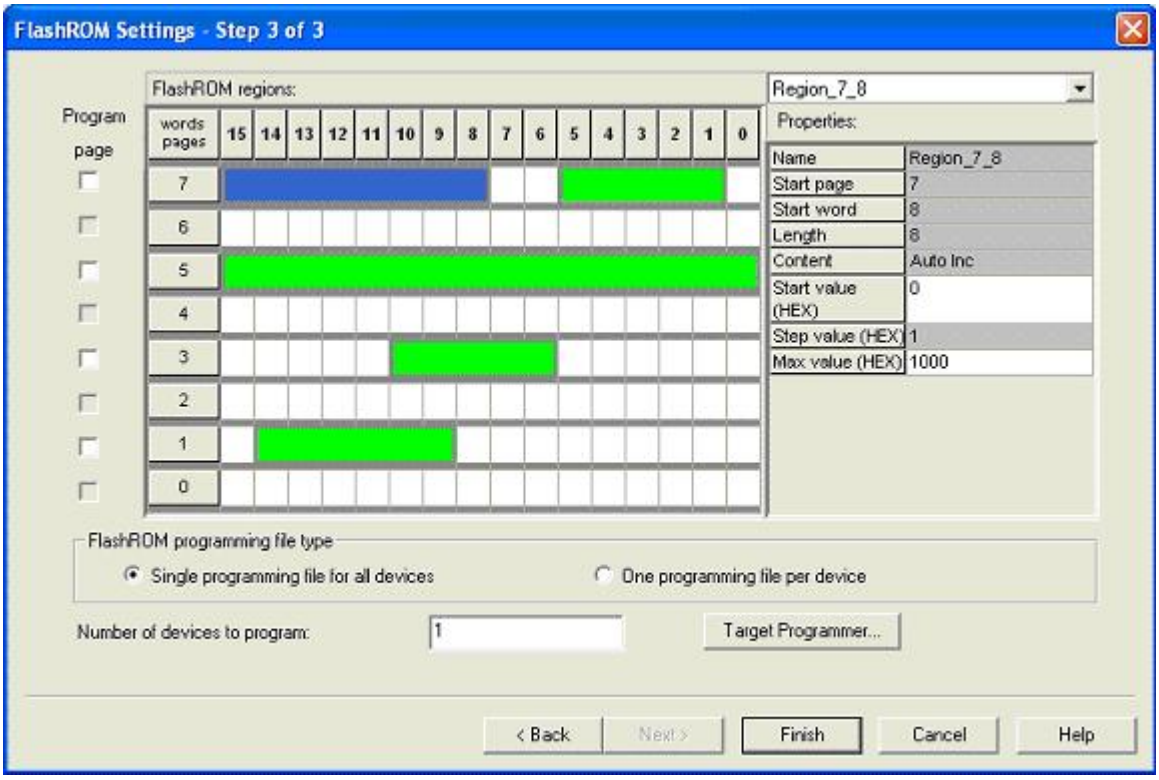


Programming the FlashROM

You can program selected memory pages and specify the region values of the FlashROM.

To program FlashROM:

1. Select FlashROM from the Generate Programming File page.
2. Enter the location of the FlashROM configuration file.
   The **FlashROM Settings** page appears (see figure below).

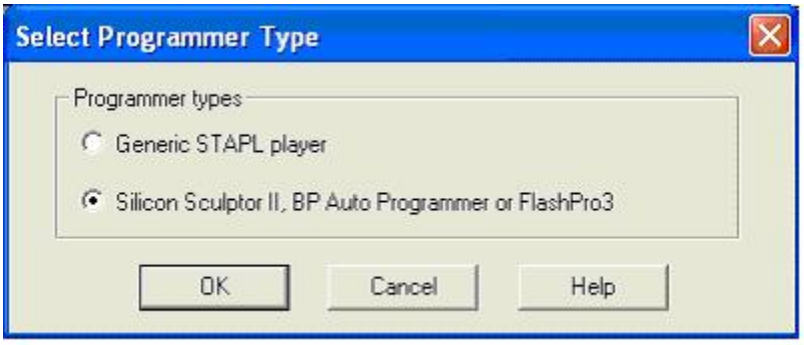


3. Select the FlashROM memory page that you want to program.
4. Enter the data value for the configured regions.
5. If you selected the region with a **Read From File**, specify the file location. See Custom Serialization Data for FlashROM Region for more information.
6. If you selected the **Auto Increment** region, specify the **Start** and **Max** values.
7. Complete steps 8 and 9 if you have a **Read from file** and/or **Auto Increment** region in the FlashROM.
8. Select the type of FlashROM programming files you want to generate from the two options below:

   Single programming file for all devices option: generates one programming file with all the generated increment values or with values in the custom serialization file.

   One programming file per device: generates one programming file for each generated increment value or for each value in the custom serialization file.

9. Enter the number of devices you want to program.
10. Click the **Target Programmer** button.
    The **Select Programmer Type** dialog box appears (see figure below).



11. Select your target **Programmer type**.
12. Click **OK**.
    FlashPoint generates your programming file.

Note: You cannot change the FlashROM region configuration from FlashPoint. You can only change the configuration from the ACTgen FlashROM core generator.

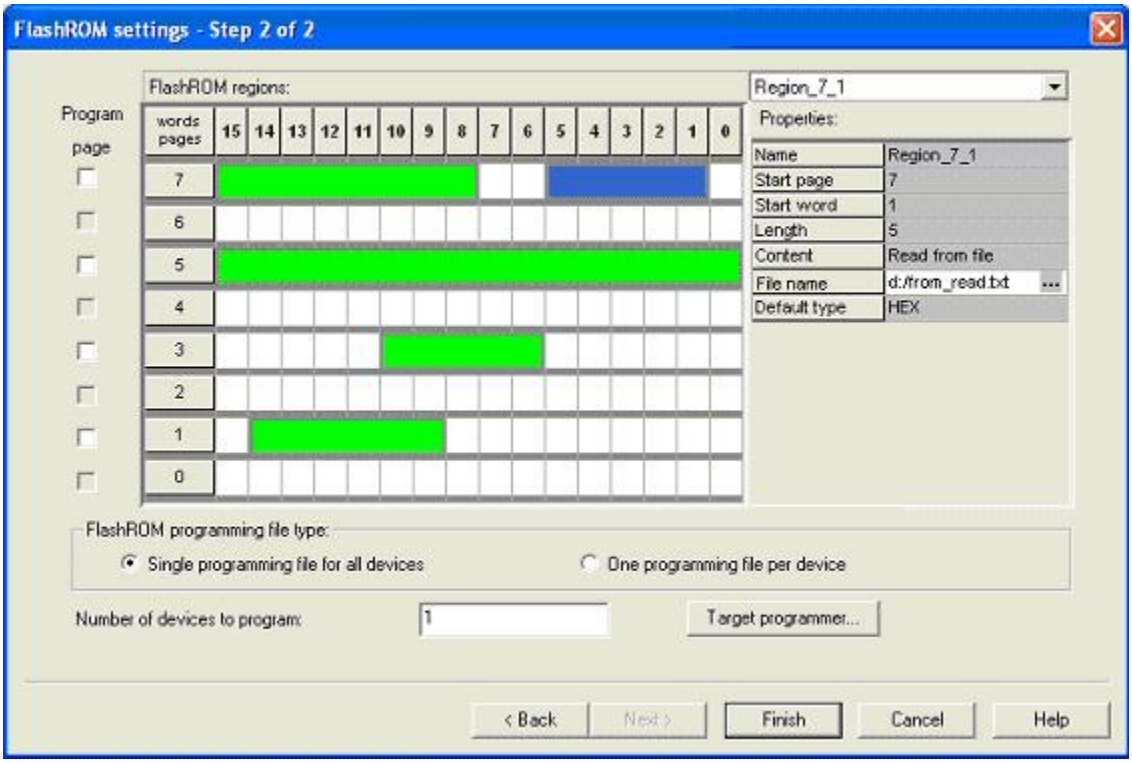
For more information, see ACTgen online help.

Custom Serialization Data for FlashROM region

FlashPoint enables you to specify a custom serialization file as a source to provide content for programming into a **Read from file FlashROM** region. You can use this feature for serializing the target device with a custom serialization scheme.

To specify a FlashROM region:

1. From the **Properties** section in the **FlashROM Settings** page, select the file name of the custom serialization file (see figure below). For more information on custom serialization files, see Custom Serialization Data File Format.



2. Select the FlashROM programming file type you want to generate from the two options below:

   - Single programming file for all devices option: generates one programming file with all the values in the custom serialization file.

   - One programming file per device: generates one programming file for each value in the custom serialization file.

3. Enter the number of devices you want to program.
4. Click the **Target Programmer** button.
5. Select your target **Programmer type**.
6. Click **OK**.

Custom Serialization Data File Format

FlashPoint supports custom serialization data files specifying the data in binary, HEX, decimal, or ASCII text. The custom serialization data files may contain multiple data with the Line Feed (LF) character as the delimiter.

Syntax

```
Custom serialization data file = <hex region data list> | <decimal region data list> |
        <binary region data list> | <ascii text data list>
```

```
Hex region data list = < hex data> <new line> { < hex data> <new line> }

Decimal region data list = <decimal data> <new line> {<decimal data><new line> }

Binary region data list = <binary data> <new line> { <binary data> <new line> }


ASCII text region data list = < ascii text data> <new line>

                    { < ascii text data> <new line> }


hex data = <hex digit> {<hex digit>}


decimal data = < decimal digit> {< decimal digit>}


binary data = < binary digit> {< binary digit>}


ASCII text data = <ascii character> {< ascii character >}


new line = LF


binary digit = ‘0’|‘1’


decimal digit = ‘0’|‘1’|‘2’|‘3’|‘4’|‘5’|‘6’|‘7’|‘8’| ‘9’


hex digit = ‘0’|‘1’|‘2’|‘3’|‘4’|‘5’|‘6’|‘7’|‘8’|‘9’|‘A’|‘B’|‘C’|‘D’ | ‘E’| ‘F’ |


‘a’| ‘b’ | ‘c’| ‘d’ | ‘e’| ‘f’


ascii character = characters from SP(0x20) to‘~’(0x7E)
```

Semantics

Each custom serialization file has only one type of data format (binary, decimal, hex or ASCII text). If a file contains a mixed format, it is considered an invalid file.

The length of each data must be shorter or equal to the selected region length. If the data is shorter then the selected region length, the most significant bits shall be padded with 0’s. If the specified region length is longer then the selected region length, it is considered an invalid file.

The digit / character length is as follows:

-Binary digit: 1 bit.

-Decimal digit: 4 bits.

-Hex digit: 4 bits.

-ASCII Character: 8 bits.

Hex serialization data file example

The following example is a Hex serialization data file for a 40-bit region:

```
123AEd210
AeB1
235SedF11 (This is an error: invalid hex digit)
0001242E
4300124EFE (This is an error: data out of range)
…
```

The following is an example of programming "AeB1" into Region_7_1 located on page 7, Word 5 to Word 1 in the **FlashROM settings** page. See Custom serialization data for FlashROM region for more information.

	Word 15	...	Word 6	Word 5	Word 4	Word 3	Word 2	Word 1	Word 0
Page 7	...	...	...	00	00	00	AE	B1	...

Binary serialization data file example

The following example is a binary serialization data file for a 16-bit region:

```
1100110011010001
100110011010011
11001100110101111  (This is an error: data out of range)
1001100110110111
1001100110110112  (This is an error: invalid binary digit)
```

Decimal serialization data file example

The following example is a decimal serialization data file for a 16-bit region:

```
65534
65535
65536  (This is an error: data out of range)
6553A  (This is an error: invalid decimal digit)
```

Text serialization data file example

The following example is a text serialization data file for a 32-bit region:

```
AESB
A )e
ASE3 23  (This is an error: data out of range)
65A~
```

```
1234
AEbF
```

Programming the FPGA Array

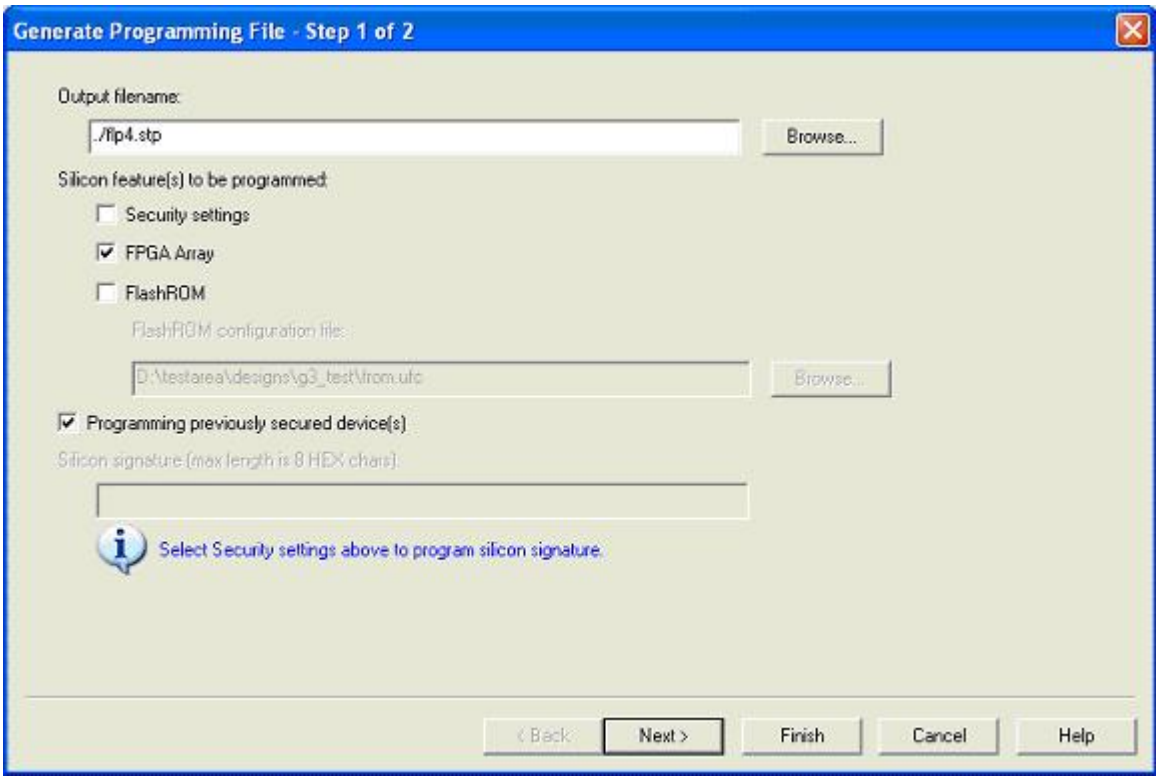
You can program the FPGA Array by selecting the silicon feature, **FPGA Array** in the **Generate Programming File** page and clicking **OK**. See Generate a programming file for more information.

Reprogramming a Secured Device

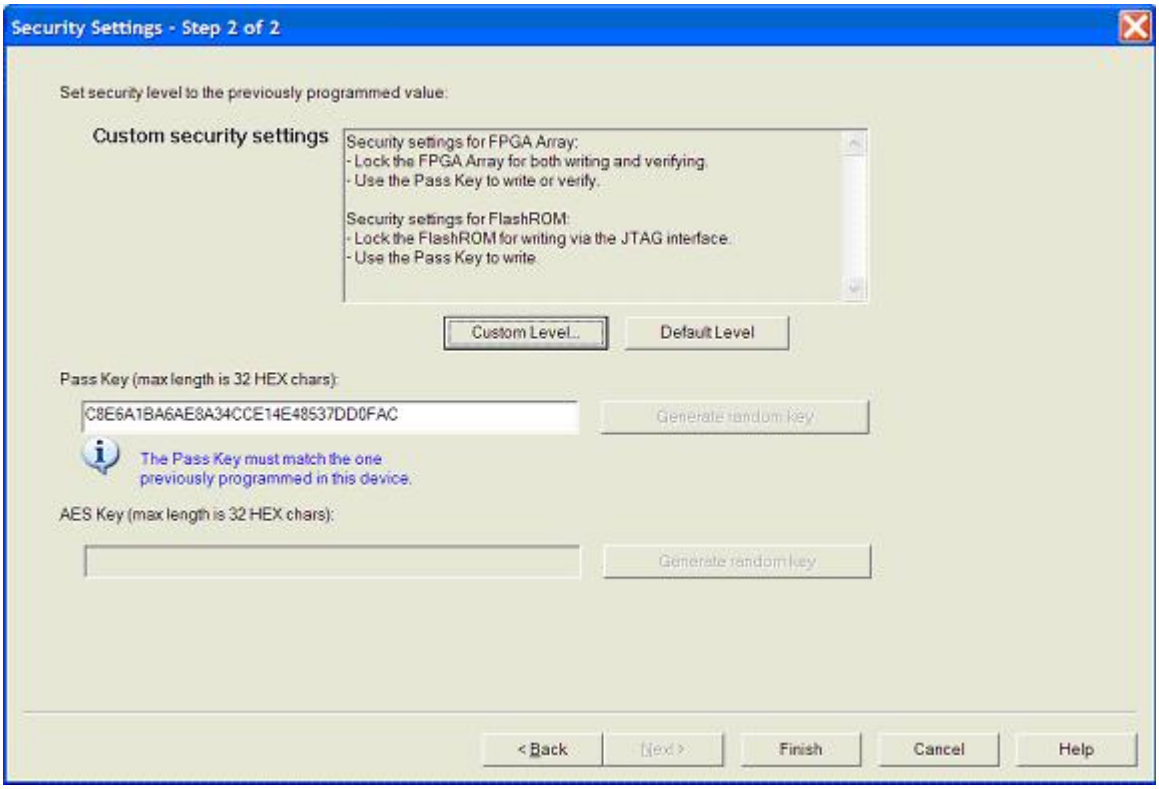
You must know the previous Security Settings of the device before you can reprogram a device with Security Settings.

To program a secured device:

1. In the Generate Programming File page, click the Programming previously secured devices(s) check box(see figure below).



2. Specify the previously programmed security setting for the FlashROM and/or the FPGA Array.
3. If you programmed the device with a custom security level, click the **Custom Level** button to open the **Custom security** dialog box, and select the **Security Settings** for the FPGA Array or the FlashROM that you programmed (see figure below).



4. Enter the previously programmed Pass Key and/or the AES Key.
5. Click **Finish**.

Note: Enter the AES Key only if you want to perform encrypted programming.

FlashLock

Actel's ProASIC and ProASIC^{PLUS} devices contain FlashLock circuitry to lock the device by disabling the programming and readback capabilities after programming. Care has been taken to make the locking circuitry very difficult to defeat through electronic or direct physical attack.

FlashLock has three security options: No Lock, Permanent Lock, and Keyed Lock.

No Lock

Creates a programming file which does not secure your device.

Permanent Lock

The permanent lock makes your device one time programmable. It cannot be unlocked by you or anyone else.

Keyed Lock

Within each ProASIC and ProASIC^{PLUS} device, there is a multi-bit security key user key. The number of bits depends on the size of the device. The tables below show the key size of different ProASIC and ProASIC^{PLUS} devices, respectively. Once secured, read permission and write permission can only be enabled by providing the correct user key to first unlock the device. The maximum security key for the device is shown in the dialog box.

Key Size of ProASIC Devices

Device	Key Size (bits)	Key Size (Hex)
A500K050	51 Bits	13
A500K130	51 Bits	13
A500K180	51 Bits	13
A500K270	51 Bits	13

Key Size of ProASIC^{PLUS} Devices

Device	Key Size (bits)	Key Size (Hex)
APA075	79 Bits	20
APA150	79 Bits	20
APA300	79 Bits	20
APA450	119 Bits	30
APA600	167 Bits	42
APA750	191 Bits	48
APA1000	263 Bits	66

Programming the Security Bit

Two device programmers, Silicon Sculptor and Flash Pro, are available for ProASIC and ProASIC^{PLUS} devices. If the programming file contains the security key, by default the Silicon Sculptor and Flash Pro programming software automatically enables the "secure" option and programs the security key. You can turn this off, should you decide not to program using the security key.

Please refer to the application note "Implementation of Security in Actel's ProASIC and ProASIC^{PLUS} Flash-Based FPGAs" for more details.

Generating Bitstream and STAPL Files

Bitstream allows you to generate a STAPL file for ProASIC, ProASIC^{PLUS}, and ProASIC3/E devices, or a bitstream file for ProASIC and ProASIC^{PLUS} families. Please consult the Program Files table to find out which file type you should choose.

To generate a bitstream or STAPL file:

1. In the **Tools** menu, click **Programming File** or click the Programming File button in the Design Flow window.
2. Select **Bitstream** or **STAPL** from the File Type drop-down list box. Bitstream files are not available for ProASIC3/E devices.
3. **FlashLock**. Select one of the following options:
4. No Locking: Creates a programming file which does not secure your device.
5. Use Keyed Lock: Creates a programming file which secures your device with a FlashLock key. The maximum security key for the device is shown in the dialog box. The maximum security key for the device is shown in the dialog box.
6. Use Permanent Lock: Creates a one-time programmable device.
7. Click **OK**. Designer validates the security key and alerts you to any concerns.

Note: The bitstream file header contains the security key.

Generating a Fuse File

Fuse allows you to generate a programming file for your Actel Antifuse devices. Fuse files work with Actel's Silicon Sculptor programmers. (For Axcelerator families, you must use the Silicon Sculptor II programmer.)

To generate a fuse programming file:

1. In the **Tools** menu, click **Programming File** or click the Programming File button in the Design Flow window.

   File Type. Select the appropriate file type in the File Type pull-down menu. Select “AFM-APS2” if you are using Silicon Sculptor programmer.

   Silicon Signature (Optional): Enter a 5 digit hexadecimal value in the Silicon Signature box to identify the design. Valid characters are “0” through “9,” and “a” through “f.”

   Output filename: Designer automatically names the file based on the <design_name>.adb file. You can change the name by entering it in the File Name box. Click Browse to change the directory. Do not add a file extension or suffix to the file name. The Designer software automatically adds the extension to the programming file name when you specify the programming format.

   Generate Probe File Also: This option automatically generates a PRB file for use with Silicon Explorer

   Disable clamping diode for unused I/O pins: (SX-A and eX families). Check box to disable clamping diode.

   Use the JTAG Reset Pull-up Resistor: (Axcelerator family) Select to enable pull-up resistors on the TRSTB pin (JTAG Reset pin which is active low). This is not part of the JTAG standard but can be useful if you want to make sure that the JTAG tap controller is not reset by mistake if the TRSTB pin is not connected. The pull-up resistor guarantees that if the pin is not driven to low (active), the pin is left in an inactive state (high).

   Use the Global Set Fuse: (Axcelerator family) Select to set flip-flops to a known state after power-up. If not selected all flip-flops are set to '0' at power up. If this option is used, all flip-flops are set to '1' at power up.

2. Click **OK** when finished to save the file.

Generating Prototype Files

When designing for RTAX-S, you can use the Axcelerator family of devices for prototyping. Please refer to the application note, Prototyping RTAX-S Using Axcelerator Devices for more information.

To generate prototype files:

1. From the **Tools** menu, click **Generate Prototype**. In the Generate Prototype Files dialog box, make the following selections:

   Silicon Signature (Optional). Enter a 5 digit hexadecimal value in the Silicon Signature box to identify the design. Valid characters are “0” through “9,” and “a” through “f.”

   Output filename. Designer automatically names the file based on the <design_name>.adb file. You can change the name by entering it in the File Name box. Click Browse to change the directory. Do not add a file extension or suffix to the file name. The Designer software automatically adds the extension to the programming file name when you specify the programming format.

   Generate Probe File Also. This option automatically generates a .prb file for use with Silicon Explorer

   Use the JTAG Reset Pull-up Resistor: Select to enable pull-up resistors on the TRSTB pin (JTAG Reset pin which is active low). This is not part of the JTAG standard but can be useful if you want to make sure that the JTAG tap controller is not reset by mistake if the TRSTB pin is not connected. The pull-up resistor guarantees that if the pin is not driven to low (active), the pin is left in an inactive state (high).

   Use the Global Set Fuse: Select to set flip-flops to a known state after power-up. If not selected all flip-flops are set to '0' at power up. If this option is used, all flip-flops are set to '1' at power up.

2. Click **OK**. The AFM file is generated.

About Tcl Commands

A Tcl (Tool Command Language) file contains scripts for simple or complex tasks. You can run scripts from either the Windows or UNIX command line or store and run a series of Tcl commands in a ".tcl" batch file. You can also run scripts from within Designer.

Designer supports the following Tcl scripting commands:

Command	Action
backannotate	Extracts timing delays from your post layout data
close_design	Closes the current design
compile	Performs design rule check and optimizes the input netlist before translating the source code into machine code
export	Converts a file from its current format into the specified file format, usually for use in another program
extended_run_shell	Runs multiple iterations of layout through Designer
get_defvar	Returns the value of the Designer internal variable you specify
get_design_filename	Returns the fully qualified path of the specified design file
get_design_info	Returns detailed information about your design, depending on which arguments you specify
import_aux	Imports the specified file as an auxiliary file, which are not audited and do not require you to re-compile the design
import_source	Imports the specified file as a source file, which include your netlist and design constraints
is_design_loaded	Returns True if the design is loaded into Designer; otherwise, returns False
is_design_modified	Returns True if the design has been modified since it was last compiled; otherwise, returns False
is_design_state_complete	Returns True if the specified design state is complete (for example, you can inquire as to whether a die and package has been selected for the design); otherwise, returns False
layout	Place-and-route your design
layout (advanced options for the SX family)	Sets advanced place-and-route features for SX family designs
layout (advanced options for ProASIC)	Sets advanced place-and-route features for ProASIC family designs
new_design	Creates a new design (.adb) file in a specific location for a particular design family such as Axcelerator or ProASIC3
open_design	Opens an existing design in the Designer software
pin_assign	Assigns the named pin to the specified port but does not lock its assignment.
pin_commit	Saves the pin assignments to the design (.adb) file.
pin_fix	Locks the pin assignment for the specified port, so the pin cannot be moved during place-and-route.
pin_fix_all	Locks all the assigned pins on the device so they cannot be moved during place-and-route.
pin_unassign	Unassigns a specific pin from a specific port. The unassigned pin location is then available for other ports.
pin_unassign_all	Unassigns all pins from a specific port.
pin_unfix	Unlocks the specified pin from its port.

report	Generates the type of report you specify: Status, Timing, Timer Violations, Flip-flop, Power, Pin, or I/O Bank
save_design	Writes the design to the specified filename
set_defvar	Sets the value of the Designer internal variable you specify
set_design	Specifies the design name, family and path in which Designer will process the design
set_device	Specifies the type of device and its parameters
smartpower_add_pin_in_domain	Adds a pin to either a Clock or Set domain
smartpower_commit	Saves the changes made in SmartPower to the design file (.adb) in Designer
smartpower_create_domain	Creates a new clock or set domain
smartpower_remove_domain	Removes an existing domain
smartpower_remove_pin_frequency	Removes the frequency of an existing pin
smartpower_remove_pin_of_domain	Removes a clock pin or a data pin from a Clock or Set domain, respectively.
smartpower_restore	Restores previously committed constraints
smartpower_set_domain_frequency	Sets the frequency of a domain
smartpower_set_pin_frequency	Sets the frequency of an existing pin
timer_add_clock_exception	Adds an exception to or from a pin with respect to a specified clock
timer_add_pass	Adds the pin to the list of pins for which the path must be shown passing through in the timer
timer_add_stop	Adds the specified pin to the list of pins through which the paths will not be displayed in the timer
timer_commit	Saves the changes made to constraints in Timer into the Designer database.
timer_get_path	Displays the Timer path information in the Log window
timer_get_clock_actuals	Displays the actual clock frequency in the Log window
timer_get_clock_constraints	Displays the clock constraints (period/frequency and dutycycle) in the Log window
timer_get_maxdelay	Displays the maximum delay constraint between two pins of a path in the Log window
timer_get_path_constraints	Displays the path constraints set for maxdelay in the Timer in the Log window
timer_remove_clock_exception	Removes the previously set clock constraint
timer_remove_pass	Removes the previously entered path pass constraint

timer_remove_stop	Removes the path stop constraint on the specified pin
timer_restore	Restores previously committed constraints
timer_setenv_clock_freq	Sets a required clock frequency, in MHz, for the specified clock
timer_setenv_clock_period	Sets the clock period constraint for the specified clock
timer_set_maxdelay	Adds a maximum delay constraint for the path
timer_remove_all_constraints	Removes all the timing constraints previously entered in the Designer system

Note: Tcl commands are case sensitive. However, their arguments are not.

