



CALYPTO

Catapult[®] C Library Builder User's and Reference Manual

Calypto Design Systems, Inc
2933 Bunker Hill Lane, Suite 202
Santa Clara, CA 95054
Tel: +1-408-850-2300
Fax: +1-408-850-2301

The software described herein is copyright ©2002-2011 Calypto Design Systems, Inc. All rights reserved. The software described herein, which contains confidential information and trade secrets, is property of Calypto Design Systems, Inc.

This manual is copyright ©2005-2011 Calypto Design Systems, Inc. Printed in U.S.A. All rights reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, transmitted, or reduced to any electronic medium or machine-readable form without prior written consent from Calypto Design Systems, Inc.

This manual and its contents are confidential information of Calypto Design Systems, Inc., and should be treated as confidential information by the user under the terms of the nondisclosure agreement and software license agreement, as applicable, between Calypto Design Systems, Inc. and user.

Last Updated: October 2011

Product Version: Release 2011a

Table of Contents

Chapter 1

Introduction	11
Library Builder Overview	11
Catapult C Library Builder Licenses	12
Invoking the Graphical User Interface	12
Features of the User Interface	13
Library Builder Main Window	13
The Task Bar Window	15
The Command Input and Transcript Window	15
The Library Explorer Window	19
The Library Editor Window	21
The Farm Window	22
Getting Help	23
Setting Library Builder Options	24
Set General Options	24
Set Messages Options	27
Set Component Library Options	31
Set Catapult C Synthesis Options	31
Set Farm Options	33
Set Input Compiler Options	35
View Compiler Settings	37
Set Text Editor Options	37
Set Flows Options	38
Set Precision Flow Options	39
Set Design Compiler Flow Options	41
Set RTL Compiler Flow Options	43
Set TalusDesign Flow Options	45
Saving and Restoring Session Options	47

Chapter 2

Creating and Editing Libraries	51
Setting the Working Directory	51
Creating a New Library	51
Cadence RTL Compiler Options	53
Magma Talus Design Options	57
Precision RTL Synthesis Options	59
Synopsys Design Compiler Options	61
Editing RAM Library Properties	67
Editing the RAM Library Variables	68
Editing RAM Components Parameters	69
Editing RAM Formula Information	70
Editing RAM Timing	71

Editing RAM Ports	72
Saving Libraries	73
Chapter 3	
Library Characterization	75
Multi-Point Characterization	76
Characterizing Libraries or Components	78
Queuing Multiple Libraries for Multi-day Run	81
Library Characterization Results	81
Delay Characterization Properties	82
Resetting the Data Before Another Characterization	83
Plotting the Characterization Data	83
Adding/Removing QMODs	85
Viewing the Characterization Transcript	87
Troubleshooting Library Failures	88
Using the Library Farm	90
Enabling and Configuring Library Farm Options	90
Setting Up Library Farm Hosts	92
Chapter 4	
Creating Custom Operators and Interfaces	95
Introduction	95
Creating the Custom Operator C++ Function	96
Creating a Library for the Custom Operators	96
Creating an ASIC Blank Library	96
Creating an FPGA Blank Library	97
Importing Custom Operators from C++ and HDL	97
Importing Operators from C++ Functions	98
Importing Netlists	101
Editing Libraries	105
Modules	106
Creating RAM without a Reset	118
Programmable Reset Polarity and Multiple Resets	118
Manually Defining Custom Operators	120
Creating Custom Operators with State	124
Verifying the Custom Operator RTL and Custom C++ Function	127
Using Custom Interfaces with SCVerify	127
Chapter 5	
Commands	135
General Command Syntax	135
Documentation Conventions for Catapult Commands	136
Command Interface to the SIF Database	138
Common Command Switches	141
Using Tcl Commands in Scripts	143
Interactive Command Line	144
GUI	144
Command Line Invocation Argument	144

Table of Contents

Tcl Startup Script	145
Command Reference	145
application get	148
application exit	150
application report	151
catapult -product library_builder	152
dofile	154
flow get	155
flow package names	157
flow package option add	158
flow package option get	160
flow package option remove	161
flow package option set	162
flow package provide	163
flow package require	165
flow package script	167
flow package vcompare	168
flow package versions	170
flow package vsatisfies	171
flow run	172
help command	173
help message	175
library add	177
library characterize	180
library edit	181
library get	182
library import	185
library load	190
library rename	192
library remove	194
library report	196
library save	198
library save_commands (Deprecated)	200
library set	201
logfile	203
options defaults	204
options exists	205
options get	206
options load	208
options save	210
options set	213
quit	215
set_working_dir	216
utility farm add	217
utility farm get	218
utility farm release	220
utility farm remove	221
utility farm reserve	222
utility farm reset	223