See Also

Tcl documentation conventions

Introduction to Tcl scripting

Basic syntax

Variables

Command substitution

Quotes and braces

Control structures

Lists and arrays

Print statement and Return values

Types of Tcl commands

Running Tcl scripts from the command line

Running Tcl scripts from within Designer

Exporting Tcl scripts

Sample Tcl scripts

Tcl Documentation Conventions

The following table shows the typographical conventions used for the Tcl command syntax.

Syntax Notation	Description
`command -argument`	Commands and arguments appear in `Courier New` typeface.
`variable`	Variables appear in blue, italic `Courier New` typeface. You must substitute an appropriate value

	for the variable.
[-argument *value*] [*variable*]+	Optional arguments begin and end with a square bracket with one exception: if the square bracket is followed by a plus sign (+), then users must specify at least one argument. The plus sign (+) indicates that items within the square brackets can be repeated. Do not enter the plus sign character.

Note: All Tcl commands are case sensitive. However, their arguments are not.

Examples

Syntax for the get_defvar command followed by a sample command:

```
get_defvar variable
```

```
get_defvar "DESIGN"
```

Syntax for the backannotate command followed by a sample command:

```
backannotate [-dir directory_name] -name filename -format format_type -language
language [-netlist][-pin]
```

```
backannotate -dir \
 {..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
```

Wildcard Characters

You can use the following wildcard characters in names used in Tcl commands:

Wildcard	What it does
\	Interprets the next character literally
?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Note: The matching function requires that you add a slash (\) before each slash in the port, instance, or net name when using wildcards in a PDC command and when using wildcards in the Find feature of the MultiView Navigator. For example, if you have an instance named "A/B12" in the netlist, and you enter that name as "A\\/B*" in a PDC command, you will not be able to find it. In this case, you must specify the name as A\\\\/B*.

Special Characters ([ ], { }, and \)

Sometimes square brackets ([ ]) are part of the command syntax. In these cases, you must either enclose the open and closed square brackets characters with curly brackets ({ }) or precede the open and closed square brackets ([ ]) characters with a backslash (\). If you do not, you will get an error message.

For example:

```
pin_assign -port {LFSR_OUT[0]} -pin 15
```

or

```
pin_assign -port LFSR_OUT\[0\] -pin 180
```

Note: Tcl commands are case sensitive. However, their arguments are not.

Entering Arguments on Separate Lines

To enter an argument on a separate line, you must enter a backslash (\) character at the end of the preceding line of the command as shown in the following example:

```
backannotate -dir \
 {..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
```

See Also

Introduction to Tcl scripting

Basic syntax

About Tcl commands

backannotate

The backannotate command is equivalent to executing the Back-Annotate command within the Tools menu. You can export an SDF file, after layout, along with the corresponding netlist in the VHDL or Verilog format. These files are useful in backannotated timing simulation.

```
Backannotate -name file_name -format format_type -language language -dir
directory_name [-netlist] [-pin]
```

Arguments

`-name file_name`

Use a valid file name with this option. You can attach the file extension .sdf to the File_Name, otherwise the tool will append .sdf for you.

`-format format_type`

Only SDF format is available for back annotation

`-language language`

The supported Language options are

VHDL93 – For VHDL-93 style naming in SDF

VERILOG – For Verilog style naming in SDF

`-dir directory_name`

Specify the directory in which all the files will be extracted.

`-netlist`

Forces a netlist to be written. The netlist will be either in Verilog or VHDL depending on the

`-pin`

Designer exports the pin file with this option. The .pin file extension is appended to the design name to create the pin file.

Supported Families

All

Notes

We advise you to export both SDF and the corresponding VHDL/Verilog files. This will avoid name conflicts in the simulation tool.

Designer must have completed layout before this command can be invoked, otherwise the command will fail.

Exceptions

-pin is not supported forProASIC and ProASICPLUS families.

Examples

Example 1:

```
backannotate
```

Uses default arguments and exports SDF file for back annotation

Example 2:

```
backannotate -dir \
 {..\my_design_dir} -name "fanouttest_ba.sdf" -format "SDF" -language \ "VHDL93" -netlist
```

This example uses some of the options for VHDL

Example 3:

```
backannotate -dir \
 {..\design} -name "fanouttest_ba.sdf" -format "SDF" -language "VERILOG" \
-netlist
This example uses some of the options for Verilog
```

Example 4:

```
If  { [catch { backannotate -name "fanouttest_ba" -format "SDF" } ]} {
              Puts “Back annotation failed”
              # Handle Failure
} else {
              Puts “Back annotation successful”
              # Proceed with other operations
}
```

You can catch exceptions and respond based on the success of backannotate operation

close_design

The close_design command closes the current design and brings Designer to a fresh state to work on a new design.

```
close_design
```

Arguments

None

Supported Families

All

Notes

This is equivalent to selecting the Close command in the File menu.

Exceptions

None

Example

```
if  { [catch { close_design }] {
              Puts “Failed to close design”
              # Handle Failure
} else {
              puts “Design closed successfully”
              # Proceed with processing a new design
}
```

See Also

open_design, close_design, new_design

compile

The compile command performs design rule check on the input netlist. Compile also performs some optimizations on the design through logic combining and buffer tree modifications. Compile options vary by family.

Supported Families

MX, SX, SX-A, and eX

Axcelerator

ProASIC, ProASIC^{PLUS}

ProASIC3/E

Compile Tcl arguments available for MX, SX, SX-A, and eX

```
compile -nl_pins_overwrite
```

Arguments

`-nl_pins_overwrite`

Overwrites the imported netlist with the changes made in PinEditor.

Notes

None

Exceptions

None

Example

```
if  { [catch { compile nl_pins_overwrite }] {
              Puts Failed compile
              # Handle Failure
} else {
              puts Compile successful
              # Proceed to Layout
}
```

Compile Tcl arguments available for Axcelerator

```
compile -combine_register value
```

Arguments

`-combine_register value`

The value should be ON for this optimization to take effect.

The following table shows the acceptable values for this argument:

Value	Description
on	Combines registers at the I/O into I/O-Registers.
off	Does not combine registers at the I/O

Notes

None

Exceptions

None

Example

```
compile -combine_register ON
```

Compile Tcl arguments available for ProASIC and ProASICPLUS

```
compile -ram_io_region <ON|OFF>
```

Arguments

`-ram_io_region value`

The following table shows the acceptable values for this argument:

Value	Description
ON	Includes RAM and I/O in the use_global and set_net_region constraints and in spines (LocalClock regions) created with the ChipPlanner tool in the MultiView Navigator
OFF	Does not include RAM and I/O in the use_global and set_net_region constraints and in spines (LocalClock regions)

	created with the ChipPlanner tool in the MultiView Navigator.

Default: ON for new designs; OFF for designs created with Designer v5.1 and earlier.

Notes

None

Exceptions

None

Examples

```
compile -ram_io_region OFF
compile -ram_io_region ON
```

Compile Tcl arguments available for ProASIC3/E

```
compile
   -pdc_abort_on_error value
    -pdc_eco_display_unmatched_objects value
    -pdc_eco_max_warnings value
    -demote_globals value
    -demote_globals_max_fanout value
    -promote_globals value
    -promote_globals_min_fanout value
    -promote_globals_max_limit value
    -localclock_max_shared_instances value
    -localclock_buffer_tree_max_fanout value
    -combine_register value
    -delete_buffer_tree value
    -delete_buffer_tree_max_fanout value
    -report_high_fanout_nets_limit value
```

Arguments

`-pdc_abort_on_error value`

Changes the “Abort on PDC error” behavior. The following table shows the acceptable values for this argument:

Value	Description
ON	Stops the flow if any error is reported in reading your PDC file
OFF	Skips errors in reading your PDC file and just report them as warnings.

Default: ON

Note: The flow always stops in the following two cases (even if this option is OFF):

If there is a Tcl error (for example, the command does not exist or the syntax of the command is incorrect)

The assign_local_clock command for assigning nets to LocalClocks fails. This may happen if any floor planning DRC check fails, such as, region resource check, fix macro check (one of the load on the net is outside the LocalClock region). If such an error occurs, then the Compile command fails. Correct your PDC file to proceed.

`-pdc_eco_display_unmatched_objects value`

Displays netlist objects in PDC that are not found in the imported netlist during Compile ECO mode.The following table shows the acceptable values for this argument:

Value	Description
ON	Reports netlist objects not found in the current netlist when reading the internal ECO PDC constraints
OFF	Specifies not to report netlist objects not found in the current netlist when reading the internal ECO PDC constraints

Default: OFF

`-pdc_eco_max_warnings value`

Defines the maximum number of errors/warnings in Compile ECO mode.

The value is the maximum number of error/warning messages to be displayed in the case of reading ECO constraints.

Default: 10000.

`-demote_globals value`

Enables/disables global clock demotion of global nets to regular nets. The following table shows the acceptable values for this argument:

Value	Description
OFF	Disables global demotion of global nets to regular nets
ON	Enables global demotion of global nets to regular nets

Default: OFF

`-demote_globals_max_fanout value`

Defines the maximum fanout value of a demoted net;where value is the maximum value
Default: 12

Note: A global net is not automatically demoted (assuming the option is on) if the resulting fanout of the demoted net (if it was demoted) is greater than the max fanout value. Actel recommends that the automatic global demotion only act on small fanout nets. Actel recommends that you drive high fanout nets with a clock network in the design to improve routability and timing.

`-promote_globals value`

Enables/disables global clock promotion. The following table shows the acceptable values for this argument:

Value	Description
ON	Enables global promotion of nets to global clock network
OFF	Disables global promotion of nets to global clock network

Default: OFF

`-promote_globals_min_fanout value`

Defines the minimum fanout of a promoted net; where value is the minimum fanout of a promoted net.
Default:200

`-promote_globals_max_limit value`

Defines the maximum number of nets to be automatically promoted to global The default value is 0. This is not the total number as nets need to satisfy the minimum fanout constraint to be promoted. The promote_globals_max_limit value does not include globals that may have come from either the netlist or PDC file (quadrant clock assignment or global promotion).

Note: Demotion of globals through PDC or Compile is done before automatic global promotion is done.

Note: You may exceed the number of globals present in the device if you have nets already assigned to globals or quadrants from the netlist or by using a PDC file. The automatic global promotion adds globals on what already exists in the design.

`-localclock_max_shared_instances value`

Defines the maximum number of shared instances allowed to perform the legalization. This option is also available for quadrant clocks.

value is the maximum number of instances allowed to be shared by 2 LocalClock nets assigned to disjoint regions to perform the legalization (default is 12, range is 0-1000). If the number of shared instances is set to 0, no legalization is performed.

Note: If you assign quadrant clocks to nets using MultiView Navigator, no legalization is performed.

`-localclock_buffer_tree_max_fanout value`

Defines the maximum fanout value used during buffer insertion for clock legalization. This option is also available for quadrant clocks.

Set *value* to 0 to disable this option and prevent legalization (default value is 12, range is 0-1000). If the value is set to 0, no buffer insertion is performed. If the value is set to 1, there will be one buffer inserted per pin.

`-combine_register value`

Combines registers at the I/O into I/O-Registers. The following table shows the acceptable values for this argument:

Value	Description
ON	Combines registers at the I/O into I/O-Registers
OFF	Does not optimize and combine registers at the I/O.

Default: OFF

`-delete_buffer_tree value`

Enables/disables buffer tree deletion on the global signals. The buffer and inverter are deleted. The following table shows the acceptable values for this argument:

Value	Description
ON	Enables buffer tree deletion from the netlist
OFF	Disables buffer tree deletion from the netlist

Default: OFF

`-delete_buffer_tree_max_fanout` *value*

Defines the maximum fanout of a net after buffer tree deletion;

value is the maximum value; the default value is 12.

Note: A net does not automatically remove its buffer tree (assuming the option is on) if the resulting fanout of the net (if the buffer tree was removed) is greater than the max fanout value. Actel recommends that the automatic buffer tree deletion only act on small fanout nets. Actel recommends that you drive high fanout nets with a clock network in the design to improve routability and timing.

`-report_high_fanout_nets_limit` *value*

Enables flip-flop net sections in the compile report and defines the number of nets to be displayed in the high fanout.

Default: 10

Notes

None

Exceptions

None

Examples

```
compile \
    -pdc_abort_on_error "ON" \
    -pdc_eco_display_unmatched_objects "OFF" \
    -pdc_eco_max_warnings 10000 \
    -demote_globals "OFF" \
    -demote_globals_max_fanout 12 \
    -promote_globals "OFF" \
    -promote_globals_min_fanout 200 \
    -promote_globals_max_limit 0 \
    -localclock_max_shared_instances 12 \
    -localclock_buffer_tree_max_fanout 12 \
    -combine_register "OFF" \
    -delete_buffer_tree "OFF" \
    -delete_buffer_tree_max_fanout 12 \
    -report_high_fanout_nets_limit 10
```

export

The syntax and arguments for the **export** command vary depending on which device you are designing. Click the appropriate command in the following list to see the syntax, arguments, and other information:

export (ProASIC3/E)

export (ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A)

---

extended_run_shell

This script runs multiple iterations of layout through Designer. Use this script from the tcl shell "acttclsh". **This is the script or command-line equivalent to using the multiple-pass layout (in the GUI).**

```
$ACTDIR/bin/acttclsh $ACTDIR/scripts/extended_run_shell.tcl
       -adb adbFilename [-n numIterations] [-timing_driven]
       [-c clockname] [-compare_script compareScript] [-save_all]
       [-stop_on_success] [-continue_from_last_seed]
       [-place value] [-incremental_place value]
       [-incremental_route value]
       [-effort_level n] [-timing_weight n]
```

Arguments

`$ACTDIR`

Location of the installation directory for Libero/Designer.

`[-adb adbFilename]`

This is the design file to run mulitple iterations of layout.

`-timing_driven`

Set this switch to run timing driven layout.

`-c clockname`

Clock mode - the run with the minimum value of the maximum register to register delay for the specified clock is chosen as the best result. Default mode chooses the run with the highest frequency of the slowest clock as the best result.

Note: if '-compare_script compareScript' is specified, then that script will be used to determine the best run.

`-compare_script compareScript`

Sets the script that is used to determine whether a run is better than previous iterations. If only the script name is given (no path), then the script is assumed to be $ACTDIR/scripts/<compareScript>, otherwise the full path of the script must be given. The default is iterate_reg2reg_compare.tcl if the "-c clockname" option is used, and iterate_minfreq_compare.tcl otherwise.

`-save_all`

Saves all intermediate designs in ${adbFilename}_${iteration}.adb. The best result is also stored in $adbFilename as well. The default behavior does not save all results.

`-stop_on_success`

Only perform iterations until the design is successful. When the iterate_violations_compare.tcl isLayoutBetter method is used, Success is defined as there being no negative slack (no timing violations). When the other isLayoutBetter methods are used there is no effect (all iterations are attempted).

`-continue_from_last_seed`

Use the def variable MULTIPASS_LAYOUT_SEED_INDEX+1 as the starting seed.

`-place` *`value`*

Default is "on".The following table shows the acceptable values for this argument:

Value	Description
on	Implements place
off	Does not implement place

`-incremental_place` *`value`*

Sets incremental place. The following table shows the acceptable values for this argument:

Value	Description
on	Activates incremental place.
off	Deactivates incremental place

`-incremental_route` *`value`*

Default is "off".

The following table shows the acceptable values for this argument:

Value	Description
on	Activates incremental route
off	Deactivates incremental route

`[-effort_level n]`

This is an advanced option. N should be an integer. This option is available only forProASIC^{PLUS}.

`[-timing_weight n]`

This is an advanced option. N should be an integer. This option is available only forProASIC^{PLUS}.

Return: A non-zero value will be returned on error.

Supported Families

All

Notes

This is not a Tcl command. It is a shell script that can be run from the command line.

Exceptions

`N/A`

Example

N/A

See Also

N/A

get_defvar

The get_defvar command provides access to the internal variables within Designer and returns it's value.

```
get_defvar variable
```

Arguments

The *variable* is the Designer internal variable.

Supported Families

All

Notes

This command also prints the value of the Designer variable on the log window.

Exceptions

None

Example

Example 1: Prints the design name on the log window.

```
   get_defvar "DESIGN"
set variableToGet "DESIGN"
set valueOfVariable [get_defvar $variableToGet]
puts "The value is $valueOfVariable"
```

See Also

set_defvar

get_design_fileName

The get_design_filename command can be used to retrieve the full qualified path of the design file.

```
get_design_filename
```

Arguments

None

Supported Families

All

Notes

The result will be an empty string if the design has not been saved to disk.

This command is equivalent to the command “get_design_info DESIGN_PATH.” This command predates get_design_info and is supported for backward-compatibility.

Exceptions

The command will return an error if a design is not loaded.

The command will return an error if arguments are passed.

Example

```
if { [ is_design_loaded ] } {
  set design_location [ get_design_filename ]
  if {$design_location != "" } {
    puts “Design is at $design_location.”
  } else {
    puts “Design has not been saved to a file on disk.”
  }
} else {
  puts "No design is loaded."
}
```

See Also

get_design_info

is_design_loaded

is_design_modified

is_design_state_complete

get_design_info

The get_design_info command can be used to retrieve some basic details of your design.

```
get_design_info value
```

Arguments

The argument must be one of the valid string values. The possible values are summarized in the table below:

Value	Description
name	Design name. The result is set to the design name string.
family	Silicon family. The result is set to the family name.
design_path	Fully qualified path of the design file. The result is set to the location of the .adb file. If a design has not been saved to disk, the result will be an empty string. This command replaces the command

	get_design_filename.
design_folder	Directory (folder) portion of the design_path.
design_file	Filename portion of the design_path.
cwdir	Current working directory. The result is set to the location of the current working directory
die	Die name. The result is set to the name of the selected die for the design. If no die is selected, this is an empty string.
Package	Package. The result is set to the name of the selected package for the design. If no package is selected, this is an empty string.
Speed	Speed grade. The result is set to the speed grade for the design. If no speed grade is selected, this is an empty string.

Supported Family

All

Notes

The result value of the command will be a string value.

Exceptions

The command will return an error if a design is not loaded.

The command will return an error if more than one argument is passed.

The command will return an error if the argument is not one of the valid values.

Example

The following example uses get_design_info to display the various values to the screen.

```
if { [ is_design_loaded ] } {
  puts "Design is loaded."
  set bDesignLoaded 1
} else {
  puts "No design is loaded."
  set bDesignLoaded 0
}
if { $bDesignLoaded != 0 } {
  set var [ get_design_info NAME ]
  puts "  DESIGN NAME:\t$var"
  set var [ get_design_info FAMILY ]
  puts "  FAMILY:\t$var"
  set var [ get_design_info DESIGN_PATH ]
  puts "  DESIGN PATH:\t$var"
  set var [ get_design_info DESIGN_FILE ]
  puts "  DESIGN FILE:\t$var"
  set var [ get_design_info DESIGN_FOLDER ]
  puts "  DESIGN FOLDER:\t$var"
```

```
  set var [ get_design_info CWDIR ]
  puts "  WORKING DIRECTORY:  $var"
  set var [ get_design_info DIE ]
  puts "  DIE:\t$var"
  set var [ get_design_info PACKAGE ]
  puts "  PACKAGE:\t'$var'"
  set var [ get_design_info SPEED ]
  puts "  SPEED GRADE:\t$var"
  if { [ is_design_modified ] } {
    puts "The design is modified."
  } else {
    puts "The design is unchanged"
  }
}
puts "get_design.tcl done"
```

See Also

get_design_filename

is_design_loaded

is_design_modified

is_design_state_complete

import_aux

Imports the specified auxiliary file into the design. Equivalent to executing the Import Auxiliary Files command from the File menu.

```
import_aux -format file_type [-merge_timing] filename
```

Arguments

`-format file_type`

Specifies the file format of the file to import. You can import one of the following types of files: pdc, sdc, pin, dcf, saif, vcd, or crt.

`-merge_timing value`

Specifies whether to preserve all existing timing constraints when you import an SDC file. Same as selecting or unselecting the "Keep existing timing constraints" check box in the Import Files dialog box. The following table shows the acceptable values for the this option:

Value	Description
yes/true	Designer merges the timing constraints from the imported SDC file with the existing constraints saved in the constraint database.
no/false	The existing timing constraints are replaced by the constraints in the newly imported SDC file.

`filename`

Specifies the name of the auxiliary file to import.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, SX/SX-A

Notes

Auxiliary files are not audited and are handled as one-time data-entry or data-change events, similar to entering data using one of the interactive editors (for example, PinEditor or Timer).

Some timing constraints (such as multi_cycle) are not supported in the Timer GUI and must be implemented by importing the SDC file. If you import the SDC file as an auxiliary file, you do not have to re-compile your design. However, auditing is disabled when you import auxiliary files, and Designer cannot detect the changes to your SDC file(s) if you import them as auxiliary files.

Exceptions

For auxiliary SDC constraints in eX, SX-A, ProASIC^{PLUS}, Axcelerator and ProASIC3/E:

```
import_aux -format sdc -merge_timing yes/no ...
import_aux -format sdc -merge_all yes/no ...
import_aux -format sdc -merge yes/no ...
```

All NGT families support SDC. The import_aux command does not support any merge options for physical constraints.

For these families, in import_source, the -merge_all and -merge options are mapped only to -merge_timing. When recorded, the import-aux SDC always writes out -merge_timing.

Examples

```
import_aux -format sdc -merge_timing yes mydesign.sdc
import_aux -format pdc {C:/designs/mydesign.pdc}
```

See Also

import_source

Importing auxiliary files

Importing source files

Importing files

Tcl documentation conventions

import_Source

Imports the specified source file into the design. Equivalent to executing the Import Source File command from the File menu.

```
import_source [-merge_timing value][-merge_physical value][-merge_all value][-format
file_type [-abort_on_error][-top_entity][-flavor value]filename]+
```

Arguments

`-merge_timing value`

Specifies whether to preserve all existing timing constraints when you import an SDC file. Same as selecting or unselecting the "Keep existing timing constraints" check box in the Import Files dialog box. The following table shows the acceptable values for this option:

Value	Description
yes	Designer merges the timing constraints from the imported SDC file with the existing constraints saved in the constraint database. If there is a conflict, the new constraint has priority over the existing constraint.
no	All existing timing constraints are replaced by the constraints in the newly imported SDC file.

`-merge_physical` *value*

Specifies whether to preserve all existing physical constraints when you import a PDC file. Same as selecting or unselecting the "Keep existing physical constraints" check box in the Import Files dialog box. The following table shows the acceptable values for this option:

Value	Description
yes	Designer preserves all existing physical constraints that you have entered either using one of the MVN tools (ChipPlanner, PinEditor, or the I/O Attribute Editor) or a previous GCF or PDC file. The software resolves any conflicts between new and existing physical constraints and displays the appropriate message.
no	All existing physical constraints are replaced by the constraints in the newly imported PDC file.

`-merge_all` *value*

Specifies whether to preserve all existing physical and timing constraints when you import an SDC and/or a PDC file. Same as selecting or unselecting the "Keep existing physical constraints" and "Keep existing timing constraints" check boxes in the Import Files dialog box. The following table shows the acceptable values for this option:

Value	Description
yes	Designer preserves all existing physical constraints that you have entered either using one of the MVN tools (ChipPlanner, PinEditor, or the I/O Attribute Editor) or a previous GCF or PDC file. The software resolves any conflicts between new and existing physical constraints and displays an appropriate message. Any existing timing constraints from your ADB are merged with the new information from your imported files. New constraints override any existing timing constraints whenever there is a conflict
no	All the physical constraints in the newly imported GCF or PDC files are used. All pre-existing physical constraints are lost. Existing timing constraints from the ADB are replaced by the new timing constraints from your imported file.

`-format file_type`

Specifies the file format of the file to import. You can import one of the following types of files: adl, edif, verilog, vhdl, gcf, pdc, sdc, or crt.

`-abort_on_error`

Aborts a PDC file if it encounters an error during import.

`-top_entity`

Specifies the top entity to a VHDL file.

`-flavor`

Specifies the type of netlist. It can be edif, viewlogic, or mgc.

`filename`

Specifies the name of the source file to import.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, SX/SX-A

Notes

None.

Exceptions

Your script -merge options vary according to family as shown below:

For source SDC (no PDC/GCF) in eX and SX-A:

```
import_source -merge_timing yes/no ...
import_source -merge yes/no  ...
import_source -merge_all yes/no  ...
```

```
The eX and SX-A families do not support merging of physical constraints. For these families, in
import_source, the -merge_all and -merge options are mapped just to -merge_timing.
```

For constraints in ProASIC^{PLUS}:

```
import_source -merge_physical yes/no ...
import_source -merge_all yes/no ...
import_source -merge yes/no ...
```

ProASIC^{PLUS} andProASIC support GCF, but the -merge option in the import_source only affects GCF in terms of the physical constraints.

ProASIC does not support PreCompile import of the SDC files. The -merge_timing option has no effect on this import_source for ProASIC. The -merge_all and -merge options map to -merge_physical for ProASIC in import_source.

For ProASIC^{PLUS}, Axcelerator and ProASIC3/E

```
import_source -merge_physical yes/no -merge_timing yes/no ...
import_source -merge_all yes/no  ...
import_source -merge yes/no  ...
```

The -merge_all and -merge options map to both -merge_physical and -merge_timing options for these families.

Examples

Consider the following sample scripts:

```
import_source  \
 -merge_physical "no" \
 -merge_timing "yes"
```

```
  -format "EDIF" -edif_flavor "GENERIC" \
 {.\designs\mydesign.edn} \
 -format "sdc" \
 {.\designs\mydesign.sdc} \
 -format "pdc" -abort_on_error "no" \
 {.\designs\mydesign.pdc} \
import_source \
 -merge_physical "no" \
 -merge_timing "yes" \
 -format "verilog" \
 {mydesign.v}
import_source \
  -merge_physical "no" \
  -merge_timing "no"
  -format "vhdl" -top_entity "aclass" \
   {C:/mynetlist.vhd}
import_source \
  -merge_physical "no" \
  -merge_timing "no"
  -format "adl" {mydesign.adl
```

See Also

import_source

Importing auxiliary files

Importing source files

Importing files

Tcl documentation conventions

is_design_loaded

The is_design_loaded returns a Boolean value (0 for false, 1 for true) indicating if a design is loaded in the Designer software. True is returned if a design is currently loaded.

```
is_design_loaded
```

Arguments

None

Supported Family

All

Notes

Some Tcl commands are valid only if a design is currently loaded in Designer. Use the 'is_design_loaded' command to prevent runtime errors by checking for this before invoking the commands.

Exceptions

The command will return an error if arguments are passed.

Example

The following code will determine if a design has been loaded.

```
set bDesignLoaded [ is_design_loaded ]
if { $bDesignLoaded == 0 } {
  puts "No design is loaded."
}
```

See Also

get_design_filename

get_design_info

is_design_modified

is_design_state_complete

is_design_modified

The is_design_modified returns a Boolean value (0 for false, 1 for true) indicating if a design has been modified in the Designer software. True is returned if a design has been modified.

```
is_design_modified
```

Arguments

None

Supported Family

Family: All

Notes

Some Tcl commands are valid only if a design has been modified in Designer. Use the is_design_modified command to prevent runtime errors by checking for this before invoking the commands.

Exceptions

The command will return an error if arguments are passed.

Example

The following code will determine if a design has been modified.

```
set bDesignModified [ is_design_modified ]
if { $bDesignModified == 0 } {
```

```
  puts "Design has not been modified."
}
```

See Also

get_design_filename

get_design_info

is_design_loaded

is_design_state_complete

is_design_state_complete

The is_design_state_complete command returns a Boolean value (0 for false, 1 for true) indicating if a specific design state is valid. True is returned if the specified design state is valid.

```
is_design_state_complete value
```

Arguments

The single argument must be one of the valid string values. The possible values are summarized in the table below:

Value	Description
SETUP_DESIGN	The design is loaded and the family has been specified for the design
DEVICE_SELECTION	The design has completed device selection (die and package). This corresponds to having successfully called the set_device command to set the die and package
NETLIST_IMPORT	The design has imported a netlist
COMPILE	The design has completed the compile command
LAYOUT	The design has completed the layout command
BACKANNOTATE	The design has exported a post-layout timing file (e.g. SDF)
PROGRAMMING_FILES	The design has exported a programming file (e.g. AFM)

Supported Family

All

Notes

Certain commands can only be used after Compile or Layout has been completed.

The is_design_state_complete command allows a script to check the design state before calling one of these state-limited commands.

Exceptions

The command will return an error if a design is not loaded.

The command will return an error if more than one argument is passed.

The command will return an error if the argument is not one of the valid values.

Example

The following code runs layout, but checks that the design state for layout is complete before calling backannotate.

```
layout -timing_driven
set bLayoutDone [ is_design_state_complete LAYOUT ]
if { $bLayoutDone  != 0 } {
  backannotate -name {mydesign_ba}  -format "SDF" -language "verilog"
  }
}
```

See Also

compile

get_design_filename

get_design_info

is_design_loaded

is_design_modified

layout

set_design

set_device

layout

Sets the layout mode and placer value; this command is identical to the layout command in the Designer GUI.

```
layout -value [-run_placer value]
```

Arguments

`layout -value`

Sets layout mode to be timing driven or standard. If not given, default is standard or the mode used in the previous layout command. Values are summarized in the table below:

Value	Description
timing_driven	Sets timing driven layout mode
standard	Sets standard layout mode

`-run_placer|-placer|-place value`

Invokes or skips placement. If set to OFF, previous placement will be used. Possible values are summarized in the table below. Note that all three of the commands (-run_placer, -placer, and -place) execute the same action.

Value	Description
ON	Invokes placement
OFF	Skips placement

Supported Families

All

Notes

Please refer to the Advanced Layout Options for your device family for more information.

Exceptions

None

Example

The following code runs layout and specifies the timing-driven option.

```
layout -timing_driven
```

See Also

layout (advanced options for SX family)

layout (advanced options forProASIC)

layout (advanced options for Axcelerator)

layout (Advanced Options for the SX family)

This is equivalent to executing commands within the Advanced Layout Options dialog box for the SX, SX-A, and eX families.

```
layout [-timing_driven] [-incremental value] [-extended_run value] [-effort_level
value] [-timing_weight value]
```

Arguments

`-timing_driven`

Sets layout mode to timing driven. If not given, default is standard or the mode used in the previous layout command.

`-incremental value`

Invokes or skips incremental mode. Possible values are summarized in the table below.

Value	Description
ON	Invokes incremental mode, and sets the previous routing information as the initial starting point
OFF	Skips incremental mode, discards previous information

`-extended_run value`

Invokes or skips extended_run mode. Possible values are summarized in the table below.

Value	Description
ON	Invokes extended run mode
OFF	Skips extended run mode

`-effort_level value`

Sets the effort level; number may range from 25 to 500.

`-timing_weight value`

Sets the timing weight value; number may range from 10 to 150.

Supported Families

`SX,SX-A, eX`

Notes

None

Exceptions

None

Example

```
layout [-timing_driven] [-incremental OFF] [-extended_run OFF] [-effort_level 50] [-timing_weight 100]
```

See Also

layout

layout (advanced options forProASIC)

layout (advanced options for Axcelerator)

layout (Advanced Options for ProASIC and ProASICPLUS)

This is equivalent to executing commands within the Advanced Layout Options dialog box. These commands perform the layout (placement and/or routing) for the design

```
layout [-value]
[-run_placer value]
[-place_incremental value]
[-placer_seed value]
[-show_placer_seed]
[-timing_weight value]
[-run_router value]
[-route_incremental value]
```

Arguments

`layout -value`

Sets layout mode to be timing driven or standard. If not given, default is timing-driven. Values are summarized in the table below:

Value	Description
timing_driven	Sets timing driven layout mode
standard	Sets standard layout mode

`-run_placer|-placer|-place` *`value`*

`Invokes or skips placement. If set to OFF, previous placement will be used. Possible values are summarized in the table below. Note that all three of the commands (-run_placer, -placer, and -place) execute the same action.`

Value	Description
ON	Invokes placement
OFF	Skips placement

`-place_incremental|-incremental` *`value`*

Sets your incremental placement options for layout. Values are summarized in the table below. Note that the -place_incremental and -incremental commands are interchangeable.

Value	Description
ON	Sets the previous placement as the initial starting point
OFF	(default) Discards previous placement
FIX	Sets the previously placed macros' locations as "FIXED" and continues to place the remaining ones

`-placer_seed` *`value`*

An integer value that you can set to change the initial random seed number for the placement.

`-show_placer_seed`

Causes Layout to display the initial random seed number used for the placement.

`-timing_weight` *`value`*

The timing weight for the placement. Values range from 1 to 4 in whole integers. The lower numbers give less weight to timing in the placement. Timing weight 4 is the same as timing driven placement using the option –timing_driven.

`-run_router|-router|-route` *`value`*

Invokes (if placement is successful) or skips routing. Values are described below.

Value	Description
ON	Invokes routing if placement is successful
OFF	Skips routing

`-route_incremental` *`value`*

Invokes or skips incremental mode. Possible values are summarized in the table below.

Value	Description
ON	Invokes incremental mode, and sets the previous routing information as the initial starting point
OFF	Skips incremental mode, discards previous information

Supported Families

ProASIC,ProASIC[PLUS]

Notes

Please refer to theProASIC and ProASICPLUS Layout Options for more information.

Exceptions

None

Example

```
layout
layout -standard
layout -timing_weight 3
layout -place_incremental FIX -route_incremental ON
layout -placer_seed 120
```

See Also

layout (for SX, SX-A, and eX)

layout (for Axcelerator)

layout (Advanced Options for Axcelerator)

This is equivalent to executing commands within the Advanced Layout Options dialog box.

```
layout [-value]
[-run_placer value]
[-place_incremental value]
[-placer_seed value]
[-effort_level value]
[-run_router value]
[-route_incremental value]
```

Arguments

`layout -value`

Sets layout mode to be timing driven or standard. If not given, default is standard or the mode used in the previous layout command. Values are summarized in the table below:

Value	Description
timing_driven	Sets timing driven layout mode

standard	Sets standard layout mode

`-run_placer|-placer|-place` *value*

`Invokes or skips placement. If set to OFF, previous placement will be used. Possible values are summarized in the table below. Note that all three of the commands (-run_placer, -placer, and -place) are interchangeable.`

Value	Description
ON	Invokes placement
OFF	Skips placement

`-place_incremental|-incremental` *value*

Sets your incremental placement options for layout. Values are summarized in the table below. Note that the -place_incremental and -incremental commands are interchangeable.

Value	Description
ON	Sets the previous placement as the initial starting point
OFF	(default) Discards previous placement
FIX	Sets the previously placed macros' locations as "FIXED" and continues to place the remaining ones

`-placer_seed` *value*

An integer value that you can set to change the initial random seed number for the placement.

`-effort_level` *value*

The effort level for the placement. Values range from 1 to 5 in whole integers. The lower numbers tend to give a result more quickly, but higher levels yield better performance in timing and routability.

`-run_router|-router|-route` *value*

Invokes (if placement is successful) or skips routing. Values are described below.

Value	Description
ON	Invokes routing if placement is successful
OFF	Skips routing

`-route_incremental` *value*

Invokes or skips incremental mode. Possible values are described in the table below.

Value	Description
ON	Invokes incremental mode, and sets the previous routing information as the initial starting point
OFF	Skips incremental mode, discards previous information

Supported Families

Axcelerator

Notes

Please refer to the Axcelerator Layout options for more information.

Exceptions

N/A

Example

```
layout
layout -timing_driven -effort_level 4
layout -place_incremental FIX -route_incremental ON
layout -placer_seed 120
```

See Also

layout (for SX)

layout (forProASIC)

new_design

The new_design command creates a new design.

```
new_design -name design_name -family family_name -path pathname
```

Arguments

-name design_name

The name of the design. This is used as the base name for most of the files generated from Designer.

-family family_name

The Actel device family for which the design is being targeted.

-path path_name

The physical path of the directory in which the design files will be created.

Supported Families

Family: All

Notes

You need all the 3 arguments for this command. This command will setup the Designer software for importing design source files.

Exceptions

None

Example

Example 1: Creates a new ACT3 design with the name “test” in the current folder.

```
new_design -name "test" -family "ACT3" -path {.}
```

Example 2: These set of commands create a new design through variable substitution.

```
set desName "test
set famName "ACT3"
set path {d:/examples/test}
new_design -name $desName -family $famName -path $path
```

Example 3: Design creation and catch failures

```
if  { [catch { new_design -name $desName -family $famName -path $path }] {
              Puts "Failed to create a new design"
              # Handle Failure
} else {
              puts "New design creation successful"
              # Proceed to Import source files
}
```

See Also

close_design

open_design,

save_design

set_design

open_design

The open_design command opens an existing design into the Designer software.

```
open_design file_name
```

Arguments

`file_name`

is the complete adb file path. The complete path is not provided then the directory is assumed to be the current working directory.

Supported Families

All

Notes

All previously open designs must be closed before opening a new design.

Exceptions

None

Example

Example 1: Opens an existing design from the file “test.adb” in the current folder.

```
open_design {test.adb}
```

Example 2: Design creation and catch failures.

```
set designFile {d:/test/my_design.adb}
if  { [catch { open_design $designFile }] {
              Puts “Failed to open design”
              # Handle Failure
} else {
              puts “Design opened successfully”
              # Proceed to further processing
}
```

See Also

close_design

new_design

save_design

pin_assign

Use to either assign the named pin to the specified port or assign attributes to the specified port. This command has two syntax formats. The one you use depends on what you are trying to do. The first syntax format assigns the named pin to the specified port. The second one assigns attributes to the specified port.

```
pin_assign [-nofix] -port portname -pin pin_number
pin_assign  -port portname  [-iostd value] [-iothresh value][-outload value] [-slew value][-res_pull value]
```

Arguments

`-nofix`

Unlocks the pin assignment (by default, assignments are locked).

`-port portname`

Specifies the name of the port to which the pin is assigned.

`-pin pin_number`

Specifies the alphanumeric number of the pin to assign.

`-iostd value`

Sets the I/O standard for this pin. Choosing a standard allows the software to set other attributes such as the slew rate and output loading. If the voltage standard used with the I/O is not compatible with other I/Os in the I/O bank, then assigning an I/O standard to a port will invalidate its location and automatically unassign the I/O. The following table shows the acceptable values for the supported devices:

I/O Standard	ProASIC PLUS	Axcelerator	ProASIC	RTSX-S	SX/SX-A	eX
CMOS				X		
CUSTOM				X	X	X
GTLP25	X	X				
GTLP33	X					
GTL33	X	X				
GTL25	X	X				
HSTL1	X	X				
HSTLII	X					
LVCMOS33	X		X			
LVCMOS25	X	X				
LVCMOS25_50	X		X			
LVCMOS18	X	X	X			
LVCMOS15	X	X	X			
LVDS		X				
LVPECL	X*	X				
LVTTL	X	X	X	X	X	
TTL	X	X	X	X	X	
PCI	X	X	X	X	X	X
PCIX	X	X	X			
SSTL2I	X	X				
SSTL2II	X	X				
SSTL3I	X	X				
SSTL3II	X	X				

*Supported only on dedicated LVPECL I/Os.

`-iothresh` *value*

Sets the compatible threshold level for inputs and outputs. This attribute is for SX-A and RTSX-S families only. The default I/O threshold is based upon the I/O standard. You can set the I/O Threshold independently of the I/O specification in the PinEditor tool by selecting **CUSTOM** in the I/O Standard cell. The following table shows the acceptable values for the supported devices (SX-A, RTSX-S, and eX):

Value	Description
CMOS	RTSX-S devices only. An advanced integrated circuit (IC) manufacturing process technology for logic and memory,

	characterized by high integration, low cost, low power, and high performance. CMOS logic uses a combination of p-type and n-type metal-oxide-semiconductor field effect transistors (MOSFETs) to implement logic gates and other digital circuits found in computers, telecommunications, and signal processing equipment.
LVTTL	(Low-Voltage TTL) A general purpose standard (EIA/JESDSA) for 3.3V applications. It uses an LVTTL input buffer and a push-pull output buffer.
PCI	A computer bus for attaching peripheral devices to a computer motherboard in a local bus. This standard supports both 33 MHz and 66 MHz PCI bus applications. It uses an LVTTL input buffer and a push-pull output buffer. With the aid of an external resistor, this I/O standard can be 5V-compliant for most families, excluding ProASIC3/E families.

`-slew` *`value`*

Sets the output slew rate. Slew control affects only the falling edges. Rising edges are not affected. This attribute is only available for LVTTL, PCI, and PCI outputs. For LVTTL, it can either be high or low. For PCI and PCIX, it can only be set to high. The following table shows the acceptable values for the supported devices (ProASIC PLUS, Axcelerator, ProASIC, SX-A, RTSX-S, and eX):

Value	Description
high	Sets the I/O slew to high
low	Sets the I/O slew to low

`-res_pull` *`value`*

Allows you to include a weak resistor for either pull-up or pull-down of the input buffer. The following table shows the acceptable values for the supported devices (ProASIC PLUS, Axcelerator, ProASIC, SX-A/RTSX-S, and eX):

Value	Description
up	Includes a weak resistor for pull-up of the input buffer
down	Includes a weak resistor for pull-down of the input buffer
none	Does not include a weak resistor

`-out_load` *`value`*

Indicates the output-capacitance value based on the I/O standard selected. This option is not available in ACTgen. This attribute determines what Timer will use as the loading on the output pin and applies only to outputs. You can enter a capacitive load as an integral number of picofarads (*pF*). The default is 35*pF*. This attribute is available only for the following devices: ProASIC PLUS, Axcelerator, ProASIC, SX-A/RTSX-S, and eX.

Supported Families

ProASIC PLUS, Axcelerator,ProASIC, SX, SX-A, RTSX-S, and eX

Notes

You must use `pin_commit` after this command (see Examples below) to save the changes to your design.

The -iothresh argument is also referred to as "Loading" in some families.

You can use this command for designs created with PinEditor in MVN and PinEditor Standalone.

This command does not support MX and SX devices.

The following table provides a summary of all supported I/O features for SX-A, RTSX-S, and eX devices:

I/O Standard	Device			Output Slew-rate Control	Power-Up State Control	Hot-Swap	Loading (pf)
	SX-A	eX	RT54SX-S				
2.5V LVCMOS	X	X		X	X	Yes	35
3.3V LVTTL	X	X	X	X	X	Yes	35
5V TTL	X	X	X	X	X	Yes	35
3.3V PCI	X		X	High	X	No	10
5V PCI	X		X	High	X	Yes	50
5V CMOS			X	X	X	Yes	35

Exceptions

None

Examples

```
pin_assign -port usw0 -pin A2
```

```
pin_assign -port usw0 -iostd LVTTL -slew Low -res_pull Down
```

See Also

pin_commit

pin_fix

pin_unassign

Tcl documentation conventions

pin_commit

Saves the pin assignments to the design (.adb) file.

```
pin_commit
```

Arguments

None

Supported Families

ProASIC PLUS, Axcelerator,ProASIC, SX, SX-A, RTSX-S, MX, and eX

Notes

To save your changes, you must add the pin_commit command to the end of the script.

You can use this command for designs created with PinEditor in MVN and PinEditor Standalone.

Exceptions

None

Examples

```
pin_commit
```

See Also

pin_fix

pin_unfix

pin_assign

pin_unassign

Tcl documentation conventions

pin_fix

Locks the pin assignment for the specified port, so the pins cannot be moved during place-and-route.

```
pin_fix -port portname
```

Arguments

`-port portname`

Specifies the name of the port to which the pin must be locked at its assigned location.

Supported Families

ProASIC PLUS, Axcelerator,ProASIC, SX, SX-A, RTSX-S, MX, and eX

Notes

You cannot move locked pins during place-and-route.

You can assign only one pin to a port.

You must use `pin_commit` after this command (see Examples below) to save the changes to your design.

You can use this command for designs created with PinEditor in MVN and PinEditor Standalone.

Exceptions

None

Examples

```
pin_fix -port clk
pin_commit
```

See Also

pin_commit

pin_unfix

pin_assign

pin_unassign

Tcl documentation conventions

pin_fix_all

Locks all the assigned pins on the device so they cannot be moved during place-and-route.

```
pin_fix_all
```

Arguments

None

Supported Families

ProASIC PLUS, Axcelerator,ProASIC, SX, SX-A, RTSX-S, MX, and eX

Notes

You cannot move locked pins during place-and-route.

You must use `pin_commit` after this command (see Examples below) to save the changes to your design.

You can use this command for designs created with PinEditor in MVN and PinEditor Standalone.

Exceptions

None

Example

```
pin_fix_all
pin_commit
```

See Also

pin_commit

pin_fix

pin_unfix

pin_assign

pin_unassign

Tcl documentation conventions

pin_unassign

Unassigns the pin from the specified port. The unassigned pin location is then available for other ports. (Only one pin can be assigned to a port.)

```
pin_unassign -port portname
```

Arguments

`-port portname`

Specifies the name of the port for which the pin must be unassigned.

Supported Families

ProASIC PLUS, Axcelerator,ProASIC, SX, SX-A, RTSX-S, MX, and eX

Notes

You must use `pin_commit` after this command (see Examples below) to save the changes to your design.

You can use this command for designs created with PinEditor in MVN and PinEditor Standalone.

Exceptions

None

Examples

```
pin_unassign -port "clk"
pin_commit
```

See Also

pin_commit

pin_fix

pin_fix_all

pin_unfix

pin_assign

pin_unassign

Tcl documentation conventions

pin_unassign_all

Unassigns all the pins from all the ports so that all pin locations are available for assignment.

```
pin_unassign_all
```

Arguments

None

Supported Families

ProASIC PLUS, Axcelerator, ProASIC, SX, SX-A, RTSX-S, MX, and eX

Notes

You must use `pin_commit` after this command (see Example below) to save the changes to your design.

You can use this command for designs created with PinEditor in MVN and PinEditor Standalone.

Exceptions

None

Examples

```
pin_unassign_all
pin_commit
```

See Also

pin_commit

pin_fix

pin_unfix

pin_assign

pin_unassign

Tcl documentation conventions

pin_unfix

Unlocks the pins assigned to the specified port, so the pins can be moved during place-and-route.

```
pin_unfix -port portname
```

Arguments

`-port portname`

Specifies the name of the port containing pins to unlock.

Supported Families

ProASIC PLUS, Axcelerator,ProASIC, SX, SX-A, RTSX-S, MX, and eX

Notes

You can move locked pins during place-and-route.

You must use the `pin_commit command` after this command (see Examples below) to save the changes to your design.

You can use this command for designs created with PinEditor in MVN and PinEditor Standalone.

Exceptions

None

Examples

```
pin_unfix -port rst
pin_commit
```

See Also

pin_commit

pin_fix

pin_assign

pin_unassign

Tcl documentation conventions

report

The report command provides you with frequently-used information in a convenient format.

You can generate the following types of reports using this command:

- Status Report
- Timing Report
- Timing Violations Report
- Flip-flop Report
- Power Report
- Pin Report
- I/O Bank Report

save_design

The save_design command saves the current design in Designer to a file.

```
save_design filename
```

Arguments

The design is written to a file denoted by the variable *filename* as an ADB file.

Supported Families

All

Notes

If filename is not a complete path name, the ADB file is written into the current working directory.

Exceptions

None

Example

Example 1: Saves the design to a file “test.adb” in the current folder.

```
save_design {test.adb}
```

Example 2: Save design and check if it saved successfully.

```
set designFile {d:/test/my_design.adb}
if  { [catch { save_design $designFile }] {
              Puts “Failed to save design”
              # Handle Failure
} else {
              puts “Design saved successfully”
```

```
                # Proceed to make further changes
}
```

See Also

close_design,

new_design

open_design,

set_design

This set_design command specifies the design name, family and path in which Designer will process the design. This step is absolutely required before importing the source files.

```
set_design -name design_name -family family_name -path path_name
```

Arguments

`-name design_name`

The name of the design. This is used as the base name for most of the files generated from Designer.

`-family family_name`

The Actel device family for which the design is being targeted.

`-path path_name`

The physical path of the directory in which the design files will be created.

Supported Families

All

Notes

You need all 3 arguments for this command to setup your design.

Example

Example 1: Sets up the design and checks if there are any errors

```
set_design -name "test" -family "Axcelerator" -path {.}
set desName “test
set famName “ACT3”
set path {d:/examples/test}
if  { [catch { set_design -name $desName -family $famName -path $path }] {
                Puts “Failed setup design”
                # Handle Failure
} else {
                puts “Design setup successful”
                # Proceed to Import source files
}
```

See Also

new_design,

set_device

set_device

The set_device command specifies the type of device and its parameters.

Syntax

```
set_device -family family_name -die die_name -package package_name -speed speed_grade
-voltage voltage -voltrange volt_range -temprange temp_range -iostd default_io_std -
pci value -jtag value -probe value -trst value
```

Arguments

`-family family_name`

Specifies the name of the FPGA device family.

`-die die_name`

Specifies the part name.

`-package package_name`

Specifies the selected package for the device.

`-speed speed_grade`

Specifies the speed grade of the part.

`-voltage voltage`

Specifies the core voltage of the device.

`-voltrange volt_range`

Specifies the voltage range to be applied for the device. It is generally MIL, COM and IND denoting Military, Commercial and Industrial respectively.

`-temprange temp_range`

Specifies the voltage range to be applied for the device. It is generally MIL, COM and IND denoting Military, Commercial and Industrial respectively.

`-iostd default_io_std`

Specifies the default I/O Standard of the part

`-pci value`

Specified if the device needs to configure the IO for PCI specification. Values are summarized in the table below.

Value	Description
yes	Device is configured for PCI specification
no	Device is not configured for PCI specification

`-jtag value`

Specifies if pins need to be reserved for JTAG. Values are summarized in the table below.

Value	Description
yes	Pins are reserved for JTAG
no	Pins are not reserved for JTAG

`-probe value`

Specifies if the pins need to be preserved for probing. Values are summarized in the table below.

Value	Description
yes	Pins are preserved for probing
no	Pins not preserved for probing

`-trst value`

Specifies if the pins need to be reserved for JTAG test reset. Values are summarized in the table below.

Value	Description
yes	Pins are preserved for JTAG test reset
no	Pins are not preserved for JTAG test reset

Supported Families

All

Notes

At least one option must be specified for this command. Some of the options may not apply for certain families that do not support the features.

Example

Example 1: Setting up a PA design.

```
set_device -die "APA075" -package "208 PQFP" -speed "STD" -voltage "2.5" \
-jtag "yes" -trst "yes" -temprange "COM" -voltrange "COM"
```

See Also

new_design,

set_design

set_defvar

The set_defvar command sets an internal variable in the Designer system.

```
set_defvar variable value
```

Arguments

Variable must be a valid Designer internal variable and could be accompanied by an optional value. If the *value* is provided, the *variable* is set the *value*. If the *value* is null the *variable* is reset.

Supported Families

All

Notes

Must have at least one argument.

Exceptions

None

Example

Example 1:

```
set_defvar "FORMAT" "VHDL"
```

Sets the FORMAT internal variable to VHDL.

Example 2:

```
set variableToSet "DESIGN"
set valueOfVariable "VHDL"
set_defvar $variableToSet $valueOfVariable
```

These commands set the FORMAT variable to VHDL, shows the use of variables for this command.

See Also

get_defvar

smartpower_add_pin_in_domain

Adds a pin into a clock or set domain.

```
smartpower_add_pin_in_domain -pin_name {pin_name} -pin_type {value} -domain_name
{domain_name} -domain_type {value}
```

Arguments

`-pin_name {pin_name}`

Specifies the name of the pin to add to the domain.

`-pin_type {value}`

Specifies the type of the pin to add. The following table shows the acceptable values for this argument:

Value	Description
clock	The pin to add is a clock pin
data	The pin to add is a data pin

`-domain_name {domain_name}`

Specifies the name of the domain in which to add the specified pin.

`-domain_type {value}`

Specifies the type of domain in which to add the specified pin. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

Supported Families

ProASIC3/E,ProASIC[PLUS], Axcelerator, and ProASIC

Notes

The *domain_name* must be a name of an existing domain.

The *pin_name* must be a name of a pin that exists in the design.

Exceptions

None

Examples

The following example adds a clock pin to an existing Clock domain:

```
smartpower_add_pin_in_domain -pin_name { XCMP3/U0/U1:Y } -pin_type {clock} -domain_name {clk1} -
domain_type {clock}
```

The following example adds a data pin to an existing Set domain:

```
smartpower_add_pin_in_domain -pin_name {XCMP3/U0/U1:Y}   -pin_type {data} -domain_name {myset} -
domain_type {set}
```

See Also

smartpower_remove_pin_of_domain

Tcl documentation conventions

smartpower_commit

Saves the changes made to the Designer database.

```
smartpower_commit
```

Arguments

None

Supported Families

ProASIC3/E,ProASIC[PLUS], Axcelerator, and ProASIC

Notes

None

Exceptions

None

Examples

```
smartpower_commit
```

See Also

smartpower_restore

Tcl documentation conventions

smartpower_create_domain

Creates a new clock or set domain.

```
smartpower_create_domain -domain_type {value} -domain_name {domain_name}
```

Arguments

`-domain_type {value}`

Specifies the type of domain to create. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

`-domain_name {domain_name}`

Specifies the name of the new domain.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, and ProASIC

Notes

The domain name cannot be the name of an existing domain.

The domain type must be either clock or set.

Exceptions

None

Examples

The following example creates a new clock domain named "clk2":

```
smartpower_create_domain -domain_type {clock} -domain_name {clk2}
```

The following example creates a new set domain named "myset":

```
smartpower_create_domain -domain_type {set} -domain_name {myset}
```

See Also

smartpower_remove_domain

Tcl documentation conventions

smartpower_remove_domain

Removes an existing clock or set domain.

```
smartpower_remove_domain -domain_type {value} -domain_name {domain_name}
```

Arguments

`-domain_type {value}`

Specifies the type of domain to remove. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

`-domain_name {domain_name}`

Specifies the name of the domain to remove.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, and ProASIC

Notes

The domain name must be the name of an existing domain.

The domain type must be either clock or set.

Exceptions

None

Examples

The following example removes the clock domain named "clk2":

```
smartpower_remove_domain -domain_type {clock} -domain_name {clk2}
```

The following example removes the set domain named "myset":

```
smartpower_remove_domain -domain_type {set} -domain_name {myset}
```

See Also

smartpower_create_domain

Tcl documentation conventions

smartpower_remove_pin_frequency

Removes the frequency associated with a specific pin. This pin will have a default frequency based on its domain.

```
smartpower_remove_pin_frequency -pin_name {pin_name}
```

Arguments

`-pin_name {pin_name}`

Specifies the name of the pin for which the frequency will be removed.

Supported Families

ProASIC3/E,ProASIC[PLUS], Axcelerator, and ProASIC

Notes

The *pin_name* must be the name of a pin that already exists in the design and already belongs to a domain.

Exceptions

None

Examples

The following example removes the frequency from the pin named "count8_clock":

```
smartpower_remove_pin_frequency -pin_name {count8_clock}
```

See Also

smartpower_set_pin_frequency

Tcl documentation conventions

smartpower_remove_pin_of_domain

Removes a clock pin or a data pin from a Clock or Set domain, respectively.

```
smartpower_remove_pin_of_domain -pin_name {pin_name} -pin_type {value} -domain_name
{domain_name} -domain_type {value}
```

Arguments

`-pin_name {pin_name}`

Specifies the name of the pin to remove from the domain.

`-pin_type {value}`

Specifies the type of the pin to remove. The following table shows the acceptable values for this argument:

Value	Description
clock	The pin to remove is a clock pin
data	The pinto remove is a data pin

`-domain_name {domain_name}`

Specifies the name of the domain from which to remove the pin.

`-domain_type {value}`

Specifies the type of domain from which the pin is being removed. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

Supported Families

ProASIC3/E,ProASIC[PLUS], Axcelerator, and ProASIC

Notes

The domain name must be the name of an existing domain.