utility farm set. 224

Index

Third-Party Information

List of Figures

Figure 1-1. Setting Up the C Library Builder Shortcut	13
Figure 1-2. Library Builder Session Window	14
Figure 1-3. Command Input and Transcript Window	15
Figure 1-4. Parts of a Message in the Transcript Window	16
Figure 1-5. Viewing the Long Description of a Message	17
Figure 1-6. Filtering Comments Out of the Transcript (before and after).	18
Figure 1-7. Library Explorer Window	19
Figure 1-8. Library Editor Window	22
Figure 1-9. Library Farm Window	23
Figure 1-10. Set Options General Menu	25
Figure 1-11. Messages Options Dialog Box	28
Figure 1-12. Set Options Component Library	31
Figure 1-13. Set Options Catapult C Synthesis Dialog	32
Figure 1-14. Library Farm Set Up	33
Figure 1-15. Input Options Dialog Box	36
Figure 1-16. Compiler Settings Page	37
Figure 1-17. Set Options Text Editor Menu	38
Figure 1-18. Flow Options	39
Figure 1-19. Precision Flow Options	39
Figure 1-20. Design Compiler Flow Options	41
Figure 1-21. RTL Compiler Flow Options	44
Figure 1-22. Talus Design Flow Options	46
Figure 2-1. Library Creation Dialog Box for Cadence RTL Compiler	54
Figure 2-2. Library Creation Dialog Box for Talus Design	57
Figure 2-3. Library Creation Dialog Box for Precision RTL	60
Figure 2-4. Library Creation Dialog Box for Design Compiler	62
Figure 2-5. Editing the RAM Library Title Variable	69
Figure 2-6. Editing RAM Parameter Information	70
Figure 2-7. Editing RAM Formula Information	71
Figure 2-8. Editing RAM Timing Values	72
Figure 2-9. Editing RAM Ports	73
Figure 3-1. Adder Characteristics Graph	76
Figure 3-2. Multi-Point Characterization Settings on Library Creation Dialog	77
Figure 3-3. Characterizing the Library or Component	79
Figure 3-4. Characterization Status Information	80
Figure 3-5. Characterization Passes	82
Figure 3-6. Sequential Timing Delay Diagram	83
Figure 3-7. Opening a Plot Window for the mgc_xor Module	84
Figure 3-8. Updated Plot with Ninps along X-Axis	85
Figure 3-9. Adding QMOD from the Plot Window	86

Figure 3-10. Viewing the Characterization Transcript	87
Figure 3-11. Opening Failed Object in Catapult C Synthesis	89
Figure 3-12. Entering New Area and Timing Values	90
Figure 3-13. Library Farm Window	92
Figure 4-1. Library with Imported Operator	100
Figure 4-2. Dependencies Imported from Netlist	105
Figure 4-3. Parameters Imported from Netlist.	107
Figure 4-4. Parameters Set Manually	108
Figure 4-5. Input Register Setting	108
Figure 4-6. Effects of Input Register Setting.	109
Figure 4-7. Port Settings Overview	109
Figure 4-8. Effect of Input Register on SeqDelay	110
Figure 4-9. Using the GUI to Set Properties	111
Figure 4-10. Operator with State (MAC).	125
Figure 4-11. mgc_in_wire_wait Transactor Resource.	128
Figure 4-12. mgc_in_wire_wait Transactor Resource Body.	129
Figure 4-13. mgc_in_wire_wait Transactor Resource at_active_clk Method.	130
Figure 4-14. mgc_in_wire_wait Transactor Resource update_z Method	130
Figure 4-15. mgc_in_wire_wait Transactor Resource drive_v_signals Method.	131
Figure 4-16. FSL Timing	132
Figure 5-1. Hierarchy of Objects in the SIF Database.	138
Figure 5-2. Nodes of Interest in the Library Database Hierarchy	183

List of Tables

Table 1-1. Types of Library Builder Message	29
Table 1-2. Platform Identifiers in Registry Filenames	48
Table 3-1. Component Characterization Status	81
Table 4-1. Categories of Module Types	101
Table 4-2. Pin Associations	111
Table 4-3. Operator Pin Bindings	112
Table 4-4. Property Mapping	117
Table 5-1. Basic Tcl Syntax	135
Table 5-2.	136
Table 5-3.	136
Table 5-4. Documentation Conventions for Command Syntax	137
Table 5-5. Commands That Take Database Path Arguments	139
Table 5-6. Alphabetical Command Summary	145
Table 5-7.	168

Library Builder Overview

Catapult™ Library Builder generates libraries for Catapult C Synthesis. These libraries can consist of both technology-specific operators and IP blocks. The technology specific operators, or base library components, such as adders and multipliers, are used by Catapult C Synthesis to schedule an algorithm. Library Builder obtains technology-specific timing and area data for these operators (*characterization*) by using downstream synthesis tools such as Design Compiler and Precision RTL. Operators are characterized based on specific target technologies enabling Catapult C Synthesis to construct very efficient schedules and deliver predictable timing closure for your algorithms.