The pin name must be the name of an existing pin.

Exceptions

None

Examples

The following example removes the clock pin named "XCMP3/UO/U1:Y" from the clock domain named "clockh":

```
smartpower_remove_pin_of_domain -pin_name {XCMP3/U0/U1:Y}
-pin_type {clock} -domain_name {clockh} -domain_type {clock}
```

The following example removes the data pin named "count2_en" from the set domain named "InputSet":

```
smartpower_remove_pin_of_domain -pin_name {count2_en} -pin_type
{data} -domain_name {InputSet} -domain_type {set}
```

See Also

smartpower_add_pin_in_domain

Tcl documentation conventions

smartpower_restore

Restores all power information previously committed in SmartPower.

```
smartpower_restore
```

Arguments

None

Supported Families

ProASIC3/E,ProASIC[PLUS], Axcelerator, and ProASIC

Notes

None

Exceptions

None

Examples

```
smartpower_restore
```

See Also

smartpower_commit

Tcl documentation conventions

smartpower_set_domain_frequency

Sets the frequency of a domain in megahertz (MHz).

```
smartpower_set_domain_frequency -domain_type {value} -domain_name {domain_name} -
clock_freq {value} -data_freq {value} -pin_freq {value}
```

Arguments

`-domain_type {value}`

Specifies the type of domain to set. The following table shows the acceptable values for this argument:

Value	Description
clock	The domain is a clock domain
set	The domain is a set domain

`-domain_name {domain_name}`

Specifies the name of the domain for which the frequency will be set.

`-clock_freq {value}`

Specifies the clock frequency in megahertz (MHz), which can be any positive decimal number. This argument is available only for a clock domain.

`-data_freq {value}`

Specifies the data frequency in megahertz (MHz), which can be any positive decimal number. This argument is available only for a clock domain.

`-pin_freq {value}`

Specifies the value of the pin frequency in megahertz (MHz), which can be any positive decimal number, which can be any positive decimal number. This argument is available only for a set domain.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, and ProASIC

Notes

The domain type must be either “clock” or “set.”

The domain name must be the name of an existing domain.

The clock frequency must be a positive decimal number. Specifying the unit as part of the frequency value is optional. You must enter a space between the frequency value and the unit. You set the clock frequency only for clock domains.

The data frequency must be a positive decimal number. Specifying the unit as part of the data frequency value is optional. You must enter a space between the data frequency value and the unit.

Exceptions

None

Examples

The following example sets the clock and data frequency of a clock domain:

```
smartpower_set_domain_frequency -domain_type {clock} -domain_name {clk1} -clock_freq {32} or {30 MHZ}
-data_freq {3} or {3 Mhz}
```

The following example sets the data frequency of a set domain:

```
smartpower_set_domain_frequency -domain_type {set} -domain_name {set1} -data_freq {10}
```

See Also

smartpower_create_domain

smartpower_remove_domain

Tcl documentation conventions

smartpower_set_pin_frequency

Sets the frequency of a pin in megahertz (MHz). If you do not use this command, each pin will have default frequency based on its domain.

```
smartpower_set_pin_frequency -pin_name {pin_name} -pin_freq {value}
```

Arguments

`-pin_name {pin_name}`

Specifies the name of the pin for which the frequency will be set.

`-pin_freq {value}`

Specifies the value of the frequency in MHz, which can be any positive decimal number.

Supported Families

ProASIC3/E,ProASIC PLUS, Axcelerator, and ProASIC

Notes

The *pin_name* must be the name of a pin that already exists in the design and already belongs to a domain.

When specifying the unit, a space must be between the frequency value and the unit.

Exceptions

None

Examples

This example sets the frequency of the pin named "count8_clock" to 100 MHz:

```
smartpower_set_pin_frequency -pin_name {count8_clock} -pin_freq {100}
```

See Also

smartpower_remove_pin_frequency

Tcl documentation conventions

timer_add_clock_exception

Adds an exception to or from a pin with respect to a specified clock.

```
timer_add_clock_exception -clock clock_name -pin pin_name -dir from_value to_value
```

Arguments

`-clock clock_name`

Specifies the name of the clock.

`-pin pin_name`

Specifies the exception on the pin in a synchronous network that should be excluded from the specified clock period.

`-dir {from_value to_value}`

Specifies direction. The *from_value* refers to paths starting at that particular pin and *to_value* refers to paths ending at that particular pin.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

None

Exceptions

None

Examples

The following example adds a clock exception from the pin reg_q_a_10_/U0:CLK with respect to the clock clk.

```
timer_add_clock_exception -clock {clk} -pin {reg_q_a_10_/U0:CLK} -dir {from}
```

The following example adds a clock exception to the pin reg_q_a_10_/U0:E with respect to the clock clk.

```
timer_add_clock_exception -clock {clk} -pin {reg_q_a_10_/U0:E} -dir {to}
```

See Also

timer_remove_clock_exception

Tcl documentation conventions

timer_add_pass

Adds a pin to the list of pins for which the path must be shown passing through, in the Timer.

```
timer_add_pass -pin pin_name
```

Arguments

`-pin pin_name`

Specifies the name of the pin to be included for displaying the timing path through it.

Supported Families

ProASIC3/E, ProASIC[PLUS], Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

Setting a pass on a module pin enables you to see a path through individual pins.

Exceptions

None

Examples

This example adds a pass through the pin named "reg_q_a_0_:CLK" in the design:

```
timer_add_pass -pin {reg_q_a_0_:CLK}
```

This example adds a pass through a clear pin named "reg_q_a_0_:CLR" in the design

```
timer_add_pass -pin {reg_q_a_0_:CLR}
```

See Also

timer_add_stop

Tcl documentation conventions

timer_add_stop

Adds the specified pin to the list of pins through which the paths will not be displayed in the Timer.

```
timer_add_stop -pin pin_name
```

Arguments

`-pin pin_name`

Specifies the name of the pin to be excluded from displaying the path.

Supported Families

ProASIC3/E, ProASIC[PLUS], Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

Without stop points, you see all the paths from pad to pad in the design. If you do not want to see the paths going through registers clock pins, you could specify these as stop points. The path going through those pins would not be displayed.

Exceptions

None

Examples

The following example adds a stop to the pin named "a<2>" in the design:

```
timer_add_stop -pin {a<2>}
```

The following example adds a stop to a clock and a clear pin named "reg_q_a_0_:CLR" in the design:

```
timer_add_stop -pin {reg_q_a_0_:CLK}
```

```
timer_add_stop -pin {reg_q_a_0_:CLR}
```

See Also

timer_add_pass

Tcl documentation conventions

timer_commit

Saves the changes made to constraints into the Designer database.

```
timer_commit
```

Arguments

None

Supported Families

ProASIC3/E,ProASIC[PLUS], Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

None

Exceptions

None

Examples

```
timer_commit
```

See Also

timer_restore

Tcl documentation conventions

timer_get_path

Displays the path between the specified pins in the Log window.

```
timer_get_path -from source_pin -to destination_pin \
[-exp value ] \
```

```
[-sort value ]  \
[-order value ] \
[-case value ] \
[-maxpath maximum_paths ] \
[-maxexpath maximum_paths_to_expand] \
[-mindelay minimum_delay] \
[-maxdelay maximum_delay] \
[-breakatclk value ] \
[-breakatclr value ]
```

Arguments

`-from source_pin`

Specifies the name of the source pin for the path.

`-to destination_pin`

Specifies the name of the destination pin for the path.

`-exp value`

Specifies whether to expand the path. The following table shows the acceptable values for this argument:

Value	Description
yes	Expands the path
no	Does not expand the path

`-sort value`

Specifies whether to sort the path by either the actual delay or slack value. The following table shows the acceptable values for this argument:

Value	Description
actual	Sorts the path by the actual delay value
slack	Sorts the path by the slack value

`-order value`

Specifies whether the maximum list size is based on the longest or shortest paths. The following table shows the acceptable values for this argument:

Value	Description
long	Base the maximum list size on the longest path in the design
short	Base the maximum list size on the shortest path in the design

`-case value`

Specifies whether the report will include timing values for the worst, typical, or best cases. The following table shows the acceptable values for this argument:

Value	Description
worst	Includes timing values for the worst cases
typ	Includes timing values for typical cases
best	Includes timing values for the best cases

`-maxpath maximum_paths`

Specifies the maximum number of paths to display.

`-maxexpath maximum_paths_to_expand`

Specifies the maximum number of paths to expand.

`-breakatclk value`

Specifies whether to break the paths at the register clock pins. The following table shows the acceptable values for this argument:

Value	Description
yes	Breaks the paths at the register clock pins
no	Does not break the paths at the register clock pins

`-breakatclr value`

Specifies whether to break the paths at the register clear pins. The following table shows the acceptable values for this argument:

Value	Description
yes	Breaks the paths at the register clear pins
no	Does not break the paths at the register clear pins

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

None

Exceptions

None

Examples

The following example returns the paths from input port headdr_dat<31> to the input pin of register u0_headdr_data1_reg/data_out_t_31 in typical conditions.

```
timer_get_path -from "headdr_dat<31>" \
-to  "u0_headdr_data1_reg/data_out_t_31/U0:D" \
```

```
-case typ  \
-exp "yes" \
-maxpath "100" \
-maxexpath "10"
```

The following example returns the paths from clock pin of register gearbox_inst/bits64_out_reg<55> to the output port pma_tx_data_64bit[55]

```
timer_get_path -from "gearbox_inst/bits64_out_reg<55>/U0:CLK" \
     -to {pma_tx_data_64bit[55]} \
     -exp "yes"
```

See Also

Tcl documentation conventions

timer_get_clock_actuals

Displays the actual clock frequency in the Log window, when Timer is initiated.

```
timer_get_clock_actuals -clock clock_name
```

Arguments

`-clock clock_name`

Specifies the name of the clock with the frequency (or period) to display.

Supported Families

ProASIC3/E, ProASIC^{PLUS}, Axcelerator,ProASIC, MX, eX, and SX/SX-A

Notes

None

Exceptions

None

Examples

This example displays the actual clock frequency of clock "clk1" in the Log window:

```
timer_get_clock_actuals -clock clk1
```

See Also

timer_get_clock_constraints

Tcl documentation conventions

timer_get_clock_Constraints

Returns the constraints (period, frequency, and duty cycle) on the specified clock.

```
timer_get_clock_constraints -clock clock_name
```

Arguments

`-clock clock_name`

Specifies the name of the clock with the constraint to display.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

None

Exceptions

None

Examples

The following example displays the clock constraints on the clock named "clk" in the Log window:

```
timer_get_clock_constraints -clock clk
```

See Also

timer_get_clock_actuals

Tcl documentation conventions

timer_get_maxdelay

Displays the maximum delay constraint between two pins in a path in the Log window.

```
timer_get_maxdelay -from source_pin -to destination_pin
```

Arguments

`-from source_pin`

Specifies the name of the source pin in the path.

`-to destination_pin`

Specifies the name of the destination pin in the path.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

You can use the following macros in this command:

`$in()`

`to specify all input pins`

`$out()`

`to specify all output pins`

`$reg(clock_name)`

`to specify all registers related to clock_name`

Exceptions

None

Examples

The following example displays the maximum delay constraint from all registers of clk166 to all output pins in the Log window:

```
timer_get_maxdelay -from {$reg(clk166)[*]} -to {$out()[*]}
```

The following example displays the maximum delay constraint from the pin clk166 to the pin reg_q_a_9_/U0:CLK in the Log window:

```
timer_get_maxdelay -from {clk166} -to {reg_q_a_9_/U0:CLK}
```

The following example displays, in the Log window, the maximum delay constraint from all input pins to all registers of clock166 and also checks for errors in the command:

```
if [ catch {timer_get_maxdelay -from {$in()[*]} -to {$reg(clk)[*]}} ] {
  puts "Error getting max_delay information"
} else {
  puts "Successfully obtained max_delay information"
}
```

See Also

timer_set_maxdelay

Tcl documentation conventions

timer_get_path_Constraints

Displays the path constraints that were set as the maximum delay constraint in the Timer.

```
timer_get_path_constraints
```

Arguments

None

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

If no maximum delay constraints are set, this command will not report any delay values. The information is displayed in the Log window.

Exceptions

None

Examples

The following example displays the paths set in the Timer as the maximum delay constraint in the Log window:

```
timer_get_maxdelay -from {$reg(clk166)[*]} -to {$out()[*]}
```

See Also

timer_set_maxdelay

Tcl documentation conventions

timer_remove_clock_exception

Removes the previously set clock constraint.

```
timer_remove_clock_exception -clock clock_name -pin pin_name -dir {value}
```

Arguments

`-clock clock_name`

Specifies the name of the clock from which to remove the constraint.

`-pin pin_name`

Specifies the name of the pin to remove.

`-dir {from_value to_value}`

Specifies direction. The *from_value* refers to the paths that start at that pin and *to_value* refers to the paths that end at that pin.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

None

Exceptions

None

Examples

This example removes a clock exception from the pin reg_q_a_10_/U0:CLK with respect to the clock clk:

```
timer_remove_clock_exception -clock {clk} -pin {reg_q_a_10_/U0:CLK} -dir {from}
```

This example removes a clock exception to the pin reg_q_a_10_/U0:E with respect to the clock clk:

```
timer_remove_clock_exception -clock {clk} -pin {reg_q_a_10_/U0:E} -dir {to}
```

See Also

time_add_clock_exception

Tcl documentation conventions

timer_remove_pass

Removes the path pass constraint that was previously entered.

```
timer_remove_pass -pin pin_name
```

Arguments

`-pin pin_name`

Specifies the name of the pin from which to remove the path pass constraint.

Supported Families

ProASIC3/E,ProASIC[PLUS], Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

None

Exceptions

None

Examples

The following example removes the pass constraint from the clock pin reg_q_a_0_:CLK:

```
timer_remove_pass -pin {reg_q_a_0_:CLK}
```

See Also

timer_add_pass

Tcl documentation conventions

timer_remove_stop

Removes the path stop constraint on the specified pin that was previously entered.

```
timer_remove_stop -pin pin_name
```

Arguments

`-pin pin_name`

Specifies the name of the pin from which to remove the path stop constraint.

Supported Families

ProASIC3/E,ProASIC[PLUS], Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

Export of script writes the constraint wrong with timer_remove_pass instead of timer_remove_stop.

Exceptions

None

Examples

The following example removes the path stop constraint on the clear pin reg_q_a_0_:CLR:

```
timer_remove_stop -pin {reg_q_a_0_:CLR}
```

See Also

timer_add_stop

Tcl documentation conventions

timer_restore

Restores constraints previously committed in Timer.

```
timer_restore
```

Arguments

None

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

None

Exceptions

None

Examples

```
timer_restore
```

See Also

timer_commit

Tcl documentation conventions

timer_setenv_clock_freq

Sets a required clock frequency for the specified clock in megahertz (MHz).

```
timer_setenv_clock_freq -clock clock_name -freq value [-dutycycle dutycycle]
```

Arguments

`-clock clock_name`

Specifies the name of the clock for which to set the required frequency.

`-freq value`

Specifies the frequency in MHz.

`-dutycycle dutycycle`

Specifies the duty cycle for the clock constraint.

Notes

None

Exceptions

None

Examples

The following example sets a clock frequency of 100MHz on the clock clk1:

```
timer_setenv_clock_freq -clock {clk1} -freq 100.00
```

See Also

timer_setenv_clock_period

Tcl documentation conventions

timer_setenv_clock_period

Sets the clock period constraint on the specified clock.

```
timer_setenv_clock_period -clock clock_name [-unit {value}] -period period_value [-dutycycle dutycycle]
```

Arguments

`-clock clock_name`

Specifies the name of the clock for which to set the period.

`-unit {value}`

Specifies the unit for the clock period constraint. The default is ns. The following table shows the acceptable values for this argument:

Value	Description
ns	nanoseconds
ps	picoseconds

`-period period_value`

Specifies the period in the specified unit.

`-dutycycle dutycycle`

Specifies the duty cycle for the clock constraint.

Notes

None

Exceptions

None

Examples

The following example sets a clock period of 2.40ns on the clock named clk1:

```
timer_setenv_clock_period -clock {clk1} -unit {ns} -period 2.40
```

See Also

Tcl documentation timer_setenv_clock_freq

conventions

timer_set_maxdelay

Adds a maximum delay constraint to the specified path.

```
timer_set_maxdelay -from source_pin -to destination_pin [-unit {value} -delay
delay_value
```

Arguments

`-from source_pin`

Specifies the name of the source pin in the path.

`-to destination_pin`

Specifies the name of the destination pin in the path.

`-unit {value}`

Specifies whether the delay unit is in nanoseconds or picoseconds. The following table shows the acceptable values for this argument:

Value	Description
ns	Sets the delay in nanoseconds
ps	Sets the delay in picoseconds

`-delay delay_value`

Specifies the actual delay value for the path.

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

You can use the following macros in this command:

`$in()`

`to specify all input pins`

`$out()`

`to specify all output pins`

`$reg(clock_name)`

`to specify all registers related to clock_name`

Exceptions

None

Examples

The following example sets a maximum delay of 20 nanoseconds from all registers of clk166 to all output pins:

```
timer_set_maxdelay -from {$reg(clk166)[*]} -to {$out()[*]} -unit {ns} -delay 20.00
```

The following example sets a maximum delay of 30 nanoseconds from all input pins to all output pins:

```
timer_set_maxdelay -from {$in()[*]} -to {$out()[*]} -unit {ns} -delay 30.00
```

See Also

timer_get_maxdelay

Tcl documentation conventions

timer_remove_all_Constraints

Removes all the timing constraints in the current design.

```
timer_remove_all_constraints
```

Arguments

None

Supported Families

ProASIC3/E,ProASIC^{PLUS}, Axcelerator, ProASIC, MX, eX, and SX/SX-A

Notes

None

Exceptions

None

Examples

The following example removes all of the constraints from the design and then commits the changes:

```
timer_remove_all_constraints
timer_commit
```

See Also

timer_commit

Tcl documentation conventions

About Design Constraints

Design constraints are specifications for placing, implementing, naming, and timing considerations of physical and logical assignments. They are usually either restrictions or properties in your design. There are several types of constraints: routing, timing, area, mapping, and placement constraints.

Timing constraints

ocation and region assignment constraintsLocation and region assignment constraints (placing and routing)

I/O assignment constraints (pin location and I/O attributes)

Attributes

You use constraints to ensure that a design meets timing performance and required pin assignments.

Designer supports both physical and timing constraints. You can set constraints by either using Actel's interactive tools or by importing constraint files directly into Designer.

Designer Naming Conventions

The names of ports, instances, and nets in an imported netlist are sometimes referred to as their *original names*. Port names appear exactly as they are defined in a netlist. For example, a port named A/B appears as A/B in ChipPlanner, PinEditor, and I/O Attribute Editor in MultiView Navigator. Instances and nets display the original names plus an escape character (\) before each backslash (/) and each slash (\) that is not a hierarchy separator. For example, the instance named `A/\B` is displayed as `A\/\\B`.

The names of ports, instances, and nets that you use in PDC commands depends on the device for which you are writing the PDC command. Following are Designer's naming conventions by device.

ProASIC3/E

The following components use a netlist's original names:

PDC reader/writer

SDC reader/writer

Compile report

SDF/Netlist writer for back annotation

MultiView Navigator tools: PinEditor, ChipPlanner, and I/O Attribute Editor (NetlistViewer uses compile names instead)

The following components use a netlist's compile names:

NetlistViewer in MultiView Navigator

Timer

Smart power

Axcelerator

Use the compiled names for all components.

ProASIC and ProASIC PLUS

For ports, instances, and nets, use their original names in GCF files.

See Also

PDC Naming Conventions

SDC Command Limitations

Timing Constraints

Timing constraints can be entered using the interactive Timer tool or by importing a constraint file.

Constraint File Type	Supported Families
SDC	Axcelerator, ProASIC PLUS, ProASIC3/E, SX-A, and eX
DCF	SX, SX-A, MX, eX, ACT1, ACT2, and ACT3
GCF	ProASIC ONLY

To understand the complexity of a design and its performance, perform place-and-route with no constraints to see if routing can complete without constraints. If routing completes successfully, you can open Timer to see if the physical design meets timing requirements.

ProASIC only: If you are using a synthesis tool such as Synopsys Design Compiler, Actel recommends that you use it to generate a forward SDF file containing path constraints only.

Over constraining a design may result in increased place-and-route run times, while not improving design performance.

Location and Region Assignment Constraints

Location and region assignment constraints are physical constraints for setting location and region assignments for a specific architecture and device.

You can create and edit regions on your chip and assign logic to those regions using the ChipPlanner tool or by importing constraint files. ChipPlanner works with **ProASIC3E**, **ProASIC3**, **ProASIC PLUS**, **Axcelerator**, and **ProASIC** devices.

Alternatively, you can import constraint files to set or change your location and region assignments. The type of constraint file you use depends on your device family. See Types of physical constraints for more information.

I/O Assignment Constraints

I/O assignment constraints are physical constraints for setting the pin location and I/O attributes for a specific architecture and device.

You can configure and assign input and output macros and their attributes to pins using the PinEditor tool or by importing constraint files. There are two different versions of PinEditor. The version you use depends on which product family you are designing for:

For **ProASIC3E, ProASIC3, ProASIC PLUS**, **Axcelerator**, and **ProASIC** devices, use PinEditor in MultiView Navigator. To edit I/O attributes, use the I/O Attribute Editor tool.

For all other families, use the standalone version of PinEditor which has a built-in I/O attribute editor.

When you open your design (.adb) file, Designer automatically presents you with the appropriate tools in the Design Flow window.

You can also choose to import constraint files to set or change your location and region assignments. The type of constraint file you use depends on your device family. See Types of physical constraints for more information.

Attributes

In an FPGA schematic, attributes are the instructions assigned to symbols or nets. These instructions can indicate the placement, routing, naming, and other characteristics of the symbols or nets. Designer uses this information during the place-and-route of a design. Some constraints are also referred to as attributes.

See "I/O Attribute Editor" in the MultiView Navigator User's Guide for more detailed information about input and output attributes. For details about specific attributes, see the Designer User's Guide.

Overview - Entering Constraints

You can enter design constraints in two ways:

Using a **constraint editor tool**. Designer's constraint editors are graphical user interface (GUI) tools for creating and modifying physical, logical, and timing constraints. Using these tools enables you to enter constraints without having to understand GCF, PDC, or other file syntax. Which constraint editor you use depends on which type of device you are designing.

For **ProASIC3E, ProASIC3, ProASIC PLUS**, **Axcelerator**, and **ProASIC** devices, use the tools within the MultiView Navigator:

- ChipPlanner - Sets location and region assignments

- PinEditor in MVN - Sets the pin location constraints

- I/O Attribute Editor - Sets I/O attributes

For all other families, you will use the following tools:

- ChipEditor - Sets location and region assignments

- PinEditor Standalone - Sets I/O attributes and pin location constraints

Importing a **constraint file**: GCF, PDC, SDC, DCF, and PIN. The type of file you use depends on which type of device you are designing.

- GCF (ProASIC and ProASIC PLUS families)

- PDC (Axcelerator, ProASIC3E, and ProASIC3 families)

- SDC (Axcelerator, ProASIC3E, ProASIC3, ProASIC PLUS, eX, and SX-A families)

- DCF (earlier Antifuse families such as eX, SX-A, and SX)

- PIN (only valid for earlier Antifuse families such as eX, SX-A, and SX)

Using Constraint Editors

Assigning I/O Constraints

I/O assignment constraints are physical constraints for setting the pin location and I/O attributes for a specific architecture and device.

To assign I/O constraints for **ProASIC3E**, **ProASIC3**, **ProASIC PLUS**, **Axcelerator**, and **ProASIC** families, use PinEditor and I/O Attribute Editor in MultiView Navigator.

PinEditor in MVN - Sets the pin location constraints.

I/O Attribute Editor - Sets I/O attributes.

For all other families, use PinEditor Standalone to set the I/O attributes and pin location constraints.

PinEditor is the package layout tool for assigning I/O ports to package pins. You can use the I/O Attribute Editor tool for viewing, sorting, and editing I/O attributes. Additionally, you can use this tool to lock and unlock pins in your design. PinEditor Standalone has a built-in I/O attribute editor.

You can also assign I/O constraints in PDC or GCF files and then import them into your design.

See the MultiView Navigator User's Guide for more information on how to assign and modify I/O constraints. See the Design Constraints chapter of the Designer User's Guide for more information about constraints.

Assigning Location and Region Constraints

You can set constraints for locations and regions using the ChipPlanner or ChipEditor tool as well as via constraint files. You can use constraints to view routing information and influence the place and route of your design for more optimal results

The tool you use depends on which product family you are designing for:

For **ProASIC3E**, **ProASIC3**, **ProASIC PLUS**, **Axcelerator**, and **ProASIC** devices, use ChipPlannerChipPlanner. ChipPlanner is the floorplanning tool you use to create and edit regions on your chip and assign logic to these regions.

For other design families, use ChipEditorChipEditor. ChipEditor is a graphical application for viewing and assigning I/O and logic macros.

If you choose to use constraint files to set your location and region assignments, the exact constraint file you use depends on your device family. See Types of physical constraintsTypes of physical constraints for more information.

When you open your design (.adb) file, Designer automatically presents you with the appropriate tools in the Design Flow window.

You can also assign location and region constraints in PDC or GCF files and then import them into your design.

See the MultiView Navigator User's Guide for more information on how to assign and modify location and region constraints. See the Design Constraints chapter of the Designer User's Guide for more information about constraints.

About Physical Design Constraint (PDC) Files

A PDC file is a Tcl script file specifying physical constraints. This file can be imported and exported from Designer. Any constraint that you can enter using the PinEditor in MVN or ChipPlanner tool, you can also use in a PDC file.

Note: Only ProASIC3/E, and Axcelerator devices support PDC files.

Designer supports the following PDC commands.

Command	Action
assign_global_clock	Assigns user-defined nets to global clock networks by promoting the net using a CLKINT macro
assign_local_clock	Assigns user-defined nets to local clock routing (Axcelerator) or to either LocalClock or QuadrantClock regions (ProASIC3/E)
assign_net_macros	Assigns the macros connected to a net to a specified defined region
assign_region	Assigns macros to a pre-specified region
define_region	Defines a rectilinear region
define_region	Defines a rectangular region
delete_buffer_tree	Removes all buffers and inverters from a given net for ProASIC3 and ProASIC3E devices
dont_touch_buffer_tree	Restores all buffers and inverters that were removed from a given net with the delete_buffer_tree command

move_region	Moves a region to new coordinates
reset_floorplan	Deletes all defined regions. Placed macros are not affected
reset_io	Resets all attributes on a macro to the default values
reset_iobank	Resets an I/O banks technology to the default technology
reset_net_critical	Resets net criticality to default level
set_io	Sets the attributes of an I/O
set_iobank	Specifies the I/O bank's technology
set_location	Places a given logic instance at a particular location
set_multitile_location	Assigns specified two-tile and four-tile macros to specified locations on the chip
set_net_critical	Sets net criticality, which is issued to influence placement and routing in favor of performance
set_vref	Specifies which pins are VREF pins
set_vref_defaults	Sets the default VREF pins for specified banks
unassign_global_clock	Assigns clock nets to regular nets
unassign_local_clock	Unassigns the specified user-defined net from a LocalClock or QuadrantClock region
unassign_macro_from_region	Unassigns macros from a specified region, if they are assigned to that region
unassign_net_macro	Unassigns macros connected to a specified net from a defined region
undefine_region	Removes the specified region

Note: PDC commands are case sensitive. However, their arguments are not.

See Also

About design constraints

Exporting files

Importing PDC files

Importing auxiliary files

PDC syntax conventions

PDC naming conventions

Importing PDC Files (ProASIC3E, ProASIC3, and Axcelerator families only)

You can import a PDC file as either a source, or as an auxiliary file.

Importing PDC file as a Source file

When importing a PDC file as a source file, the netlist must also be imported along with the PDC commands. Furthermore, when importing the PDC file as a source file, you have the option of keeping the existing physical constraints. The Keep Existing Physical

Constraints option in the Import Source Files dialog box enables you to merge or replace existing constraints when you re-import new or modified PDC file.

Select the **Keep Existing Physical Constraints** option to preserve all existing physical constraints that you have entered either using one of the MVN tools (ChipPlanner, PinEditor, or the I/O Attribute Editor) or a previous PDC file. The software will resolve any conflicts between new and existing physical constraints and display the appropriate message. The Keep Existing Constraints option is **Off** by default. When this option is **Off**, all the physical constraints in the newly imported PDC files are used. All pre-existing constraints are lost. When this option is **On**, the physical constraints from the newly imported PDC files are merged with the existing constraints.

Importing PDC file as an Auxiliary file

When importing a PDC file as an auxiliary file, the new or modified PDC constraints are merged with the existing constraints. The software resolves any conflicts between new and existing physical constraints and displays the appropriate message. The following PDC commands are not supported when the PDC file is imported as an auxiliary file:

set_io - register Yes | No

assign_local_clock

unassign_local_clock

assign_global_clock (not for Axcelerator)

unassign_global_clock (not for Axcelerator)

delete_buffer_tree (not for Axcelerator)

dont_touch_buffer_tree (not for Axcelerator)

Note: See the help topic for each command in your PDC file to make sure it is supported in an auxiliary file.

You can specify the following types of constraints in a Physical Design Constraint (PDC) file:

I/O standards and attributes

VCCI and VREF for all or some of the banks

Pin assignments

Placement locations

Net criticality (Axcelerator only)

Region creation and assignment

LocalClock creation and deletion

Global clock assignments and un-assignments

Note: File names or paths with spaces may not import into Designer. Rename the file or path, removing the spaces, and then re-import the file.

Types of Constraints

Constraints are used to ensure that a design meets timing performance and required pin assignments. For ProASIC and ProASIC^{PLUS} families, the types of constraints that can be defined in a GCF constraint file include:

Timing constraints

Global resource constraints

Netlist optimization constraints

Placement constraints

ProASIC and ProASICPLUS Timing Constraints

Timing constraints are used to ensure that a design meets the required timing performance.

ProASIC Timing Constraints

Constraints can be entered using a ProASIC constraints file (.gcf) or using an SDF path constraints file. These forward SDF files are generated by synthesis tools. The two formats cannot be combined in one file. However, SDF files and ProASIC (.gcf) constraint files can be used for the same design. Place-and-Route considers timing constraints and attempts to meet them. After routing, Designer displays a message that indicates if your timing constraints were met.

ProASICPLUS Timing Constraints

ProASIC^{PLUS} supports only SDC timing constraint files. (GCF timing constraints are no longer supported.) If you open a ProASIC^{PLUS} design with GCF timing constraints in Designer it converts your GCF timing constraints to SDC constraints automatically. After your GCF timing constraints are converted Designer creates an SDC file with your new constraint information. You do NOT have to re-import your SDC file to get your constraints, it happens internally. See the related topics below for more information on GCF to SDC timing constraint conversion.

Place-and-route considers your timing constraints and attempts to meet them. After routing, Designer displays a message that indicates if your timing constraints were met.

GCF to SDC Timing Constraints Conversion

Each GCF timing constraint maps to a specific SDC timing constraint. Use the list below to evaluate the corresponding constraints.

Note: GCF to SDC conversion is only supported for ProASIC^{PLUS} devices.

GCF Timing Constraint	SDC Timing Constraint
create_clock	create_clock
set_false_path	set_false_path
set_input_to_register_delay	set_max_delay
set_multicycle_path	set_multicycle_path
set_register_to_output_delay	set_max_delay
set_max_path_delay	set_max_delay

GCF Syntax Conventions

A ProASIC constraint consists of a statement and an argument, terminated by a semicolon. Statements are not case sensitive. However, cell instance, net, and port names used as arguments may be quoted and are case sensitive. Except for white spaces, all ASCII characters can be used. Comments are allowed in constraints files and must be preceded by two forward slashes (//). Time

values are given in nanoseconds. When constraints are duplicated, the last one specified for a specific item overwrites any previous similar constraints already specified for the considered item.

This section describes syntax conventions for notation, user data variables, and comments. Comments begin with double slashes (//) and are terminated by a newline character.

Syntax Conventions for Notation

Notation	Description
item	Represents a syntax item
item ::= definition	item is defined as definition
item ::= definition1 ||= definition2	item is defined as either definition1 or definition2 (Multiple alternative syntax definitions are allowed)
[item]	Item is optional
{ item }	Item is a list of required items. At least one item must appear.
KEYWORD	Keywords appear in uppercase characters in bold type for easy identification, but are not case sensitive.
VARIABLE	Represents a variable and appears in uppercase characters for easy identification

Syntax Conventions for User Data Variables

Variable	Description
FILEIDENTIFIER	Represents a hierarchical filename.
IDENTIFIER	Represents the name of a design object. Can be a block, cell instance, net, or port. IDENTIFIERS can use any ASCII character except the white space and the slash (/), which is the hierarchical divider character (see QPATH below). IDENTIFIERS are case sensitive
POSFLOAT	Represents a positive real number; for example, 4.3, 1.15, 2.35

POSNUMBER	Represents a positive integer; for example, 1, 12, 140, 64. When representing time, POSNUMBER is expressed in nanoseconds (ns)
QPATH	Represents a hierarchical IDENTIFIER. The levels of the hierarchy are represented by IDENTIFIERS divided by a slash (/). The QPATH hierarchical IDENTIFIER may or may not be quoted

Synopsys Design Constraints (SDC) Files

Designer accepts an SDC constraint file generated by a third-party tool. This file is used to communicate design intent between tools and provide clock and delay constraints. The Synopsys Design Compiler and Prime Time can generate SDC descriptions, or you can generate the SDC file manually.

Command	Action
create_clock (SDC clock constraint)	Determines the maximum register-to-register delay in the design
set_false_path (SDC false path constraint)	Identifies paths in the design that are to be marked as false, so that they are not considered during timing analysis
set_max_delay (SDC max path constraint)	Sets the path delay of the specified ports to a restricted value
set_multicycle_path (SDC multiple cycle path constraint)	Defines the multicycle path
set_load (SDC load constraint)	Sets the capacitance to a specified value on a specified port

Generated SDC files

There can be slight differences between a user-generated SDC file and SDC files generated by other tools. For example, suppose you write the following constraint:

```
create_clock -period 100 clk
```

The SDC file from Design Compiler generates the same constraint in a different format:

```
create_clock -period 100 -waveform {0 50}
[get_ports {clk}]
```

The SDC file from Prime Time generates this constraint in yet another format:

```
create_clock -period 100.000000 -waveform
{0.000000\ 50.000000}[get_ports {clk}]
```

As long as constraint syntax and arguments conform to the syntax rules described in Designer online help, the SDC files are accepted by Timer.

About DCF Files

Delay constraint information can be described in a *.dcf file and imported into Designer. The DCF language was developed to interact directly with the Timer tool and is therefore not a recommended method.

Note: DCF files are only valid with earlier Antifuse families such as eX, SX-A, and SX. Although they are supported in eX and SX-A, Actel recommends that you use SDC files for all your constraints.

DCF files are platform dependent. If you transfer from PC to UNIX or vice-versa, you must manually translate carriage-returns (unix2dos, dos2unix, or via ftp). PC text files have an extra character for carriage returns compared to UNIX text files.

Supported command categories

Categories	Action
global_clocks	Describes the clock waveforms from the global clock distribution network; local clocks, such as gated clocks, are not directly supported
max_delays/min_delays	Describes max/min delays
io_arrival_times	Defines the arrival time to an input port
global_stops	Defines pins in don't care or false path
pin_loads	Defines the capacitance loading on package pins

DCF Syntax Rules

The syntax rules for DCF are listed below. Note that these rules cannot be used as a parsing grammar. Terminal symbols are in upper case. Non-terminal symbols, which are enclosed with <>, are in lower case. Symbols enclosed with [] are optional. The symbol | separates alternatives.

```
<DCF> =
<sec_def_name>
<sec_io_arr>
<sec_min_del>
<sec_max_del>
<sec_clk>
<sec_global_stop>
<sec_def_name> =
SECTION TOP_LEVEL_DEF_NAME <stop>
<variable>.
END <stop>
<sec_io_arr> =
SECTION IO_ARRIVAL_TIMES <stop>
[<io_arr_clauses>]
END <stop>
<io_arr_clauses> = <io_arr_clause> | <io_arr_clause> <io_arr_clauses>
<io_arr_clause> = [<number>:] <number> <timeunit> <io_list>.
<io_list> = <io> | <io> <io_list>
<io> = INPAD | OUTPAD| <variable>
<sec_max_del> =
```

```
SECTION MAX_DELAYS <stop>
<delay_clauses>
END <stop>
<sec_min_del> =
SECTION MIN_DELAYS <stop>
<delay_clauses>
END <stop>
<delay_clauses> = <delay_clause> | <delay_clause> <delay_clauses>
<delay_clause> =
DELAY <time>; SOURCE <source_list>; SINK <sink_list>;
[STOP <stop_list>]; [PASS <pass_list>].
<source_list> = {<sources>} [EXCEPT {<sources>}]
<sources> = INPAD | CLOCKED | <name_list>
<name_list> = <variable> | <variable> <name_list>
<sink_list> = {<sinks>}[EXCEPT {<sinks>}]
<sinks> = OUTPAD | GATED | <name_list>
<stop_list> = {<name_list>} [EXCEPT {<sinks>}]
<pass_list> = {<name_list>} [EXCEPT {<sinks>}]
<sec_clk> =
SECTION GLOBAL_CLOCKS <stop>
[<waveform_clauses>]
[<relational_clauses>]
END <stop>
<waveform_clauses> = <waveform_clause> | <waveform_clause> <waveform_clauses>
<waveform_clause> = WAVEFORM <variable> RISE <time>
FALL<time> PERIOD <time> [EXCEPT SOURCE {macrolist}]
[EXCEPT SINK {macrolist}].
<relational_clauses> = <check_clause> | <check_clause> <check_clauses>
<check_clause> =
MULTICYCLE <variable> SOURCE CYCLE<value> [EXCEPT <name_list>]
[; DESTINATION <clkname> CYCLE<value> <clkname> CYCLE<value>
[EXCEPT<name_list>]].
<clkname> = <clockMacro>
<time> = <number> <unit>
<number> = <int>
<stop> =. | /* NULL */
<unit> = NS | MS | PS
<variable> = same as variable in C language.
```

```
<int> = same as int in C language.
sec_global_stops> =
Section GLOBAL_STOPS.
{<pinNameList>}.
End.
<sec_pin_loads> =
Section PIN_LOADS.
<pinLoadClauses>
End.
<pinLoadClauses> = <pinLoadClause> | <pinLoadCause>l<pinLoadClauses>]
<pinLoadClause> = <number> <capUnit> [TTL | CMOS] <pinNameList>.
<capUnit> = PF | NF | UF | MF
```

About PIN Files

Pin location information may be described in a *.pin file and imported into Designer.

Note: PIN files are only valid with earlier Antifuse families such as eX, SX-A, and SX. Although pin files are supported in Axcelerator, Actel recommends that you use PDC files for all your constraints.

Supported syntax

Keywords	Action
DEF	Define top-level design entity
PIN	Define I/O location

DEF

Syntax: DEF <design_name>.

Example: DEF TARG32_WRP.

This example defines top-level structure as TARG32_WRP.

PIN

Syntax:

PIN <pin_name>;
PIN:<package_pin_number>.

Example:

PIN RST;
PIN:156.

This example assigns signal RST to package pin 156.

Importing Auxiliary Files

Auxiliary files are not audited and are treated more as one-time data-entry or data-change events, similar to entering data using one of the interactive editors (e.g. PinEditor or Timer).

Some timing constraints (such as multi_cycle) are not supported in the Timer GUI and must be implemented by importing the SDC file. If you import the SDC file as an auxiliary, you do not have to re-compile your design. However, auditing is disabled when you import auxiliary files, and Designer cannot detect the changes to your SDC file(s) if you import them as auxiliary files.

Auxiliary Files	File Type Extension	Family
Criticality	*.crt	ACT1, ACT2, ACT3, MX, XL, DX
PIN	*.pin	ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX
SDC	*.sdc	ProASIC3/E, SX-A, eX, Axcelerator, ProASIC PLUS
Physical Design Constraint	*.pdc	ProASIC3/E and Axcelerator
Value Change Dump	*.vcd	ProASIC3/E, Axcelerator, ProASIC, ProASIC PLUS
Switching Activity Intermediate File/Format	*.saif	ProASIC3/E, Axcelerator, ProASIC, ProASIC PLUS
Design Constraint File	*.dcf	Axcelerator, ACT1, ACT2, ACT3, MX, XL, DX, SX, SX-A, eX

To import an auxiliary file:

1. From the **File** menu, select **Import Auxiliary Files**. The Import Auxiliary Files dialog appears,
2. Click the **Add** button. The Add Auxiliary Files dialog box appears.
3. Select your file and click **Import**. The file is added to the Import Auxiliary Files dialog box. Continue to add more auxiliary files to the list. Some formats (like DCF and SDC) are not allowed to be imported in multiple auxiliary files.

   Modifying:If you need to modify a selection, select the file row and click **Modify**

   Deleting:If you need to delete a file, select the file row and click **Delete**.

   Ordering:Ordering your auxiliary files. Select and drag your files to specify the import order. Specifying a priority is useful if you are importing multiple PDC files.
4. After you are done adding all your Auxiliary files, click **OK**. Your auxiliary files are imported. Any errors appear in Designer's Log Window.

Note:

.vcd and .saif are used by SmartPower for power analysis.

.crt for backwards compatibility with existing designs only.

File names or paths with spaces may not import into Designer. Rename the file or path, removing the spaces, and re-import.

I/O Standards Compatibility Matrix

Not all I/O standards are compatible with each other in the same device. Click the device name in the following list to see which I/O standards are compatible for your device:

ProASIC3E

ProASIC3

Axcelerator

I/O Standards and I/O Attributes Applicability

Not all I/O attributes are applicable to all I/O standards. Click the device name in the following list to see the I/O attributes that you can modify per I/O standard for your device:

ProASIC3E

ProASIC3

Axcelerator

GCF Constraint Quick Reference

use_global

About Global ReSource Constraints

Each ProASIC and ProASIC[PLUS] device includes four global networks that have access to every tile. These four global networks provide high speed, low skew routing resources to signals such as clocks and global resets.

Once the netlist is imported, Designer sets global resource parameters and promotes the highest fanout nets to the remaining global resources unless the dont_fix_globals statement has been specified in a constraint file.

Note: When using the dont_fix_globals statement, global assignments made in the constraint files and design netlist will be honored (the constraint file entries will take precedence).

These global resource parameters can be supplemented by including global resource constraints in a constraint file. Global resource constraints can define which signals are assigned to global resources and which signals cannot be promoted to global resources. Global resource constraints can also override the default action that selects high fanout nets for use by the global resources.

Priority Order for Global Promotion

While assigning signals to global resources, Designer considers this information in the given priority:

1. set_global and set_io statements (instances of those global cells, which cannot be demoted)
2. Nets with the highest potential fanout above 32 (after removal of all buffers and inverters)
3. Global cell instantiation in a netlist (global cells which can be demoted)

Note: By default, a net with a fanout of less than 32 will not be promoted to global automatically unless the set_global or set_io constraint statements is used for this net. You can override this threshold of 32 by using the set_auto_global_fanout constraint statement.

dont_fix_globals

Use this statement to turn off the default action that automatically corrects the choice of global assignment to use only the highest fanout nets.

```
dont_fix_globals;
```

read

Use this statement to specify the name of a constraint file. A constraint file can contain multiple read statements. For example, you can put pin assignments in one file, optimization constraints in another, placement constraints in yet another, and read them all in through a master constraint file. The syntax is:

```
read [ -eco] [ -initial] file ;
```

where

`-eco` specifies that the constraint file is to be read in eco mode (engineering change order). In this mode, no errors will be reported when certain nets or instances are not found in the design. Instead a warning is generated.

`-initial` specifies that the constraint file is to be read in initial mode. In this mode, all fixed location statements will be interpreted as initial locations instead.

`file` (required) is the name of the constraint file, surrounded by double quotes.

The following example statements instruct the Designer to use constraints from the GCF files pinmap.gcf and decoder.gcf. A full path specification is given for pinmap.gcf. The file decoder.gcf has no path specification and is assumed to be in the design working directory.

```
read "/net/aries/designs/pinmap.gcf";
read "decoder.gcf";
```

set_auto_global

Use this statement to specify the maximum number of global resources to be used. The tool assigns global resources to high-fanout signals automatically.

If the user specifies a number that exceeds the actual number of global resources available in the device, Designer ignores the statement. If the user specifies 0, no automatic assignment to global resources will take place.

```
set_auto_global number ;
```

For example, the following statement specifies that of the possible four global nets available, the tool can automatically promote only two high-fanout nets:

```
set_auto_global 2;
```

set_auto_global_fanout

Use this statement to set the minimum fanout a net must have to be considered for automatic promotion to a global. By default this is set to 32.

```
set_auto_global_fanout number ;
```

For example, the following statement determines that a net must have at least a fanout of 12 before Designer considers it for automatic promotion to a global resource.

```
set_auto_global_fanout 12;
```

set_global

Use this statement to classify nets as global nets.

```
set_global hier_net_name [, hier_net_name … ];
```

For example:

```
set_global u1/u3/net_clk, u3/u1/net_7;
```

set_noglobal

Use this statement for classifying nets to avoid automatic promotion to global nets.

```
set_noglobal hier_net_name [ , hier_net_name … ];
```

For example:

```
set_noglobal u2/u8/net_14;
```

If the net was previously assigned to a global resource, this statement will demote it from the global resource.

use_global

This statement allows you to specify a single spine (LocalClock) or a rectangle of spine region which may encompass more than one spine region.

```
use_global T2 <net_name>;
```

```
use_global B1, T3 <net_name>;
```

For example, if you give the spine rectangle as B1, T3, the driven instances of the given net get a region constraint which encloses the rectangle, including the spine rectangle B1, T1, B2, T2, B2, T3.

The constraint tries to place the driver as close to center of the rectangle as possible.

The RAMs and I/Os are assigned to the LocalClock region unless the Compile option "Include RAM and I/O in Spine and Net Regions" is cleared. For designs created with v5.1 or earlier, this option is cleared by default. See "Compile Options" in the online help for more information.

You can specify the following type of rectangles:

Bn, Bm : n<=m will mean Bn, Bn+1, ... Bm

Tn, Tm : n<=m will mean Tn, Tn+1, ... Tm

Bn, Tm : n<=m will mean Bn, Tn, Bn+1,Tn+1 ... Bm, Tm

Tn, Bm : n<=m will mean Bn, Tn, Bn+1,Tn+1 ... Bm, Tm

See table for a summary of available spines.

Global Spine Usage

Device	Spine
A500K050	T1 to T3
	B1 to B3
A500K130	T1 to T5
	B1 to B5
A500K180	T1 to T6
	B1 to B6
A500K270	T1 to T7
	B1 to B7

APA075	T1 to T3
	B1 to B3
APA150	T1 to T4
	B1 to B4
APA300	T1 to T4
	B1 to B4
APA450	T1 to T6
	B1 to B4
APA600	T1 to T7
	B1 to B7
APA750	T1 to T8
	B1 to B8
APA1000	T1 to T11
	B1 to B11

Note that T1 and B1 are the leftmost top and bottom global spines, respectively.

Netlist Optimization Constraints

Netlist optimization attempts to remove all cells from a netlist that have no effect on the functional behavior of the circuit. This reduces the overall size of a design and produces faster place-and-route times. This optimization is based on the propagation of constants and inverter pushing and takes advantage of inverted inputs of the basic logic elements. Refer to the *ProASIC 500K Family* and *ProASIC* ^{PLUS} datasheet for detailed information.

Netlist optimization can be controlled by including netlist optimization constraints in constraint files submitted to Designer.

By default, all optimizations will be performed on the netlist. To control the amount of optimization that takes place, netlist optimization constraints can be used. Netlist optimization constraints can turn off all optimizations or disable the default action that

allows all optimizations to limit the type of optimizations performed. The constraints can also define a maximum fanout to be allowed after optimizations are performed and isolate particular instances and hierarchical blocks from the effect of optimization.

After completion of netlist optimization, the design is a functionally identical representation of the design produced internally for use by Designer. View the design's layout after successful placement and routing. After optimization, a number of instances that do not contribute to the functionality of the design may have been removed.

To keep the SDF file consistent with the original input netlist, deleted cells are written with zero delay so that back-annotation is performed transparently.

Netlist Optimization Constraint Syntax

The following netlist optimization options are available for all netlist optimization constraints.

buffer - removes all buffers in the design, provided that the maximum fanout is not exceeded.

const - replaces all logical elements with one or more inputs connected to a constant (logical "1" or "0") by the appropriate logic function. If the replacement logic function is identified as an inverter or buffer, that element is removed.

dangling - recursively removes all cells driving unconnected nets.

inverter - removes all inverters in the design provided that the maximum fanout is not exceeded.

dont_optimize

This statement does not optimize your buffers or inverters; instead, it removes them. When followed by one or more of the netlist optimization options, this statement turns off the named option (and preserves it).

If you have buffers or inverters that are connected to global nets, promoted global nets, or spine nets, this command is ignored and buffers and inverters are still removed. To avoid removing them, use the dont_touch option.

```
dont_optimize [{ <option> }];
```

Where <option> is one or more of the following:

```
buffer, inverter, const, dangling
```

dont_touch

This statement enables you to selectively disable optimization of named hierarchical instances. You can use the wildcard character (*) to isolate all sub-blocks under the named block. If you use this constraint, any instances (including buffers and inverters that are connected to global nets, promoted global nets, and spine nets) stay intact.

```
dont_touch hier_instance_name [, hier_instance_name … ];
```

optimize

This statement turns on all netlist optimizations (the default mode). When followed by one or more of the netlist optimization types, this statement enables only the named optimization(s).

```
optimize [{ inverter buffer const dangling} ];
```

For example:

```
optimize buffer inverter;
```

set_max_fanout

Use this statement to specify the maxFanout limit on the specified nets. Use when optimizing the buffers and inverters. The buffers and inverters are not removed if the fanout for the given net exceeds the given limit. If no net name is given, then the command is applied to all the nets in the design. The net name can be a simple net or a name having wildcard characters.

The set_max_fanout constraint is optimized to accept individual net names. If you specify a net name, the set_max_fanout constraint applies only to the named net or nets and not to the entire design.

```
set_max_fanout NUMBER <net_name_wildcard>;
```

Placement Constraints

It is possible to use placement constraints to specify block-instance and macro placement. You can specify initial, fixed, region, and macro placements. Also, placement obstructions (locations that are not to be used and thus to be kept empty during placement instances) can be specified.

For example, a constraint that places two connected blocks close together usually improves the timing performance for those blocks. Similarly, a constraint that assigns an I/O pin to a particular net forces the router to make the connection between the driving or receiving cell and the I/O itself.

Like all constraints, placement constraints limit Designer's freedom when processing the design. For instance, assigning a fixed location makes that location unavailable during placement optimization. Such removal usually limits the program's ability to produce a chip-wide solution.

Command	Action
net_critical_ports	Specifies a specific subset of critical ports on a net
set_critical	Specifies critical nets and their relative criticality over other critical nets
set_critical_port	Identifies design I/O ports that have above-normal criticality. The criticality number scales is the same for the set_critical statement
set_empty_io	Specifies a location in which no I/O pin should be placed; the location can be specified by side and offset or by name
set_empty_location	Specifies a location in which no cell should be placed
set_initial_io	Assigns package pins to I /O ports or locates I /O ports at a specified side of a device
set_initial_location	Locates a cell instance at specified x, y coordinates
set_io	Assigns package pins to I/O ports or locates I/O ports at a specified side or location of a device; this constraint is a hard constraint and cannot be overruled by the placer
set_location	Locates a cell instance at specified x,y coordinates
set_net_region	Enables you to put all the connected instances, driver, and all the driven instances for the net(s) into the target rectangle specified in the constraint

Macro

```
macro name (x1, y1 x2, y2) {

macro_statements

}
```

Where *name* is the macro name identifier, x1, y1 is the lower left coordinates of the macro, and x2, y2 is the upper right coordinates of the macro. The macro constraint must precede the corresponding set_location that places the macro in the GCF file(s).

For example:

```
macro mult (1,1 6,6) {
set_location...
}
```

Now you can use the“set_location”or set_initial_location statements to place or initially place a sub-design instance by calling its macro and then applying a translation and rotation. .

`set_initial_location (x, y)` *hier_subdesign_inst_name*

macro_name `[transformations];`

For example:

```
set_location (3,3) a/b mult flip lr;
```

Where hier_subdesign_inst_name is the hierarchical name of the instance of the sub-design, x, y is the final location of the lower left corner of the macro after all transformations have been completed, macro_name - is the name of previously defined macro, and transformations are optional, and any of the following in any order:

flip lr - flip cell from left to right

flip ud - flip cell from up to down

rotate 90 cw - rotate 90° clockwise

rotate 270 cw - rotate 270° clockwise

rotate 90 ccw - rotate 90° counter-clockwise

rotate 180 ccw - rotate 180° counter-clockwise

rotate 270 ccw - rotate 270° counter-clockwise

The transformations are processed in the order in which they are defined in the statement.

For example:

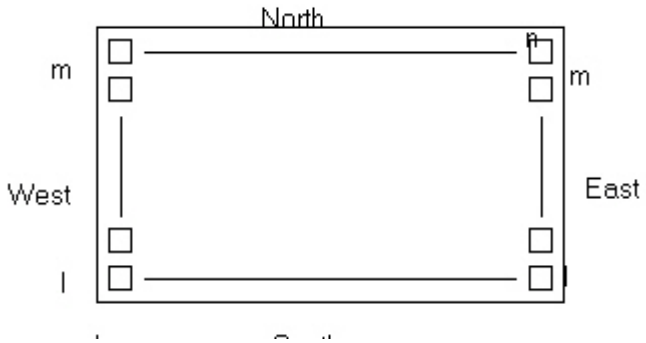
```
set_initial_location (3,3) a/b mult flip lr;
```

Package Pin and Pad Location

Generally, you are concerned with the mapping of signals (ports) to the pins of the selected package. However, you may want to control the allocation of signals to particular pads. This is accomplished by assigning ports to the pad location rather than to the package pin. Because all pads are pre-bonded to package pins, the effect is to assign ports to package pins, with the emphasis on pad location rather than package pin.

Pad location is described by the letters N (North), S (South), E (East) or W (West) followed by a space and a number. This location code determines the direction and offset of the pad with respect to the die.

The top edge of the viewer contains the North pads and the right edge contains the East pads. The number refers to the pad position along its edge. For example, N 48 corresponds to the 48th pad along the North edge of the die. The figure below shows the numbering system used for pad location.



net_critical_ports

Use this statement to specify a specific subset of critical ports on a net.

For example, the following statement identifies two inputs of the net /u1/u2/net1 that are more critical than all other connections on that net. All other connections on the net will be buffered with a BUF cell that will be placed in a tile to reduce fanout delay on the specified inputs:

```
net_critical_ports /u1/u2/net1 nandbk1.A sigproc.C;
```

set_critical

Use this statement to specify critical nets and their relative criticality over other critical nets.

```
set_critical criticality_number hier_net_name
[,hier_net_name …];
```

Where criticality_number is from 1 to 5 (1 being the default criticality for every net and 5 the highest). `hier_net_name` is the full hierarchical net name.

For example, the statements below set the timing of u1/u2/ net1 more critical than u1/u2/net5 and u1/u2/net3:

```
set_critical 5 /u1/u2/net1;
set_critical 2 /u1/u2/net5, u1/u2/net3;
```

set_critical_port

Use this statement to identify design I/O ports that have above-normal criticality. The criticality number scale is the same for the set_critical statement.

```
set_critical_port criticality_number signal_name
[,signal_name …];
```

Where `signal_name` is the name of a user-defined signal associated with a specific I/O pin on the part.