In addition to the base operator library, Library Builder allows you to build custom IP components such as memory interfaces, register files, and so on. Predefined memory templates are provided for RAM components. You can also create custom components for any existing IP written in RTL.

Use the Library Builder to:

- **Create and characterize new libraries:** For more information, see [“Creating a New Library”](#) on page 51 and [“Library Characterization”](#) on page 75.
- **Reload and edit previously characterized libraries:** You can reload previously characterized libraries into Library Builder and perform basic analysis and alterations. You have the ability to re-characterize the entire library or parts of the library. For more information, see [“Resetting the Data Before Another Characterization”](#) on page 83.
- **Distribute library characterization tasks to multiple machines:** Library Builder contains the “Library Farm” tool that can be used to speed up the process of library characterization by running characterization tasks in parallel on multiple host computers. For more information, see [“Using the Library Farm”](#) on page 90.
- **Change Component Area and Delay Values:** Library Builder provides an easy way to change the area/delay values returned from synthesis. Once these values are modified, the tool will not overwrite these values as part of subsequent characterization runs unless you specifically give an “overwrite” command.
- **Analyze and debug libraries:** One of the primary values of the tool is the ability to analyze and debug why a characterization run failed. This tool gives you an easy method to quickly browse the characterization results for obvious errors. Library Builder generates data files and log files that can be used for troubleshooting. For more information, see [“Troubleshooting Library Failures”](#) on page 88.

- **Create memory templates:** Library Builder lets you modify existing RAM templates. For more information, see [“Editing RAM Library Properties”](#) on page 67.
- **Create custom operators and interfaces:** For more information, see [“Creating Custom Operators and Interfaces”](#) on page 95.

Catapult C Library Builder Licenses

Catapult C Library Builder is a standalone product with which you will be able to create, alter, or view libraries. The automatic “characterization flow” (see [“Library Characterization”](#) on page 75) will invoke Catapult C Synthesis and Design Compiler, therefore you will be required to have licenses for these downstream products. The characterization flow itself is licensed separately. The Library Builder product includes one characterization license which allows characterizations to run on the local host machine. You can purchase additional licenses so that the Library Farm can run characterizations remotely on multiple hosts.

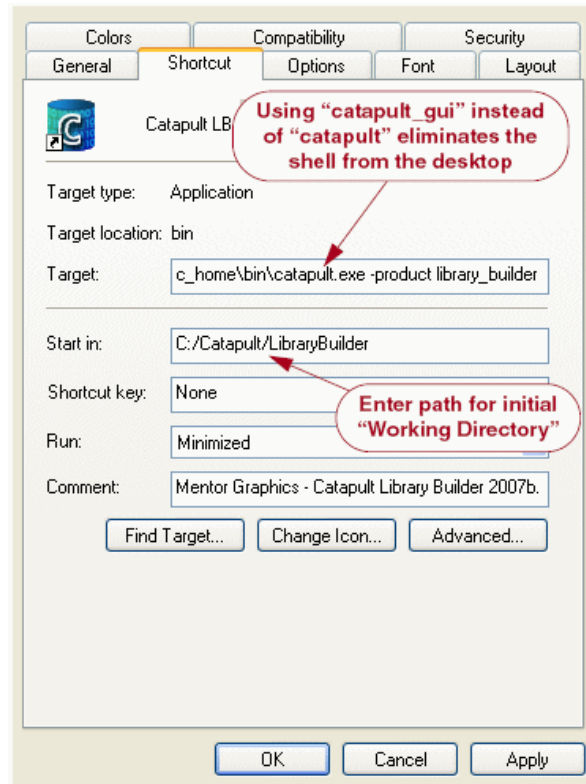
Invoking the Graphical User Interface

You can run Catapult C Library Builder from the GUI or from a shell command line. The shell interface requires that you understand Tcl command syntax and scripting techniques. The section [“General Command Syntax”](#) on page 135 provides information about how Tcl commands are used in the Catapult tool. Reference help files about the Tcl language are available in the Catapult software tree at `$MGC_HOME/pkgs/tcl_msg/man` (UNIX man pages) and `$MGC_HOME/pkgs/tcl_msg/doc` (Windows help files).