For example, the following statement sets all nets associated with device ports IOBus[3] and IOBus[5] to have criticality 3:

```
set_critical_port 3 IOBus[3], IOBus[5];
```

set_empty_io

Use this statement to specify a location in which no I/O pin should be placed. The location can be specified by side and offset or by name.

```
set_empty_io { package_pin | pad_location};
```

For example, the following statement forces pin B5 and the pin associated with the fourth tile on the North side to be empty:

```
set_empty_io B5, (N,4);
```

set_empty_location

Use this statement to specify a location in which no cell should be placed.

```
set_empty_location ( x ,y);
```

```
set_empty_location (xbl ,ybl xtr ,ytr );
```

Where x , y (required) are the (x, y) tile coordinates that specify the empty cell location and x bl , y bl x tr , y tr (required) are the x, y tile coordinates for the bottom left and top right corner of the region.

Note: Only white spaces are allowed between the coordinates.

For example, the following statement informs the placement program that location (3, 7) is unavailable for cell placement:

```
set_empty_location( 3 ,7);
```

```
set_empty_location(113 ,1 60 ,80 );
```

set_initial_io

Use this statement to initially assign package pins to I/O ports or locate I/O ports at a specified side of a device. The placer can reassign or relocate the cells during place-and-route.

```
set_initial_io { package_pin | pad_location} io_port_name
```

```
[, io_port_name , … ];
```

Where package_pin is a package pin number for a specified I/O cell.

If you use package_pin, only one io_port_name argument is allowed (required if no pin location is given). pad_location is one of N, S, E, or W, followed by a pad location number on the chip. It constrains the pin location of a specified I/O cell to a specific pad location on the chip. Only one io_port_name argument is allowed. io_port_name (required) is the name of an I/O port to be assigned to a package pin or located at a specified edge of a package.

The following example statement initially places the I/O associated with net in3 to package pin A11:

```
set_initial_io A11 in3;
```

The following example statement initially places the I/O associated with net in4 on the fourth tile on the

North side:

```
set_initial_io (N,4) in4;
```

```
set_initial_io to a side is missing.
```

```
For example,
```

```
set_initial_io   S  in5;   // assigns in5 to the South side
```

Multiple comma-separated ports can be specified when they are assigned to a side.

set_initial_location

Use this statement to initially locate a cell instance at specified x, y coordinates. The placer can relocate the cell instance during place-and-route.

```
set_initial_location ( x, y ) hier_inst_name ;
```

Where x , y (required) are the x, y tile coordinates for the location of a specified cell instance and hier_ins*t_nam*e (required) is the hierarchical path to a cell instance.

For example:

```
set_initial_location (43,105) bk3/fp5/nand3_4;
```

set_io

Use this statement to either assign package pins to I/O ports or locate I/O ports at a specified side or location of a device. This constraint is a hard constraint and can not be overruled by the placer. This may have an impact on the timing results of a design. If a hard constraint is not suitable, use the set_initial_io constraint.

```
set_io {..|..} netName/portName;
```

For example:

```
set_io A9 in1;
set_io (S,22) in2;
set_io  N  in3;     // assigns in3 to the North side
```

Multiple comma-separated ports can be specified when they are assigned to a side.

set_io_region

This constraint enables you to place specific I/O instances into a target rectangular region. The global I/Os are excluded from this constraint.

The syntax is:

```
set_io_region (x1, y1  x2, y2) p1 [, p2, p3, .... , pn] ;"
```

where

`x1, y1  x2, y2` are the lower left and upper right corners of the rectangle that define the region

`p1, ... , pn` is a list of one or more I/O instance names or ports.

If multiple instances or ports are listed, they must be separated by commas. For example:

```
set_io_region (0,41 0,48) "acc[3]", "acc[4]";
```

set_location

Use this statement to locate a cell instance at specified x,y coordinates. The placer cannot relocate the cell instance during place-and-route.

`set_location ( x,y )` *hier_inst_name* ;

`set_location (x bl ,y bl x tr ,y tr )` *hier_inst_name/*;*

Where x , y (required) are the (x, y) tile coordinates that specify the empty cell location and x bl , y bl x tr , y tr (required) are the x, y tile coordinates for the bottom left and top right corner of the region.

For example:

```
set_location (1,15) u4/u3/nand3_4;
```

```
set_location (1,1 32,32) datapath/*;
```

This statement has been extended to allow you to place a sub-design instance by calling its macro and then applying a translation and rotation. The syntax is:

```
set_location (x, y) hier_subdesign_inst_name macro_name
```

```
[transformations];
```

where

`hier_subdesign_inst_name` is the hierarchical name of the instance of the sub-design; `(x, y)` is the final location of the lower left corner of the macro after all transformations have been completed; `macro_name` is the name of the previously defined macro; `transformations` are optional, and any of the following in any order:

- `flip lr` - flip cell from left to right
- `flip ud` - flip cell from up to down
- `rotate 90 cw` - rotate 90 ° clockwise
- `rotate 180 cw` - rotate 180 ° clockwise
- `rotate 270 cw` - rotate 270 ° clockwise
- `rotate 90 ccw` - rotate 90 ° counter-clockwise
- `rotate 180 ccw` - rotate 180 ° counter-clockwise

set_memory_region

Use this statement to create a region and assign memory to it. You can only assign names of memory macros to the region. Do not specify names of individual tiles.

For cascaded memory, the set_memory_region constraint applies to the whole cascaded block, even if your statement mentions only one macro out of the whole cascaded block.

Syntax

```
set_memory_region (x1,y1 x2,y2) memory1_name [,...,memoryn_name];
```

Arguments

```
(x1,y1 x2,y2)
```

Where x1, y1 specifies the bottom-left corner and x2,y2 specifies the top-right corner of the rectangle that defines the region in which the memory macros will be assigned. The macros are constrained to this region.

```
memory1_name,...
```

Specifies the memory macro(s) to assign to the region. Macro names are hierarchical names in the user netlist.

You can use wildcards in macro names. The wildcard character (*) matches any string.

Examples

```
set_memory_region (1,101 32,101) M1/U0;
```

```
set_memory_region (1,101 48,101) M1/U0,M1/U1;
set_memory_region (1,101 128,101) M1/U*;
```

Note: You can also use set_net_region and use_global to assign memory to regions.

Additionally, you can use the MultiView Navigator (MVN) to create regions that include memory. MVN regions can span core, I/O, and/or memory.

set_net_region

This GCF constraint enables you to put all the connected instances, driver, and all the driven instances for the net(s) into the target rectangle specified in the constraint. It puts the region constraint on all the connected instances, which will be processed by the placer. You can use wildcards in the net name.

The RAMs and I/Os are assigned to the LocalClock region unless the Compile option “Include RAM and I/O in Spine and Net Regions” is cleared. For designs created with v5.1 or earlier, this option is cleared by default. See "Compile Options" in the online help for more information.

```
set_net_region (x1,y2 x2,y2) <net_name_wildcard>;
```

Note: Only white spaces are allowed between the coordinates.

create_clock

Use this statement to define clocks for the design. Multiple clocks can be specified for a given design.

```
create_clock -period <period_value> {netname|portname}
```

Where period_value is the clock period in nanoseconds and netname|portname is the name of the net through which the clocks gets propagated or name of the external port.

For example, the following statement creates a clock on external port clk with a period of 25.0 nanoseconds.

```
create_clock -period 25.0 clk;
```

generate_paths

Use this statement to modify the way Designer generates internal path constraints for the placer to do timing-driven placement.

```
generate_paths [-cover_design] [-max_paths <maxpaths>
[-top <percentage>];
```

Where -cover_design indicates to Designer to use the cover design algorithm instead of the default worst paths algorithm, -max_paths is the maximum number of paths that will be generated (default is 20% of the number of nets with minimum of 1000 or if cover_design is specified, twice the number of nets with a minimum of 1000), -top indicates the top percentage of worst paths that will be generated (default is 20%).

The cover design algorithm ensures that the generated set of paths span the entire design; every pin of every instance participates in at least one path.

After the timing analysis is done, the worst paths algorithm starts recording the signal paths starting from the worst slack. It stops after some number of paths is generated; as a result, only some part of the design may be covered by the paths.

For example, the following statement generates 4,000 maximum paths using the -cover_design algorithm.

```
generate_paths -cover_design -max_paths 4000;
```

set_false_path

Use this statement to define false paths in the design. These paths are not considered in the timing driven place-and-route system.

```
set_false_path [-from from_port] [-through any_port] [-to to_port];
```

Where from_port must be an input port of the design or a register or memory instance output pin, to_port must be an output port of the design or a register or memory instance input pin, any_port must be any instance pin. Wildcards are permitted.

For example, the following statement sets all paths starting from resetd which are going through instance const2 as false paths.

```
set_false_path -from resetd -through const2/*;
```

set_input_to_register_delay

Use this statement to define the timing budget for incoming signals to reach a register.

```
set_input_to_register_delay <delay> [-from inp_port];
```

Where delay is the timing budget for this input path, inp_port is a register or memory instance output pin. Wildcards are permitted.

For example, the following statement specifies that the timing budget is 22 nanoseconds to the register from all inputs who's names start with the letter "I".

```
set_input_to_register_delay 22 -from I*;
```

set_max_path_delay

Use this statement to constrain the maximum delay on paths. The calculate timing task will report a note in the timing report file if this delay is not met.

```
set_max_path_delay delay_value
hier_inst_name_.inst_port_name
[,hier_inst_name .inst_port_name , … ];
```

Where delay_value is a floating integer for delay in nanoseconds, hier_inst_name is the hierarchical path to a cell instance, and inst_port_name is a port name of a cell instance.

For example:

```
set_max_path_delay 12.5 "mult4/mult/nand2_2".Y, "mult4/mult/
nand3_1".A,"mult4/mult/nand3_1".Y,"mult4/mult/nor2_2".A;
```

set_multicycle_path

Use this constraint to define how many clock cycles a signal has to travel through these paths. The budget of these paths will be a multiple of the period of the clock controlling the from port.

```
set_multicycle_path <num_cycles> -from reg_port [-through_any_port] [-to_port];
```

Where num_cycles is the number of clock cycles in which the signal needs to propagate through the path, reg_port is a register of memory instance, to_port must be an output port of the design or a register or memory instance input pin, any_port must be any instance pin. Wildcards are permitted.

For example, the following statement specifies it takes two clock cycles to reach signals from instance pins /us/u1/dff*.q to instance pins /u4/ mem1/*.D.

```
set_multicycle_path 2 -from /us/u1/dff*.q -to /u4/mem1/*.D;
```

set_register_to_output_delay

Use this statement to define the timing budget for outgoing signals to be clocked out.

```
set_register_to_output_delay <delay> -to out_port;
```

Where delay is the timing budget for this output path, out_port must be an output port of the design. Wildcards are permitted.

For example, the following statement specifies the timing budget for clocking out signals on output ports starting with “O” is 22 nanoseconds.

```
set_register_to_output_delay 22 -to O*;
```

About Physical Design Constraint (PDC) Files

A PDC file is a Tcl script file specifying physical constraints. This file can be imported and exported from Designer. Any constraint that you can enter using the PinEditor in MVN or ChipPlanner tool, you can also use in a PDC file.

Note: Only ProASIC3/E, and Axcelerator devices support PDC files.

Designer supports the following PDC commands.

Command	Action
assign_global_clock	Assigns user-defined nets to global clock networks by promoting the net using a CLKINT macro
assign_local_clock	Assigns user-defined nets to local clock routing (Axcelerator) or to either LocalClock or QuadrantClock regions (ProASIC3/E)
assign_net_macros	Assigns the macros connected to a net to a specified defined region
assign_region	Assigns macros to a pre-specified region
define_region	Defines a rectilinear region
define_region	Defines a rectangular region
delete_buffer_tree	Removes all buffers and inverters from a given net for ProASIC3 and ProASIC3E devices
dont_touch_buffer_tree	Restores all buffers and inverters that were removed from a given net with the delete_buffer_tree command
move_region	Moves a region to new coordinates
reset_floorplan	Deletes all defined regions. Placed macros are not affected
reset_io	Resets all attributes on a macro to the default values
reset_iobank	Resets an I/O banks technology to the default technology
reset_net_critical	Resets net criticality to default level
set_io	Sets the attributes of an I/O
set_iobank	Specifies the I/O bank’s technology
set_location	Places a given logic instance at a particular location
set_multitile_location	Assigns specified two-tile and four-tile macros to specified locations on the chip
set_net_critical	Sets net criticality, which is issued to influence placement and routing in favor of performance
set_vref	Specifies which pins are VREF pins
set_vref_defaults	Sets the default VREF pins for specified banks

unassign_global_clock	Assigns clock nets to regular nets
unassign_local_clock	Unassigns the specified user-defined net from a LocalClock or QuadrantClock region
unassign_macro_from_region	Unassigns macros from a specified region, if they are assigned to that region
unassign_net_macro	Unassigns macros connected to a specified net from a defined region
undefine_region	Removes the specified region

Note: PDC commands are case sensitive. However, their arguments are not.

See Also

About design constraints

Exporting files

Importing PDC files

Importing auxiliary files

PDC syntax conventions

PDC naming conventions

PDC Syntax Conventions

The following table shows the typographical conventions that are used for the PDC command syntax.

Syntax Notation	Description
`command -argument`	Commands and arguments appear in `Courier New` typeface.
`variable`	Variables appear in blue, italic `Courier New` typeface. You must substitute an appropriate value for the variable.
`[-argument` *`value`*`]` `[`*`variable`*`]+`	Optional arguments begin and end with a square bracket with one exception: if the square bracket is followed by a plus sign (+), then users must specify at least one argument. The plus sign (+) indicates that items within the square brackets can be repeated. Do not enter the plus sign character.

Note: PDC commands are case sensitive. However, their arguments are not.

Examples

Syntax for the assign_local_clock (Axcelerator) command followed by a sample command:

```
assign_local_clock -type value -net netname [LocalClock_region]+
```

```
assign_local_clock -type hclk -net  reset_n tile1a tile2a
```

Syntax for the set_io (Axcelerator) command followed by a sample command:

```
set_io portname [-iostd value][-register value][-out_drive value][-slew value][-
res_pull value][-out_load value][-pinname value][-fixed value][-in_delay value]
```

```
set_io ADDOUT2 \
   -iostd PCI \
```

```
   -register yes \
   -out_drive 16 \
   -slew high \
   -out_load 10 \
   -pinname T21 \
   -fixed yes
```

Wildcard Characters

You can use the following wildcard characters in names used in PDC commands:

Wildcard	What it does
\	Interprets the next character literally
?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Note: The matching function requires that you add a slash (\) before each slash in the port, instance, or net name when using wildcards in a PDC command and when using wildcards in the Find feature of the MultiView Navigator. For example, if you have an instance named "A/B12" in the netlist, and you enter that name as "A\\/B*" in a PDC command, you will not be able to find it. In this case, you must specify the name as A\\\\/B*.

Special Characters ([ ], { }, and \)

Sometimes square brackets ([ ]) are part of the command syntax. In these cases, you must either enclose the open and closed square brackets characters with curly brackets ({ }) or precede the open and closed square brackets ([ ]) characters with a backslash (\). If you do not, you will get an error message.

For example:

```
set_iobank {mem_data_in[57]} -fixed no 7 2
```

or

```
set_iobank mem_data_in\[57\] -fixed no 7 2
```

Entering Arguments on Separate Lines

To enter an argument on a separate line, you must enter a backslash (\) character at the end of the preceding line of the command as shown in the following example:

```
set_io ADDOUT2 \
   -iostd PCI \
   -register Yes \
   -out_drive 16 \
   -slew High \
   -out_load 10 \
   -pinname T21 \
   -fixed Yes
```

See Also

About PDC files

PDC naming conventions

PDC Naming Conventions

Note: The names of ports, instances, and nets in an imported netlist are sometimes referred to as their *original names*.

Rules for displaying original names

Port names appear exactly as they are defined in a netlist. For example, a port named A/B appears as A/B in ChipPlanner, PinEditor, and I/O Attribute Editor in MultiView Navigator.

Instances and nets display the original names plus an escape character (\) before each backslash (/) and each slash (\) that is not a hierarchy separator. For example, the instance named `A/\B` is displayed as `A\/\\B`.

Which name do I use in PDC commands?

The names of ports, instances, and nets in a netlist displayed in MultiView Navigator (MVN) for ProASIC3/E devices are names taken directly from the imported netlist. However, the names displayed in MVN for Axcelerator devices are the compiled names. Compiled names remove special characters such as slash (\) and backslash (/) characters and replace them with the underscore character.

Using PDC Commands for Axcelerator

When importing PDC commands for Axcelerator devices, always use the post-compiled names of ports, instances, and nets. You can see the post-compiled names in MultiView Navigator or the compile report.

Using PDC Commands for ProASIC3/E

When writing PDC commands for ProASIC3/E devices, follow these rules:

Always use the macro name as it appears in the netlist. (See "Merged elements" in this topic for exceptions.)

Names from a netlist: For port names, use the names exactly as they appear in the netlist. For instance and net names, add an escape character (\) before each backslash (\) and each slash (/) that is not a hierarchy separator.

Names from MVN and compile report: Use names as they appear in MultiView Navigator or the compile report.

For wildcard names, always add an extra backslash (\) before each backslash.

Always apply the PDC syntax conventions to any name in a PDC command.

The following table provides examples of names as they appear in an imported netlist and the names as they should appear in a PDC file:

Type of name and its location	Name in the imported netlist	Name to use in PDC file
Port name in netlist	A/:B1	A/:B1
Port name in MVN	A/:B1	A/:B1
Instance name in a netlist	A/:B1 A$(1)	A\\/:B1 A$(1)
Instance name in the netlist but using a wildcard character in a PDC file	A/:B1	A\\\\/:B*
Instance name in MVN or a compile report	A\/:B1	A\\/:B1

Net name in a netlist	Net1/:net1	Net1\\/:net1
Net name in MVN or a compile report	Net1\/:net1	Net1\\/:net1

When exporting PDC commands, the software always exports names using the PDC rules described in this topic.

Case sensitivity when importing PDC files (ProASIC3/E only)

The following table shows the case sensitivity in the PDC file based on the source netlist.

File type	Case sensitivity
Verilog	Names in the netlist are case sensitive.
Edif	Names in the netlist are always case sensitive because we use the Rename clause, which is case sensitive.
Vhdl	Names in the netlist are not case sensitive unless those names appear between slashes (\).

For example, in VHDL, capital "A" and lowercase "a" are the same name, but \A\ and \a\ are two different names. However, in a Verilog netlist, an instance named "A10" will fail if spelled as "a10" in the set_location command:

```
set_location A10  (This command will succeed.)
set_location a10  (This command will fail.)
```

Which name to use in the case of merged elements (ProASIC3/E only)

The following table indicates which name to use in a PDC command when performing the specified operation:

Operation	Name to use
I/O connected to PLL with a hardwired connection	PLL instance name
I/O combined with FF or DDR	I/O instance name
Global promotion	<Driver instance of the net>_CLKINT

See Also

About PDC files

PDC syntax conventions

assign_global_clock

Assigns user-defined nets to global clock networks by promoting the net using a CLKINT macro.

```
assign_global_clock -net netname
```

Arguments

`-net netname`

Specifies the name of the net to promote to a global clock network. The net is promoted using a CLKINT macro, which you can place on a chip-wide clock location.

Supported Families

ProASIC3/E

Notes

The assign_global_clock command is not supported in auxiliary PDC files.

Exceptions

None

Examples

```
assign_global_clock -net globalReset
```

See Also

`assign_local_clock (Axcelerator)`

`assign_local_clock (ProASIC3/E)`

unassign_global_clock

PDC syntax conventions

PDC naming conventions

assign_local_clock

The syntax and arguments for the **assign_local_clock** command vary depending on which device you are designing. Click the device name in the following list to see the syntax, arguments, and other information for the assign_local_clock command for that particular device:

assign_local_clock (ProASIC3/E)

assign_local_clock (Axcelerator)

assign_net_macros

Assigns to a user-defined region all the macros that are connected to a net.

```
assign_net_macros region_name [net1]+
```

Arguments

`region_name`

Specifies the name of the region to which you are assigning macros. The region must exist before you use this command. See define_region (rectangular) or define_region (rectilinear). Because the define_region command returns a region object, you can write a simple command such as `assign_net_macros [define_region]+ [net]+`

`net1`

You must specify at least one net name. Net names are AFL-level (Actel flattened netlist) names. These names match your netlist names most of the time. When they do not, you must export AFL and use the AFL names. Net names are case insensitive. Hierarchical net names from ADL are not allowed. You can use the following wildcard characters in net names:

Wildcard	What it does
\	Interprets the next character as a non-special character
?	Matches any single character
*	Matches any string

[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Supported Families

ProASIC3/E and Axcelerator

Notes

Placed macros (not connected to the net) that are inside the area occupied by the net region are automatically unplaced.

Net region constraints are internally converted into constraints on macros. PDC export results as a series of `assign_region <region_name> macro1` statements for all the connected macros.

Net region constraints may result in a single macro being assigned to multiple regions. These net region constraints result in constraining the macro to the intersection of all the regions affected by the constraint.

Exceptions

If the region does not have enough space for all of the macros, or if the region constraint is impossible, the constraint is rejected and a warning message appears in the Log window.

For overlapping regions, the intersection must be at least as big as the overlapping macro count.

If a macro on the net cannot legally be placed in the region, it is not placed and a warning message appears in the Log window.

Example

```
assign_net_macros cluster_region1 keyin1intZ0Z_62
```

See Also

unassign_net_macro

PDC syntax conventions

PDC naming conventions

assign_region

Constrains a set of macros to a specified region.

```
assign_region region [macro_name]+
```

Arguments

`region`

Specifies the region to which the macros are assigned. The macros are constrained to this region. Because the `define_region` command returns a region object, you can write a simpler command such as `assign_region [define_region]+ [macro_name]+`

`macro_name`

Specifies the macro(s) to assign to the region. You must specify at least one macro name. You can use the following wildcard characters in macro names:

Wildcard	What it does
\	Interprets the next character as a non-special character

?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Supported Families

ProASIC3/E and Axcelerator

Notes

The region must be created before you can assign macros to it.

Exceptions

You can assign only hard macros or their instances to a region. You cannot assign a group name. A hard macro is a logic cell consisting of one or more silicon modules with locked relative placement.

Examples

In the following example, two macros are assigned to a region:

```
assign_region cluster_region1 des01/G_2722_0_and2 des01/data1_53/U0
```

See Also

unassign_macro_from_region

PDC syntax conventions

PDC naming conventions

define_region (rectangular region)

Defines either a rectangular region or a rectilinear region.

```
define_region [-name region_name] -type region_type [x1 y1 x2 y2]+
```

Arguments

`-name region_name`

Specifies the region name. The name must be unique. Do not use reserved names such as “bank0” and “bank<N>” for region names. If the region cannot be created, the name is empty. A default name is generated if a name is not specified in this argument.

`-type region_type`

Specifies the region type. The default is inclusive.

Region Constraint	Conditions
Empty	No macros can be assigned to an empty region
Exclusive	Only contains macros assigned to the region
Inclusive	Can contain macros both assigned and unassigned to the region

`x1 y1 x2 y2`

Specifies the series of coordinate pairs that constitute the region. These rectangles may or may not overlap. They are given as x1 y1 x2 y2 (where x1, y1 is the lower left and x2 y2 is the upper right corner in row/column coordinates). You must specify at least one set of coordinates.

Supported Families

ProASIC3/E and Axcelerator

Notes

Unlocked macros in empty or exclusive regions are unassigned from that region.

You cannot create empty or exclusive regions in areas that contain locked macros.

Use inclusive or exclusive region constraints if you intend to assign logic to a region. An inclusive region constraint with no macros assigned to it has no effect. An exclusive region constraint with no macros assigned to it is equivalent to an empty region.

Exceptions

If macros assigned to a region exceed the area's capacity, an error message appears in the Log window.

Examples

The following example defines an empty rectangular region.

```
define_region -name cluster_region1 -type empty 100 46 102 46
```

See Also

define_region (rectilinear region)

PDC syntax conventions

PDC naming conventions

define_region (rectilinear region)

Defines either a rectilinear region or a rectangular region.

```
define_region [-name region_name] -type region_type [x1 y1 x2 y2]+
```

Arguments

`-name region_name`

Specifies the region name. The name must be unique. Do not use reserved names such as “bank0” and “bank<N>” for region names. If the region cannot be created, the name is empty. A default name is generated if a name is not specified in this argument.

`-type region_type`

Specifies the region type. The default is inclusive.

Region Constraint	Conditions
Empty	No macros can be assigned to an empty region
Exclusive	Only contains macros assigned to the region
Inclusive	Can contain macros both assigned and unassigned to the region

x1 y1 x2 y2

Specifies the series of coordinate pairs that constitute the region. These rectangles may or may not overlap. They are given as x1 y1 x2 y2 (where x1, y1 is the lower left and x2 y2 is the upper right corner in row/column coordinates). You must specify at least one set of coordinates.

Supported Families

ProASIC3/E and Axcelerator

Notes

Unlocked macros in empty or exclusive regions are unassigned from that region.

You cannot create empty or exclusive regions in areas that contain locked macros.

Use inclusive or exclusive region constraints if you intend to assign logic to a region. An inclusive region constraint with no macros assigned to it has no effect. An exclusive region constraint with no macros assigned to it is equivalent to an empty region.

You can define a rectilinear region only in a PDC file; you cannot define a rectilinear region using the MultiView Navigator tool.

Exceptions

If macros assigned to a region exceed the area's capacity, an error message appears in the log window.

Examples

The following example defines a region with the name RecRegion. This region contains two rectangular areas.

```
define_region -name RecRegion -type Exclusive 0 40 3 42 0 77 7 79
```

See Also

define_region (rectangular region)

PDC syntax conventions

PDC naming conventions

delete_buffer_tree

Instructs the Compile command to remove all buffers and inverters from a given net.

```
delete_buffer_tree -net [netname]+
```

Arguments

-net netname

Specifies the names of the nets from which to remove buffer or inverter trees. This command takes a list of names. You must specify at least one net name. You can use the following wildcard characters in net names:

Wildcard	What it does
\	Interprets the next character as a non-special character
?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Supported Families

ProASIC3/E

Notes

In the ProASIC3E and ProASIC3 architectures, inverters are considered buffers because all tile inputs can be inverted. This rule is also true for all Flash architectures but not for Antifuse architectures.

Exceptions

The delete_buffer_tree command is not supported in auxiliary PDC files.

Examples

```
delete_buffer_tree net1
```

```
delete_buffer_tree netData\[*\]
```

See Also

don't_touch_buffer_tree

PDC syntax conventions

PDC naming conventions

dont_touch_buffer_tree

Undoes the delete_buffer_tree command. That is, it restores all buffers and inverters that were removed from a given net.

```
dont_touch_buffer_tree -net netname
```

Arguments

`-net netname`

Specifies the name of the net with the buffers and inverters to restore.

Supported Families

ProASIC3/E

Notes

The dont_touch_buffer_tree command is not supported in auxiliary PDC files.

Exceptions

None

Examples

This example prevents the software from removing buffer trees on the reset_n net:

```
dont_touch_buffer_tree -net reset_n
```

See Also

delete_buffer_tree

PDC syntax conventions

PDC naming conventions

move_region

Moves the named region to the coordinates specified.

```
move_region region_name [x1 y1 x2 y2]+
```

Arguments

`region_name`

Specifies the name of the region to move. This name must be unique.

`x1 y1 x2 y2`

Specifies the series of coordinate pairs representing the location in which to move the named region. These rectangles can overlap. They are given as x1 y1 x2 y2, where x1, y1 represents the lower-left corner of the rectangle and x2 y2 represents the upper-right corner. You must specify at least one set of coordinates.

Supported Families

ProASIC3/E and Axcelerator

Notes

None

Exceptions

None

Examples

This example moves the region named RecRegion to a new region which is made up of two rectangular areas:

```
move_region RecRegion 0 40 3 42 0 77 7 79
```

See Also

PDC syntax conventions

PDC naming conventions

reset_floorplan

Deletes all regions.

```
reset_floorplan
```

Arguments

None

Supported Families

`Axcelerator`

Notes

None

Exceptions

None

Examples

```
reset_floorplan
```

See Also

PDC syntax conventions

PDC naming conventions

reset_io

Restores all attributes of an I/O macro to its default values. Also, if the port is assigned, it will become unassigned.

```
reset_io portname -attributes value
```

Arguments

`portname`

Specifies the port name of the I/O macro to be reset. You can use the following wildcard characters in port names:

Wildcard	What it does
\	Interprets the next character as a non-special character
?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

`-attributes value`

Preserve or not preserve the I/O attributes during incremental flow. The following table shows the acceptable values for this argument:

Value	Description
yes	Unassigns all of the I/O attributes and resets them to their default values.
no	Unassigns only the port.

Supported Families

`ProASIC3/E and Axcelerator`

Notes

None

Exceptions

None

Examples

```
reset_io a
```

Resets the I/O macro “a” to the default I/O attributes and unassigns it.

```
reset_io b_*
```

Resets all I/O macros beginning with "b_" to the default I/O attributes and unassigns them.

```
reset_io b -attributes no
```

Only unassigns port b from its location.

See Also

set_io

PDC syntax conventions

PDC naming conventions

reset_iobank

Resets an I/O bank’s technology to the default technology, which is specified using the Designer software in the Device Selection Wizard.

```
reset_iobank bankname
```

Arguments

`bankname`

Specifies the I/O bank to be reset to the default technology. For example, for ProASIC3E and Axcelerator devices, I/O banks are numbered 0-7 (bank0, bank1,.. bank7).

Supported Families

`ProASIC3/E and Axcelerator`

Notes

Any pins that are assigned to the specified I/O bank but are incompatible with the default technology are unassigned.

Exceptions

None

Examples

The following example resets I/O bank 4 to the default technology:

```
reset_iobank bank4
```

See Also

set_iobank

PDC syntax conventions

PDC naming conventions

reset_net_critical

Resets the critical value to its default. Net criticality can vary from 1 to 10, with 1 being the least critical and 10 being the most. The default is 5. Criticality numbers are used in timing driven place-and-route.

Increasing a net's criticality forces place-and-route to keep instances connected to the net as close as possible, at the cost of other (less critical) nets.

```
reset_net_critical [netname]+
```

Arguments

netname

Specifies the name of the net to be reset to the default critical value. You must specify at least one net name. You can use the following wildcard characters in net names:

Wildcard	What it does
\	Interprets the next character as a non-special character
?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Supported Families

Axcelerator

Notes

None

Exceptions

None

Examples

This example resets the net preset_a to the default ciricality of 5:

```
reset_net_critical preset_A
```

See Also

set_net_critical

PDC syntax conventions

PDC naming conventions

set_io

The syntax and arguments for the **set_io** command vary depending on which device you are designing. Click the command next to the device name in the following list to see the syntax, arguments, and other information for the set_io command for that particular device:

set_io (ProASIC3E)

set_io (ProASIC3)

set_io (Axcelerator)

---

set_iobank

The syntax and arguments for the **set_iobank** command vary depending on which device you are designing. Click the command next to the device name in the following list to see the syntax, arguments, and other information for the set_io command for that particular device:

set_iobank (ProASIC3E)

set_iobank (ProASIC3)

set_iobank (Axcelerator)

---

set_location

Assigns the specified macro to a particular location on the chip.

```
set_location macro_name -fixed value x y
```

Arguments

`macro_name`

Specifies the name of the macro in the netlist to assign to a particular location on the chip.

`-fixed value`

Sets whether the location of this instance is fixed (that is, locked). Locked instances are not moved during layout. The default is yes. The following table shows the acceptable values for this argument:

Value	Description
yes	The location of this instance is locked.
no	The location of this instance is unlocked.

`x y`

The x and y coordinates specify where to place the macro on the chip. Use the ChipPlanner tool to determine the x and y coordinates of the location.

Supported Families

`ProASIC3/E and Axcelerator`

Notes

Use the post-compiled macro name when specifying the macro name.

Exceptions

None

Examples

This example assigns and locks the macro with the name "mem_data_in\[57\]" at the location x=7, y=2:

```
set_iobank mem_data_in\[57\] -fixed no 7 2
```

See Also

set_multitile_location

PDC syntax conventions

PDC naming conventions

set_multitile_location (ProASIC3/E)

Assigns specified two-tile and four-tile macros to specified locations on the chip. Use this command only for multi-tile, flip-flop macros and, in some cases, enable flip-flop macros).

```
set_multitile_location macro_name [-fixed value]\
  -location {x y} \
  -tile {name1 relative_x1 relative_y1} \
  -tile {name2 relative_x2 relative_y2} \
  [-tile {name3 relative_x3 relative_y3} \ ]
  [-tile {name4 relative_x4 relative_y4} \ ]
```

Arguments

`macro_name`

Specifies the hierarchical name of the macro in the netlist to assign to a particular location on the chip.

`-fixed value`

Sets whether the location of this set of macros is fixed (that is, locked). Locked macros are not moved during layout. The default is yes. The following table shows the acceptable values for this argument:

Value	Description
yes	The location of this instance is locked.
no	The location of this instance is unlocked.

`-location {x y}`

The x and y coordinates specify the absolute placement of the macro on the chip. You can use the ChipPlanner tool to determine the x and y coordinates of the location.

`-tile {name1 relative_x1 relative_y1}`

Specifies the hierarchical name and location, relative to the macro specified as the *macro_name*, of the first tile in a two- or four-tile macro. The relative placement of macro *name1* inside the macro cannot be offset by more than one. (See Notes below for placement rules.) If the macro uses four-tile macros, then you must define all four tiles. Likewise, if the macro uses two-tile macros, you must define both tiles.

You can place the following two-tile and four-tile macros with the set_multitile_location command:

Four-tile macro	DFN1P1C1	DFI1P1C1	DFN0P1C1	DFI0P1C1
Two-tile macro	DLN1P1C1	DLI1P1C1	DLN0P1C1	DLI0P1C1

Due to the ProASIC3/E architecture, if the CLR and PRE pins are NOT driven by a clock net (global, quadrant or local clock net), the enable flip-flop macros (shown below) are mapped to two-tile flip-flop macros. When CLR and PRE pins are not driven by a clock net, you must use the set_multitile_location command instead of the set_location command.

DFN1E1C0 DFN0E1C1 DFN1E0P1 DFN0E0P0 DFI1E1C0 DFI0E1P1 DFI1E0P0	DFN0E1C0 DFN1E0C1 DFN0E0P1 DFI1E1C1 DFI0E1C0 DFI1E0P1 DFI0E0P0	DFN1E0C0 DFN0E0C1 DFN1E1P0 DFI0E1C1 DFI1E0C0 DFI0E0P1	DFN0E0C0 DFN1E1P1 DFN0E1P0 DFI1E0C1 DFI0E0C0 DFI1E1P0	DFN1E1C1 DFN0E1P1 DFN1E0P0 DFI0E0C1 DFI1E1P1 DFI0E1P0

During compile, Designer maps the specified enable flip-flop macro to a two-tiled macro.

If the CLR and PRE pins are driven by a clock net, Designer maps these macros to one tile during compile. In this case, you cannot use the set_multitile_location command to place them. Instead, you must use the set_location command.

Supported Families

ProASIC3/E

Notes

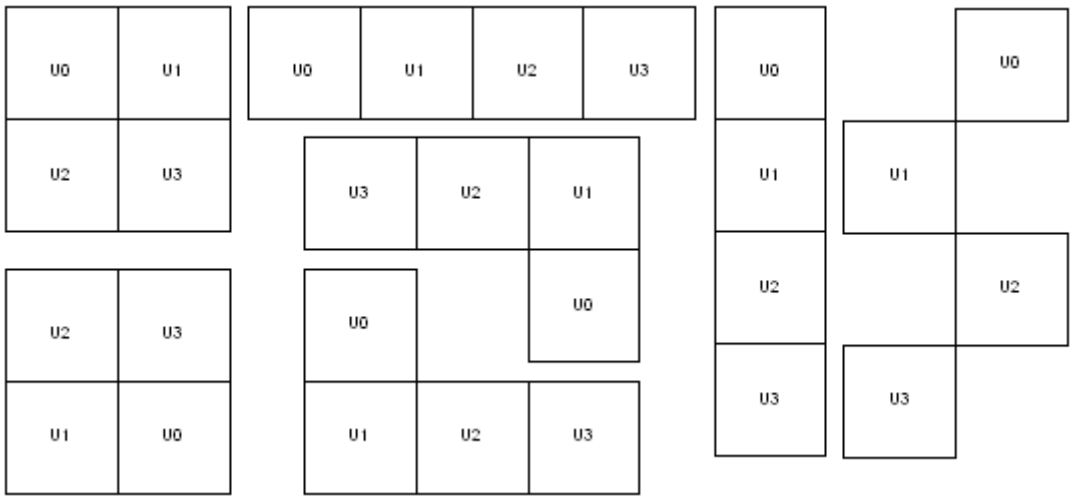
For two-tile flip-flop macros, the software appends U0 and U1 to the macro name. For four-tile flip-flop macros, the software appends U0, U1, U2 and U3 to the macro name.

The macros specified in the -tile option cannot be offset by more than one.

To ensure efficiency, you must use local connections between certain tiles in the macros. The distance between U0 and U1, U1 and U2, and U2 and U3 must not be more than one in either direction (X or Y). The required local connection between tiles is denoted by the dashes below:

Four-tile macros: U0 --- U1 --- U2 --- U3
Two-tile macros: U0 --- U1

Examples of possible placement configurations:



Exceptions

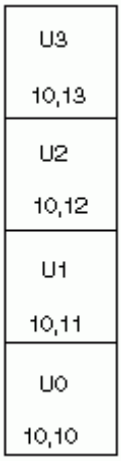
None

Examples

This example assigns and locks the macro with instance name “multi_tileff/U0 “ at the location X=10, Y=10 by specifying the relative positions of all the macros.

```
set_multitile_location multi_tileff -location {10 10} \
                       -tile { multi_tileff/U0 0 0 } \
                       -tile { multi_tileff/U1 0 1 } \
                       -tile { multi_tileff/U2 0 2 } \
                       -tile { multi_tileff/U3 0 3 } -fixed yes
```

As a result of this command, the four-tile macro placement looks like this:

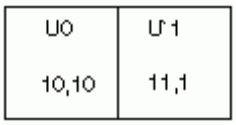


The second example shows you how to configure a two-tile macro:

```
set_multitile_location multi_tileff -location {10 10} \
```

```
-tile { multi_tileff/U0 0 0 } \
-tile { multi_tileff/U1 1 0 }
```

As a result of this command, the two-tile macro placement looks like this:



See Also

set_location

PDC syntax conventions

PDC naming conventions

set_net_critical

Sets the net criticality, which influences place-and-route in favor of performance.

```
set_net_critical criticality_number [hier_net_name]+
```

Arguments

`criticality_number`

Sets the criticality level from 1 to 10, with 1 being the least critical and 10 being the most critical. The default is 5. Criticality numbers are used in timing-driven place and route.

`hier_net_name`

Specifies the net name, which can be an AFL (Actel Flattened Netlist) net name or a net regular expression using wildcard characters. You must specify at least one net name. You can use the following wildcard characters in names:

Wildcard	What it does
\	Interprets the next character as a non-special character
?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Supported Families

`Axcelerator`

Notes

The command must have at least two parameters.

The net names are AFL names, which means they must be visible in Timer and ChipPlanner.

Increasing a net's criticality forces place-and-route to keep instances connected to the specified net as close as possible at the cost of other (less critical) nets.

Exceptions

None

Examples

This example sets the criticality level to 9 for all addr nets:

```
set_net_critical 9 addr*
```

See Also

reset_net_critical

PDC syntax conventions

PDC naming conventions

set_vref

Assigns a VREF pin to a bank. The ProASIC3E and Axcelerator families support VREF pins; however, the ProASIC3 family does not.

```
set_vref -bank [bankname][pinnum]+
```

Arguments

`-bank bankname`

Specifies the name of the bank in which the VREF pin is located. For example, for ProASIC3E and Axcelerator devices, I/O banks are numbered 0-7. Therefore, their bank names are bank0, bank1, bank2, etc.

`pinnum`

Specifies the alphanumeric pin name of the VREF pin(s). You must specify at least one pin number.

Supported Families

`ProASIC3E and Axcelerator`

Notes

While the bank name is optional, you must not specify pin names that do not belong to the bank. Pins that do not belong to a bank that require a VREF are ignored.

Some I/O technologies need VREF settings. Some technologies may also need a minimum number of VREF pins for every certain number of input pins. These details are device dependent. Refer to the databook for your device for details. Designer can assign default VREF pins. However, the number of VREF pins Designer assigns may be too conservative, and you may not need as many VREF pins as the default assignment.

Exceptions

None

Examples

The following example assigns pins A1 and B10 from I/O bank1 as VREF pins:

```
set_vref -bank bank1 A1 B10
```

See Also

set_vref_defaults

PDC syntax conventions

PDC naming conventions

set_vref_defaults

Sets the default VREF pins for the specified bank. This command is ignored if the bank does not need a VREF pin.

```
set_vref_defaults bankname
```

Arguments

`bankname`

Specifies the name of the bank in which the default VREF pin is located. For ProASIC3E and Axcelerator devices, banks are numbered from 0-7. Therefore, their bank names are bank0, bank1, bank2, etc.

Supported Families

`ProASIC3E and Axcelerator`

Notes

The ProASIC3 family does not support VREF pins.

Some I/O technologies need VREF settings. Some technologies may also need a minimum number of VREF pins for every certain number of input pins. These details are device dependent. Refer to the databook for your device for details. Designer can assign default VREF pins. However, the number of VREF pins Designer assigns may be too conservative, and you may not need as many VREF pins as the default assignment.

Exceptions

None

Examples

This example sets the default VREF pins for Bank 1:

```
set_vref_defaults bank1
```

See Also

set_vref

PDC syntax conventions

PDC naming conventions

unassign_global_clock

Demotes clock nets to regular nets.

```
unassign_global_clock -net netname
```

Arguments

`-net netname`

Specifies the name of the clock net to demote to a regular net.

Supported Families

ProASIC3/E

Notes

The unassign_global_clock command is not supported in auxiliary PDC files.

Exceptions

You cannot assign “essential” clock nets to regular nets. Clock nets that are driven by the following macros are “essential” global nets: CLKDLY, PLL, and CLKBIBUF.

Examples

```
unassign_global_clock -net globalReset
```

See Also

assign_global_clock

PDC syntax conventions

PDC naming conventions

unassign_local_clock

Unassigns the specified user-defined net from a LocalClock or QuadrantClock region.

```
unassign_local_clock -net netname
```

Arguments

`-net netname`

Specifies the name of the user-defined net to unassign.

Supported Families

`ProASIC3/E and Axcelerator`

Notes

You can create QuadrantClocks only for ProASIC3E and ProASIC3 devices.

The unassign_local_clock command is not supported in auxiliary PDC files.

Exceptions

None

Examples

```
unassign_local_clock -net reset_n
```

See Also

assign_local_clock (Axcelerator)

assign_local_clock (ProASIC3/E)

PDC syntax conventions

PDC naming conventions

unassign_macro_from_region

Specifies the name of the macro to be unassigned.

```
unassign_macro_from_region [region_name] macro_name
```

Arguments

region_name

Specifies the region where the macro or macros are to be removed.

macro_name

Specifies the macro to be unassigned from the region. Macro names are case sensitive. You cannot use hierarchical net names from ADL. However, you can use the following wildcard characters in macro names:

Wildcard	What it does
\	Interprets the next character as a non-special character
?	Matches any single character
*	Matches any string
[]	Matches any single character among those listed between brackets (that is, [A-Z] matches any single character in the A-to-Z range)

Supported Families

ProASIC3/E and Axcelerator

Notes

None

Exceptions

If the macro was not previously assigned, an error message is generated.

Examples

unassign_macro_from_region macro21

See Also

PDC syntax conventions

PDC naming conventions

unassign_net_macro

Unassigns macros connected to a specified net.

```
unassign_net_macro region_name [net1]+
```

Arguments

`region_name`

Specifies the name of the region containing the macros in the net(s) to unassign.

`net1`

Specifies the name of the net(s) that contain the macros to unassign from the specified region. You must specify at least one net name. Optionally, you can specify additional nets to unassign.

Supported Families

`ProASIC3/E and Axcelerator`

Notes

None

Exceptions

If the region is currently not assigned, an error message appears in the Log window if you try to unassign it.

Examples

```
unassign_net_macro cluster_region1  keyin1intZ0Z_62
```

See Also

assign_net_macros

PDC syntax conventions

PDC naming conventions

undefine_region

Removes the specified region.

```
undefine_region region_name
```

Arguments

`region_name`

Specifies the region to be removed.

Supported Families

`ProASIC3/E and Axcelerator`

Notes

All macros assigned to the region are unassigned.

Exceptions

To use this command, the region must have been previously defined.

Examples

```
undefine_region cluster_region1
```

See Also

define_region (rectangular region)

define_region (rectilinear region)

PDC syntax conventions

PDC naming conventions

Design Object Access Commands

Design object access commands are SDC commands. Most constraint commands require a command argument. Designer supports the SDC access commands shown below:

Design Object	Access Command
Clock	get_clocks
Port	get_ports
Input ports	all_inputs
Output ports	all_outputs
Pin	get_pins

get_clocks

Returns the named clock specified by an argument.

Syntax

```
get_clocks <pattern>
```

Argument

```
pattern
```

Specifies the pattern to match to the Timer potential clock names (port or pin names in the design).

If this command is used as a –from argument in max delay (set_max_path_delay), false path (set_false_path), and multicycle constraints (set_multicycle_path), the clock pins of all the registers related to this clock are used as path start points.

If this command is used as a –to argument in max delay (set_max_path_delay), false path (set_false_path), and multicycle constraints (set_multicycle_path), the synchronous pins of all the registers related to this clock are used as path endpoints.

Example

```
set_max_delay -from [get_ports data1] -to \
[get_clocks clk1]
```

get_pins

Returns the named pins specified by an argument.

Syntax

```
get_pins <pattern>
```

Argument

```
pattern
```

Specifies the pattern to match the pins.

Example

```
create_clock -period 10 [get_pins clock_gen/reg2:Q]
```

get_ports

Returns the named ports specified by an argument.

Syntax

```
get_ports <pattern>
```

Argument

```
pattern
```

Specifies the pattern to match the ports. This is equivalent to the macros $in()[<pattern>] when used as –from argument and $out()[<pattern>] when used as –to argument or $ports()[<pattern>] when used as a –through argument.

Example

```
create_clock -period 10 [get_ports CK1]
```

all_inputs

Returns all the input or inout ports of the design.

Syntax

```
all_inputs
```

Argument

None.

Example

```
set_max_delay -from [all_inputs] -to [get_clocks ck1]
```

all_outputs

Returns all the output or inout ports of the design.

Syntax

```
all_outputs
```

Argument

None.

Example

```
set_max_delay -from [all_inputs] -to [all_outputs]
```

create_clock (SDC clock Constraint)

The `create_clock` constraint is associated with a specific clock in a sequential design and determines the maximum register-to-register delay in the design.

Supported Families

SX-A, eX, RTAX-S, Axcelerator, RTAX-S, ProASIC^{PLUS} and ProASIC3/E

Syntax

```
create_clock -period period_value  [-waveform edge_list] source
```

Arguments

`period_value`

Mandatory argument specified in ns. No clock is created if the period is not supplied.

`edge_list`

This is optional and is supported as a duty cycle in the current version of Designer. If supplied, it must contain exactly two edges. The duty cycle info will then be added to the clock constraint.

`source`

Mandatory argument that specifies the source of the clock:

- a single design port name; for instance clk
- a single pin name; for instance reg1:CLK. This name can be hierarchical (for instance toplevel/block1/reg2:CLK)
- a list of one name in curly brackets (either port or pin); for instance: {A} or {toplevel/pll:clk1}
- an object accessor that will refer to one pin/port name for instance: [get_ports {clk}] or [get_pins reg:CLK]

Notes: No wildcard is accepted. The pin or port must exist in the design.

Valid Command Examples

```
create_clock -period 5 CK1
```

```
create_clock -period 4 -waveform {0 2} CK1
```

```
create_clock -period 6 [get_ports {CK1}]
```

Invalid Command Examples

```
create_clock -period 10
```

This command is invalid because there is no name supplied.

```
create_clock -period 3 [get_ports {CK1 CK2}]
```

This command is invalid because more than one name is used for the source.

```
create_clock -period 11 -waveform {0 2 5 7} CK1
```

This command is invalid because the supplied waveform has more than two edges.

```
create_clock CLK1 -period 20 -waveform 3 13
```

This command is invalid because this syntax of waveform is invalid.

set_false_path (SDC false path Constraint)

The `set_false_path` constraint identifies paths in the design that are to be marked as false, so that they are not considered during timing analysis.

Supported Families

SX-A, eX, RTSX-S, Axcelerator, RTAX-S, ProASIC[PLUS], and ProASIC3/E

Syntax

```
set_false_path -through through_point
```

Arguments

`through_point`

Specifies a pin or port through which the disabled paths muss pass.

a single design port name or pin name; for instance clk or toplevel/block1/inst1:A

a list of one name in curly brackets (either port or pin); for instance: {A} or {U0:D}

an object accessor that will refer to one name; for instance: [get_ports {d}] or [get_pins S:Y]

Valid Command Examples

```
set_false_path -through U0/U1:Y
```

```
set_false_path -through {Data2}
```

```
set_false_path -through [get_pins {U1/S:Y}]
```

Invalid Command Examples

```
set_false_path
```

This is invalid because there is no argument supplied.

set_load (SDC load Constraint)

Sets the capacitance to a specified value on a specified port.

Supported Families

SX-A, eX, RTSX-S, Axcelerator, RTAX-S, ProASIC3, and ProASIC3E

Syntax

```
set_load load_value  port
```

Arguments

Note: All arguments are mandatory.

`load_value`

Specifies the capacitance value. The *load_value* is an integer. There is no default value.

`port`

Specifies the port in the design on which the capacitance is to be set.

a single design output port name; for instance o1

a list of one output port name in curly brackets; for instance: {Out}

an object accessor that will refer to one output port name for instance: [get_ports{ex}]

Note: Wildcards are not allowed.

Valid Command Examples

```
set_load  35   out_p
set_load  40   {O1}
set_load  25   [get_ports out]
```

Invalid Command Examples

```
set_load 30
```

This is invalid because there is no port supplied.

set_max_delay (SDC max path Constraint)

The set_max_delay constraint sets the path delay between the specified ports to a restricted value.

Supported Families

SX-A, eX, RTSX-S, ProASIC^{PLUS} (APA), Axcelerator, RTAX-S, and ProASIC3/E

Syntax

```
set_max_delay delay_value -from from_list -to to_list
```

Arguments

`delay_value`

Mandatory argument. The unit is ns. There is no default value. A negative or null value is not allowed.

`from_list`

Mandatory collection of start point names: primary input or inout port, clock pins of registers:

a single pin/port name; for instance reg1:CLK

a list of pins/ports enclosed in curly brackets; for instance {A} or {toplevel/pll:clk1 toplevel/pll:clk2}

an object accessor: for instance [get_ports {A B}] or [get_pins reg:CLK]

Note: Wildcards are accepted.

`to_list`

Mandatory collection of end point names: primary output or inout port, synchronous pins of registers:

a single pin/port name; for instance reg2:D

a list of pins/ports enclosed in curly brackets; for instance {O*} or {out1 out2}

an object accessor: for instance [get_ports {ex}] or [get_pins reg*:D]

Note: Wildcards are accepted.

Valid Command Examples

```
set_max_delay 9 -from [get_pins reg1:CLK] -to [get_ports {out1 out2}]
set_max_delay 7.5 -from {reg2:CLK} -to [all_outputs]
set_max_delay 3.0 -from [get_ports  {Data}] -to out
set_max_delay 2.7 -from {A B} -to [get_pins {reg2*}]
set_max_delay 0.5 -from [get_ports {C*}] -to [get_clocks {clk1 clk2}]
```

Invalid Command Example

```
set_max_delay -from [get_ports {IN10 IN11}]
```

This is invalid because the *to_list* and *delay_value* are not supplied.

set_multicycle_path (SDC multiple cycle path Constraint)

Defines a path that takes multiple clock cycles.

Supported Families

ProASIC^{PLUS}, Axcelerator, RTAX-S, and ProASIC3/E

Syntax

```
set_multicycle_path path_multiplier [-from from_list] [-to to_list]
```

Arguments

`path_multiplier`

Mandatory integer that specifies the number of cycles the data path must have for setup and hold, relative to the startpoint clock before data is required at the endpoint.

`from_list`

Specifies a list of timing path startpoint names: primary input or inout port, clock pins of registers:

a single pin/port name; for instance reg1:CLK

a list of pins/ports enclosed in curly brackets; for instance {A} or {toplevel/pll:clk1 toplevel/pll:clk2}

an object accessor: for instance [get_ports {A B}] or [get_pins reg:CLK]

Note: Wildcards are accepted.

`to_list`

Specifies a list of timing path endpoint names: primary output or inout port, synchronous pins of registers:

a single pin/port name; for instance reg2:D

a list of pins/ports enclosed in curly brackets; for instance {O*} or {out1 out2}

an object accessor: for instance [get_ports {ex}] or [get_pins reg*:D]

Note: Wildcards are accepted.

Valid Command Examples

```
set_multicycle_path 2 -from [get_ports {reg1*}] \
-to {reg2:D}
set_multicycle_path 2 -from {reg:CLK}
set_multicycle_path 2 -to [get_ports {o1 o2}]
set_multicycle_path 3 -from data1 -to [get_pins {reg1*}]
set_multicycle_path 2 -from {reg1:CLK} -to [get_clocks {clk}]
```

Invalid Command Examples

```
set_multicycle_path -from [get_ports {IN0 IN1}]
```

This is invalid because the path multiplier is not supplied.

SDC Command Limitations

Not all object and design constraint commands are supported in Designer. There are limitations on SDC support. Refer to the latest Designer series Release Notes for the latest supported Object Access, Design Constraints, and Supported Features.

Naming Conventions

You can use * in the object names except for the clocks (create_clock (SDC clock constraint)) and load (set_load (SDC load constraint)).

For ProASIC^{PLUS} and ProASIC3/E families: Some special characters are not allowed in the Actel internal netlists. In case the user netlist contains object names with such characters and constraints are set on these, the tool does a mapping between user and internal object names. The Timer GUI displays the internal netlist object names, although these could be different from the object names in the SDC file. Timer applies the correct constraints on these object names.

Multiple Files

When you import SDC as a source file, you can specify multiple SDC files in the **File > Import Source** Files dialog box.

When you import SDC as auxiliary file, you can specify only one file in the **File > Import Auxiliary Files** dialog box.

global_clocks

The GLOBAL_CLOCKS section is used to describe the clock waveforms from the global clock distribution networks. Local clocks, such as gated clocks, are not directly supported. The clock waveforms are used to generate automatically the timing constraints of the paths between two sequential elements. To allow more user control when clocks interact, there are provisions to specify the clock

period transitions, which should be considered. By default, the closest transitions are used when two clocks interact. The clock waveform specification has the following format:

```
WAVEFORM clkname RISE value FALL value PERIOD value [EXCEPT SOURCE {sequential list }| EXCEPT SINK{sequential list}].
```

clkname is the name of the macro driving the clock network.

RISE/FALL/PERIOD can be specified as either an integer or a floating point number followed by an unit selected from {NS, US, PS}. The default time unit is 0.1ns.

EXCEPT {SOURCE|SINK} {list of sequential elements} is the list of sequential elements which should not be included as endpoints in the automaticallygenerated paths involving sequential elements.

The MULTICLOCK specification is used to specify which clock periods should be considered during the generation of the path constraints involving sequential elements. The default specification is to consider only the closest clock periods of the SOURCE and DESTINATION clocks. This specification has the following syntax:

```
MULTICYCLE SOURCE clkA CYCLE value EXCEPT {seqlist}; DESTINATION clkB CYCLE value EXCEPT {seqlist}.

MULTICYCLE SOURCE clkA CYCLE value EXCEPT {seqlist}.
```

clkA/clkB is the name of the macro driving the clock network.

EXCEPT {seqlist}: By default all sequential elements clocked by the clock driver are included. The EXCEPT seqlist is a list of all the sequential elements or specific pins to be excluded.

CYCLE value. Be default, the closest transitions are considered. CYCLE provides the ability to use transitions from one or more clock periods past the closest transition. CYCLE zero indicates the closest transitions. CYCLE one skips the closest set of transitions and uses the next set of transitions. The term cycle is used to avoid confusion with the term period in the clock waveform specification. This allows you to specify a cycling-stealing clocking regime.

max_delays/min_delays

The sections MAX_DELAYS and MIN_DELAYS use the following format:

```
DELAY value timeunit; SOURCE {source_name_list} [EXCEPT {source_name_list}]; SINK {sink_name_list} [EXCEPT {sink_name_list}]; [STOP {stop_name_list}]; [PASS {bypass_name_list}].
```

value can be specified as an integer.

timeunit allows {PS, NS, US}.

source_name_list is a list of signal sources. It can be one of the following: a macro output pin, macro name, or primary input.

sink_name_list is a list of signal destinations. It can be a name of a macro or a primary output.

EXCEPT allows you to exclude certain sources or sinks from consideration.

stop_name_list is the list of pin names through which further propagation of signals will not be considered. This allows you to eliminate certain paths from consideration.

bypass_name_list is the list of latches which are allowed to be intermediate path points. By default, latches are considered to be sinks or path terminals.

INPAD/OUTPAD/GATED are valid values for any of the lists, such as source_name_list or stop_name_list.

Normally there is no need to specify any timing requirements from any source to any sink clocked by an external global clock. This timing requirement can be generated automatically from the GLOBAL_CLOCK specifications and the sequential elements setup and hold times. For example, the timing constraint from a primary input to a sequential element can be derived from the sequential elements clocking waveform and the signal arrival time of the primary input.

A problem exists when two different internally-generated clock signals interact. This is due to the unpredictable and unknown skew between the two clock networks because of the routing delays from:

PAD >> internalMacro >> CLKINT

where CLKINT is the input pin of the global clock distribution network. The automatically generated path constraints will not incorporate the skew between the two clocks. In such cases, the path constraints should be expressed explicitly using the MAX/MIN_DELAYS section.

NOTE: The most stringent timing constraint dominates. Hence, all general constraints should be looser than the specific constraints. For example, in the following example, the 26.0ns constraint dominates the 42.0ns constraint:

```
DELAY 42.0 ns SOURCE INPAD SINK OUTPAD.

DELAY 26.0 ns SOURCE {$1I23:Q $1I24:Q} SINK {ack_0}.
```

If the general constraint is tighter than the specific constraint, the specific constraint will effectively become a no-operation. In the following example, the looser constraint of 42.0ns has no effect since the general constraint of 26.0ns dominates.

```
DELAY 26.0 ns SOURCE INPAD SINK OUTPAD.

DELAY 42.0 ns SOURCE {$1I23:Q $1I24:Q} SINK {ack_0}.
```

This implies that a specific path or paths which has (have) looser timing constraints must be explicitly excluded using the EXCEPT syntax. This is especially important for the path constraints generated automatically from the global clock specification. For example, the ENABLE signal for a flip-flop is sometimes allowed to be much slower than the DATA signal. The constraint associated with the DATA signal can be inferred from the global clocking specification; however, in order for a looser constraint to be associated with the ENABLE signal, the ENABLE input pin must appear in the appropriate EXCEPT list in the GLOBAL_CLOCK section.

The section MAX_DELAYS can be empty if there are no purely combinatorial paths from external sources to external sinks, and if every sequential element in the design is clocked by an external global clock. In this case, the timing constraints are generated automatically using the information in the GLOBAL_CLOCK section. Likewise, the MIN_DELAYS section can be empty.

One final word about external/internal sinks and sources with regard to the flip-flops and/or latches in the IOs: these flip-flops act as internal, not external, sources/sinks.

io_arrival_Times

The section IO_ARRIVAL_TIMES uses the following format:

```
[early_arrival_time:] late_arrival_time timeunit {source_io_list} EXCEPT {source_io_list}.
```

`{early, late}_arrival_time` is signal arrival time relative to the reference time.

`source_io_list` is an INPAD or primary input pin.

Note that the section IO_ARRIVAL_TIMES can be empty. For example,

```
SECTION IO_ARRIVAL_TIMES.

END.
```

is entirely equivalent to

```
SECTION IO_ARRIVAL_TIMES.

0 0 {INPAD}.

END
```

global_stops

The GLOBAL_STOPS section is used to disable dont care/false paths by preventing the specified pins from being used in ANY timingcritical paths. Any path involving pins that appear in this section should be removed from consideration.

pin_loads

The PIN_LOADS section is used to specify the capacitance loading and logic (TTL/CMOS) at a package pin. The default logic family is TTL. The format is:

```
capValue capUnit [TTL/CMOS]{[macList|pinList]} [EXCEPT {}].
```

I/O Attributes Reference (alphabetical order)

I/O Attributes by Family

Other than the four common attributes supported by all device families, the following table includes the attributes that each Actel device family supports.

Attribute	Family								
	ProASIC3E	ProASIC3	ProASIC PLUS	Axcelerator	ProASIC	SX-A	SX	eX	MX
Bank Name	X	X		X					
I/O Standard	X	X	X	X	X	X	X	X	X
I/O Threshold						X	X	X	X
Output Drive	X	X		X					
Slew	X	X				X	X	X	X
Power Up State						X	X	X	X
Resistor Pull	X	X		X		X			
Schmitt Trigger	X								
Input Delay	X			X					
Skew	X	X							

Output Load	X	X	X	X	X	X	X	X	X
Use Register	X	X		X					
Hot Swappable	X	X		X				X	

Refer to the appropriate datasheet for information about I/O standards for different families.

Bank Name

Purpose: Displays the name of the bank to which the I/O macro has been assigned. You cannot change the bank name.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	No
Axcelerator	Yes
ProASIC	No
SX-A	No
SX	No
eX	No
MX	No

Hot Swappable

The I/O standard specified and the selected voltage determine this **read-only** attribute.

Purpose: Indicates whether the I/O pin is hot swappable.

Families	Supported
ProASIC3E	Yes
ProASIC3	No
ProASIC PLUS	No
Axcelerator	Yes
ProASIC	No
SX-A	Yes
SX	No
eX	Yes
MX	No

Values:

If checked or ON (all standards except PCI and PCIX), a clamp diode is NOT included to allow proper hot-swap behavior. If not checked or OFF (PCI and PCIX only), the clamp diode is included as required by those specifications, but the I/O is NOT hot swappable.

Input Delay

Purpose: Indicates whether the input path delay elements are to be programmed. If they will be programmed, this option adds the specified input delay to the input path.

Families	Supported
ProASIC3E	Yes
ProASIC3	No
ProASIC PLUS	No
Axcelerator	Yes
ProASIC	No
SX-A	No
SX	No
eX	No
MX	No

Values: Use this attribute to turn the input delay on or off.

Default value: The default value is off.

For ProASIC3E devices, you specify the input delay per pin. You will see the actual delay only in Timer or in the SDF file.

Note: The actual input delay is a function of the operating conditions and is automatically computed by the delay extractor when a timing report is generated.

For Axcelerator devices, you specify the input delay per bank. You then set its input delay with the slider in the Other I/O Bank AttributesOther I/O Bank Attributes dialog box. Possible values are 0 to 31.

PDC command for setting this attribute: `-set_io in_delay` (See the set_io command for Axcelerator and for ProASIC3E in the PDC Command Reference section of the Designer User's Guide.)

I/O Standard

Purpose: Use the I/O standard attribute to assign an I/O standard to an I/O macro.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	Yes
Axcelerator	Yes
ProASIC	Yes
SX-A	Yes
SX	No
eX	No
MX	No

Note: Voltage referenced I/O inputs require an input referenced voltage (VREF). You must assign VREF pins to Axcelerator and ProASIC3E devices before running Layout.

The ProASIC3E, ProASIC3, and Axcelerator devices support multiple I/O standards (with different I/O voltages) in a single die. You can use I/O Attribute Editor to set I/O standards and attributes, or alternatively you can export and import this information using a PDC file.

Not all devices support all I/O standards. The following table shows you which I/O standards are supported by each device.

I/O Standard	ProASIC3E	ProASIC3	ProASIC PLUS	Axcelerator	ProASIC	SX-A
CMOS	X					X
CUSTOM	X					X
GTL+	X			X		
GTL 3.3V	X			X		
GTL 2.5V	X			X		
HSTL Class I	X			X		
HSTL Class II	X					
LVCMOS 3.3V	X	X				
LVCMOS 2.5V	X		X	X	X	
LVCMOS 2.5V/5.0V	X	X				
LVCMOS 1.8V	X	X		X		
LVCMOS 1.5V	X	X	X	X	X	
LVDS	X	X		X		
LVPECL	X	X	X*	X		
LVTTL/TTL	X	X	X	X	X	X
PCI	X	X	X	X	X	X
PCI-X 3.3V	X	X		X		
SSTL2 Class I and II	X			X		
SSTL3 Class I and II	X			X		

***Supported only on dedicated LVPECL I/Os.**

Note: For a list of I/O standards for all other families, refer to the datasheet for your specific device.

Descriptions

Following are brief descriptions of the I/O standard attributes in the table above:

CMOS (Complementary Metal-Oxide-Semiconductor)

An advanced integrated circuit (IC) manufacturing process technology for logic and memory, characterized by high integration, low cost, low power, and high performance. CMOS logic uses a combination of p-type and n-type metal-oxide-semiconductor field effect transistors (MOSFETs) to implement logic gates and other digital circuits found in computers, telecommunications, and signal processing equipment.

CUSTOM

An option in the I/O Attribute Editor that enables you to customize individual I/O settings such as the I/O threshold, output slew rates, and capacitive loadings on an individual I/O basis. For example, PCI mode output can be set to low-slew rate. For more information, go to the Actel web site and check the datasheet for your device.

GTL 2.5V (Gunning Transceiver Logic 2.5 Volts)

A low-power standard (JESD 8.3) for electrical signals used in CMOS circuits that allows for low electromagnetic interference at high speeds of transfer. It has a voltage swing between 0.4 volts and 1.2 volts, and typically operates at speeds of between 20 and 40MHz. The VCCI must be connected to 2.5 volts.

GTL 3.3V (Gunning Transceiver Logic 3.3 Volts)

Same as GTL 2.5V above, except the VCCI must be connected to 3.3 volts.

GTL+ (Gunning Transceiver Logic Plus)

An enhanced version of GTL that has defined slew rates and higher voltage levels. It requires a differential amplifier input buffer and an open-drain output buffer. Even though output is open-drain, the VCCI must be connected to either 2.5 volts or 3.3 volts for Axcelerator, ProASIC3, and ProASIC3E device support.

HSTL Class I and II (High-Speed Transceiver Logic)

A general-purpose, high-speed 1.5V bus standard (EIA/JESD 8-6) for signalling between integrated circuits. The signalling range is 0 V to 1.5V, and signals can be either single-ended or differential. HSTL requires a differential amplifier input buffer and a push-pull output buffer. It has four classes, of which Actel supports Class I and II. These classes are defined by standard EIA/JESD 8-6 from the Electronic Industries Alliance (EIA):

- Class I (unterminated or symmetrically parallel terminated)
- Class II (series terminated)
- Class III (asymmetrically parallel terminated)
- Class IV (asymmetrically double parallel terminated

LVCMOS 3.3V (Low-Voltage CMOS for 3.3 Volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 3.3V applications.

LVCMOS 2.5V (Low-Voltage CMOS for 2.5 Volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 2.5V applications.

LVCMOS 2.5V/5.5V (Low-Voltage CMOS for 2.5 and 5.0 Volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 2.5V and 5.0V applications.

LVCMOS 1.8V (Low-Voltage CMOS for 1.8 Volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 1.8V applications. It uses a 3.3V-tolerant CMOS input buffer and a push-pull output buffer.

LVCMOS 1.5V (Low-Voltage CMOS for 1.5 volts)

An extension of the LVCMOS standard (JESD 8-5) used for general-purpose 1.5V applications. It uses a 3.3V-tolerant CMOS input buffer and a push-pull output buffer.

LVDS (Low-Voltage Differential Signal)

A moderate-speed differential signalling system, in which the transmitter generates two different voltages which are compared at the receiver. It requires that one data bit be carried through two signal lines; therefore, you need two pins per input or output. It also requires an external resistor termination. The voltage swing between these two signal lines is approximately 350mV (millivolts). Axcelerator devices contain dedicated circuitry supporting a high-speed LVDS standard that has its own user specification.

LVPECL (Low-Voltage Positive Emitter Coupled Logic)

PECL is another differential I/O standard. It requires that one data bit is carried through two signal lines; therefore, two pins are needed per input or output. It also requires an external resistor termination. The voltage swing between these two signal lines is approximately 850mV. When the power supply is +3.3V, it is commonly referred to as low-voltage PECL (LVPECL).

LVTTL/TTL (Low-Voltage Transitor-Transistor Level)

A general purpose standard (EIA/JESDSA) for 3.3V applications. It uses an LVTTL input buffer and a push-pull output buffer.

PCI (Peripheral Component Interface)

A computer bus for attaching peripheral devices to a computer motherboard in a local bus. This standard supports both 33 MHz and 66 MHz PCI bus applications. It uses an LVTTL input buffer and a push-pull output buffer. With the aid of an external resistor, this I/O standard can be 5V-compliant for most families, excluding ProASIC3/E families.

PCI-X (Peripheral Component Interface Extended)

An enhanced version of the PCI specification that can support higher average bandwidth; it increases the speed that data can move within a computer from 66 MHz to 133 MHz. PCI-X is backward-compatible, which means that devices can operate at conventional PCI frequencies (33 MHz and 66 MHz). PCI-X is also more fault-tolerant than PCI.

SSTL2 Class I and II (Stub Series Terminated Logic 2.5V)

A general-purpose 2.5V memory bus standard (JESD 8-9) for driving transmission lines. This standard was designed specifically for driving the DDR (double-data-rate) SDRAM modules used in computer memory. It requires a differential amplifier input buffer and a push-pull output buffer. It has two classes, of which Actel supports both.

SSTL3 Class I and II (Stub Series Terminated Logic for 3.3V)

A general-purpose 3.3V memory bus standard (JESD 8-8) for driving transmission lines.

PDC command for setting this attribute: `-set_io (`See the set_io command for ProASIC3E, ProASIC3, or Axcelerator in the PDC Command Reference section of the Designer User's Guide.)

I/O Threshold

Purpose: Indicates the compatible threshold level for inputs and outputs.

Families	Supported
ProASIC3E	No
ProASIC3	No
ProASIC PLUS	No
Axcelerator	No
ProASIC	No
SX-A	Yes
SX	Yes
eX	Yes
MX	Yes

Values: Use this attribute to set the compatible threshold level for inputs and outputs. Your options are CMOS, LVTTL, or PCI.

Default value: The default I/O threshold displayed is based upon the I/O standard. If you want to set the I/O threshold independently of the I/O specification, you must select CUSTOM in the I/O standard cell.

Locked

Purpose: Indicates whether you can change the current pin assignment during layout.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	Yes
Axcelerator	Yes
ProASIC	Yes
SX-A	Yes
SX	Yes
eX	Yes
MX	Yes

Values: Use this attribute to lock or unlock the pin assignment. Selecting the check box locks the pin assignment. Clearing the check box unlocks the pin assignment. If locked, you cannot change the pin assignment. If not locked, you can.

PDC command for setting this attribute: -set_io `-fixed` (See the set_io command for ProASIC3E, ProASIC3, or Axcelerator in the PDC Command Reference section of the Designer User's Guide.)

Macro cell

Purpose: Indicates the type of I/O macro. This value is read only and is applicable only to the I/O Attribute Editor (that is, you cannot use it in GCF or PDC files).

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	Yes
Axcelerator	Yes
ProASIC	Yes
SX-A	Yes
SX	Yes
eX	Yes
MX	Yes

Output Drive

Purpose: Every I/O standard has an output drive preset; however, for some I/O standards, you can choose which one to use. The higher the drive, the faster the I/O. The faster the I/O, the more power consumed by the I/O.

Families	Supported

ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	No
Axcelerator	Yes
ProASIC	No
SX-A	No
SX	No
eX	No
MX	No

Values: Use this attribute to set the strength of the output buffer to between 2 and 24 mA, weakest to strongest, depending on your device family. The LVTTL output buffer has four programmable settings of its drive strength. Other I/O standards have full strength.

The list of I/O standards for which you can change the output drive and the list of values you can assign for each I/O standard is family-specific. Refer to the datasheet for your device for more information.

PDC command for setting this attribute: `-set_io -out_drive` (See the set_io command for ProASIC3E, ProASIC3, or Axcelerator in the PDC Command Reference section of the Designer User's Guide.)

Output Load or Loading (pf)

Note: In MVN tools, the column heading for this attribute is "Output load." In non-MVN tools, the column heading for this attribute is "Loading (pf)."

Purpose: Indicates the output-capacitance value based on the I/O standard selected in the I/O Standard cell. This option is not available in ACTgen.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	Yes
Axcelerator	Yes
ProASIC	Yes
SX-A	Yes
SX	Yes
eX	Yes
MX	Yes

Values: You can enter a capacitative load as an integral number of picofarads. 35pF is the default.

Default value: The default value is based upon the I/O specification set in the I/O Standard cell. If necessary, you can change the output capacitance default setting to improve timing definition and analysis. Both the capacitive loading on the board and the Vil/Vih trip points of driven devices affect output-propagation delay.

Timer, Timing-Driven Layout, Timing Report, and Back-Annotation automatically uses the modified delay model for delay calculations.

PDC command for setting this attribute: `-set_io -out_load` (See the set_io command for ProASIC3E, ProASIC3, or Axcelerator in the PDC Command Reference section of the Designer User's Guide.)

Pin Number

Purpose: Use this attribute to change a pin assignment by choosing one of the legal values from the drop-down list. If the pin has been assigned, the pin number appears in this column. If it hasn't been assigned, "Unassigned" appears in this column.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	Yes
Axcelerator	Yes
ProASIC	Yes
SX-A	Yes
SX	Yes
eX	Yes
MX	Yes

PDC command for setting this attribute: -set_io `-pinname` (See the set_io command for ProASIC3E, ProASIC3, or Axcelerator in the PDC Command Reference section of the Designer User's Guide.)

Port Name

Purpose: Indicates the port name of the I/O macro. This value is read only.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	Yes
Axcelerator	Yes
ProASIC	Yes
SX-A	Yes
SX	Yes
eX	Yes
MX	Yes

Power-Up State

Purpose: Indicates the power-up state of the pin.

Families	Supported
ProASIC3E	No
ProASIC3	No
ProASIC PLUS	No
Axcelerator	No
ProASIC	No
SX-A	Yes
SX	Yes

eX	Yes
MX	Yes

Values: Use this attribute to set the power-up state. Your choices are None, High, and Low.

Default value: The default value is **None**.

Resistor Pull

Purpose: Allows inclusion of a weak resistor for either pull-up or pull-down of the input buffer.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	No
Axcelerator	Yes
ProASIC	No
SX-A	Yes
SX	No
eX	No
MX	No

Values: Use this attribute to set the resistor pull. Your choices are None, Up (pull-up), or Down (pull-down).

Default value: The default value is **None** except when an I/O exists in the netlist as a port, is not connected to the core, and is configured as an output buffer. In that case, the default setting is for a weak pull-down.

PDC command for setting this attribute: -set_io `-res_pull` (See the set_io command for ProASIC3E, ProASIC3, or Axcelerator in the PDC Command Reference section of the Designer User's Guide.)

Schmitt Trigger

Purpose: A schmitt trigger is a buffer used to convert a slow or noisy input signal into a clean one before passing it to the FPGA. This is a simple, low-cost solution for a user working with low slew-rate signals. Using schmitt-trigger buffers guarantees a fast, noise-free, input signal to the FPGA.

Actel recommends that you use a schmitt trigger to buffer a signal if input slew rates fall below the values outlined in the specification for SX-A and RTSX-S devices. Depending on the application, different schmitt-trigger buffers can be used to fulfill the requirements.

Schmitt-trigger buffers are categorized in three configurations:

- Fixed threshold voltages with non-inverted outputs
- Fixed threshold voltages and inverted outputs
- Variable threshold voltages with non-inverted outputs

With the aid of schmitt-trigger buffers, low slew-rate applications can also be handled with ease. Implementation of these buffers is simple, not expensive, and flexible in that different configurations are possible depending on the application. The characteristics of schmitt-trigger buffers (e.g. threshold voltage) can be fixed or user-adjustable if required.

Families	Supported
ProASIC3E	Yes

ProASIC3	No
ProASIC PLUS	Yes*
Axcelerator	No
ProASIC	No
SX-A	No
SX	No
eX	No
MX	No

*Although ProASIC PLUS supports the schmitt-trigger attribute,you cannot edit this attribute with the MultiView Navigator tools. Instead, it has to be instantiated in the schematic or the netlist.

Values: A schmitt trigger has two possible states: on or off. The trigger for this circuit to change states is the input voltage level. That is, the output state depends on the input level, and will change only as the input crosses a pre-defined threshold.

Default value: The default value is On.

For more information, please see the "Using Schmitt Triggers for Low Slew-Rate Input" Application Note on the Actel web site.

PDC command for setting this attribute: -set_io `-schmitt_trigger` (See the set_io command for ProASIC3E in the PDC Command Reference section of the Designer User's Guide.)

Skew

Purpose: Indicates whether there is a fixed additional delay between the enable/disable time for a tristatable I/O. (A tristatable I/O is an I/O with three output states: high, low, and high impedance.) 2 ns delay on rising edge, 0 ns delay on falling edge.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	No
Axcelerator	No
ProASIC	No
SX-A	No
SX	No
eX	No
MX	No

Values: You can set the skew for a clock to either **on** or **off**.

Default value: The default skew displayed in the I/O Attribute Editor is **off**.

Note: A Tri State or "tristatable" logic gate has three output states: high, low, and high impedance. In a high impedance state, the output acts like a resistor with infinite resistance, which means the output is disconnected from the rest of the circuit.

PDC command for setting this attribute: -set_io `-skew` (See the set_io command for ProASIC3E or ProASIC3 in the PDC Command Reference section of the Designer User's Guide.)

Slew

The slew rate is the amount of rise or fall time an input signal takes to get from logic low to logic high or vice versa. It is commonly defined to be the propagation delay between 10% and 90% of the signal's voltage swing.

Purpose: Indicates the slew rate for output buffers. Generally, available slew rates are high and low.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	No
Axcelerator	Yes
ProASIC	No
SX-A	Yes
SX	Yes
eX	Yes
MX	Yes

Values: You can set the slew rate for the output buffer to either **high** or **low**. The output buffer has a programmable slew rate for both high-to-low and low-to-high transitions. The low slew rate is incompatible with 3.3V PCI requirements.

For ProASIC3E and ProASIC3 families, you can edit the slew for designs using LVTTL, all LVCMOS, or PCIX I/O standards. The other I/O standards have a preset slew value.

For the Axcelerator family, you can edit the slew only for designs using the LVTTL I/O standard. For those devices that support additional slew values, Actel recommends that you use the high and low values and let the software map to the appropriate absolute slew value.

Default value: The default slew displayed in the I/O Attribute Editor is based on the selected I/O standard. For example, PCI mode sets the default output slew rate to High.

Note: One way to eliminate problems with low slew rate is with external schmitt triggers.

In some applications, you may require a very fast (i.e. high slew rate) signal, which approaches an ideal switching transition. You can accomplish this by either reducing the track resistance and/or capacitance on the board or increasing the drive capability of the input signal. Both of these options are generally time consuming and costly. Furthermore, the closer the input signal approaches an ideal one, the greater the likelihood of unwanted effects such as increased peak current, capacitive coupling, and ground bounce. In many cases, you may want to incorporate a finite amount of slew rate into your signal to reduce these effects. On the other hand, if an input signal becomes too slow (i.e. low slew rate), then noise around the FPGA's input voltage threshold can cause multiple state changes. During the transition time, both input buffer transistors could potentially turn on at the same time, which could result in the output of the buffer to oscillate unpredictably. In this situation, the input buffer could still pass signals. However, these short, unpredictable oscillations would likely cause the device to malfunction. Actel has performed reliability tests on RTSX-S devices and the reliability of the device is guaranteed for signals with slew rates up to 500µs. This test has not been performed on the SX-A family. For more information, see the *RTSX-S TR/TF Experiment* report on the Actel web site.

PDC command for setting this attribute: -set_io `-slew` (See the set_io command for ProASIC3E, ProASIC3, or Axcelerator in the PDC Command Reference section of the Designer User's Guide.)

Use Register

Purpose: The input and output registers for each individual I/O can be activated by selecting the check box associated with an I/O. The I/O registers are NOT selected by default.

Families	Supported
ProASIC3E	Yes
ProASIC3	Yes
ProASIC PLUS	No

Axcelerator	Yes
ProASIC	No
SX-A	No
SX	No
eX	No
MX	No

PDC command for setting this attribute: -set_io `-register` (See the set_io command for ProASIC3E, ProASIC3, or Axcelerator in the PDC Command Reference section of the Designer User's Guide.)

<Jerome, for "ProASIC3E Restrictions" below - what does this mean? Can you explain it so the user can understand?>

ProASIC3E Restrictions: New DRC must be developed for I/O register combining. The following architecture rules must be enforced:

1. DDR_REG and DDR_OUT macro must be combinable. There are no resources in the core tiles to implement the I/O DDR macros.

a. I/O to DDR_REG fanout =1.

b. DDR_OUT to I/O fanout =1.

c. If DDR_REG and DDR_OUT are combined on the same I/O, they must share the same clear signal.

d. Flip-flops will not be combined in an I/O in the presence of DDR combining on the same I/O.

ProASIC3 restrictions: The default I/O register combining behavior is controlled by a user global variable set in Compile.

Closing and Exiting

Your project is automatically saved when closed. To explicitly save your project, use File -> Save Project. To save it with another name, use the Save Project As command.

To close a project, from the **File** menu, select **Close Project**.

To exit Libero IDE, from the **File** menu, select **Exit**.

Actel Headquarters

Actel Corporation is a supplier of innovative programmable logic solutions, including field-programmable gate arrays (FPGAs) based on Antifuse and Flash technologies, high-performance intellectual property (IP) cores, software development tools, and design services targeted for the high-speed communications, application-specific integrated circuit (ASIC) replacement, and radiation-tolerant markets.

Address:	Actel Corporation 2061 Stierlin Court Mountain View CA 94043-4655 USA
Phone:	(650) 318-4200 (650) 318-4600

Technical Support

Highly skilled engineers staff the Technical Support Center from 7:00 AM to 6:00 PM Pacific Time, Monday through Friday.

Visit Tech Support Online

For 24-hour support resources, visit Actel Technical Support at http://www.actel.com/custsup/search.html.

Contacting Technical Support

Contact us with your technical questions via e-mail or by phone. When sending your request to us, please be sure to include your full name, company name, email address, and telephone number.

E-mail (Worldwide):	tech@actel.com
Telephone (In U.S.):	(650) 318-4460 (800) 262-1060
Telephone (Outside the US):	Contact a local sales office

Customer Service

Contact Customer Service for order status, order expedites, return material authorizations (RMA), and first article processing. For technical issues, contact Technical Support.

From	Call
Northeast and North Central U.S.A.	(650) 318-4480
Southeast and Southwest U.S.A.	(650) 318-4480
South Central U.S.A.	(650) 318-4434
Northwest U.S.A.	(650) 318-4434
Canada	(650) 318-4480
Europe	(650) 318-4252 or +44 (0) 1276 401500
Japan	(650) 318-4743
From the rest of the world	(650) 318-4743

UNIX Help Known Issues

Related Topics Links Appear Broken

UNIX (Linux and SOL) do not support links in Related Topics buttons that point to different directories. This has to do with the way the help is built for UNIX systems.

Index

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

http://www.actel.com

Actel Corporation
2061 Stierlin Court
Mountain View, CA
94043-4655 USA
Phone 650.318.4200
Fax 650.318.4600

Actel Europe Ltd.
Dunlop House, Riverside Way
Camberley, Surrey GU15 3YL
United Kingdom
Phone +44 (0)1276 401 450
Fax +44 (0)1276 401 490

Actel Japan
EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Phone +81.03.3445.7671
Fax +81.03.3445.7668

Actel Hong Kong
39th Floor, One Pacific Place
88 Queensway, Admiralty
Hong Kong
Phone +852.227.35712
Fax +852.227.35999