You invoke the GUI by entering the `“catapult -product library_builder”` command from a Windows shell or a UNIX/Linux shell. By default, the invocation directory becomes the Library Builder *working directory*. To change the working directory, see [“Setting the Working Directory”](#) on page 51. You can also configure Library Builder to start in a different working directory by changing the “General Options” settings as described in the section [“Set General Options”](#) on page 24.

In a Windows environment, you can optionally create a *Shortcut* on the Desktop and set the shortcut properties similar to that shown in Figure 1-1. The Catapult setup program can create an invocation icon on your desktop for you.

Figure 1-1. Setting Up the C Library Builder Shortcut



Features of the User Interface

This section introduces the key features of the Library Builder graphical user interface (GUI). It describes the layout of the windows and their controls, such as menus, command buttons and dialog boxes. This section covers the following topics:

Library Builder Main Window	13
The Task Bar Window	15
The Command Input and Transcript Window	15
The Library Explorer Window	19
The Library Editor Window	21
The Farm Window	22
Getting Help	23

Library Builder Main Window

Figure 1-2 shows the default layout of the Catapult C Library Builder session window and identifies the key features. The main work windows, Library Explorer, Library Editor and Farm share the upper right portion of the session window. The Task Bar window in the upper left portion gives you quick access to some basic commands. Across the bottom portion of the session is the Command Input and Transcript window.

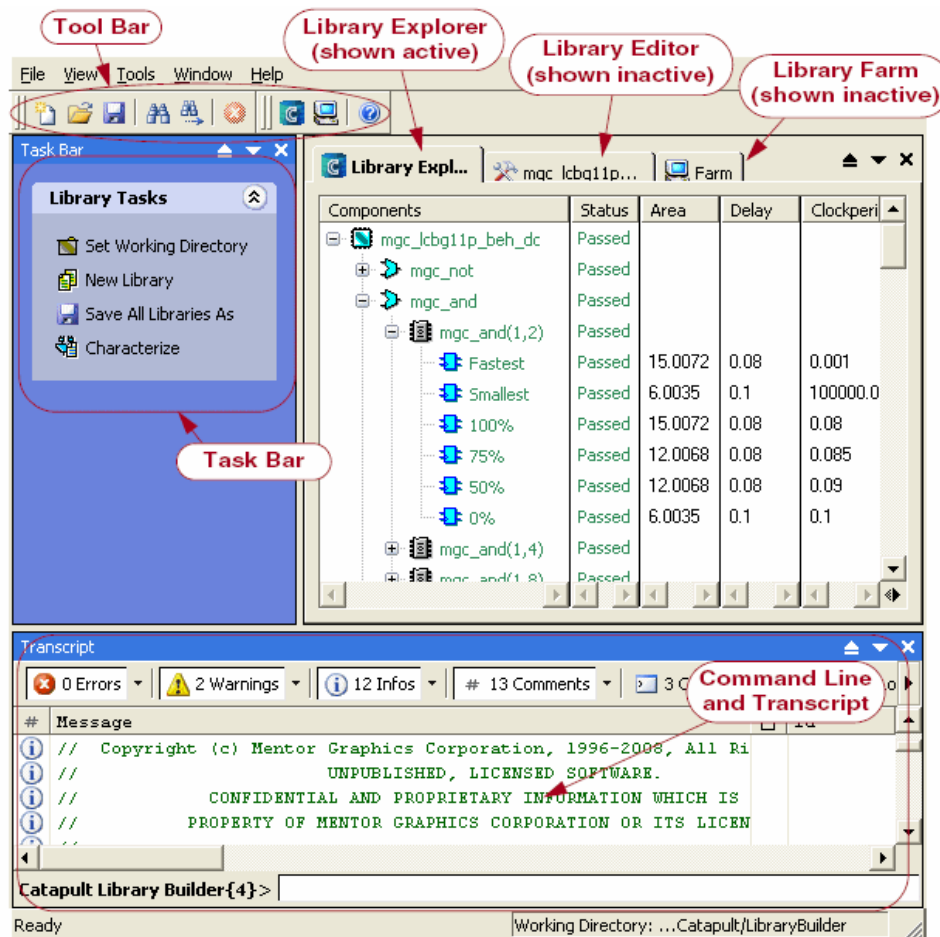
At the top of the session window is a set of pulldown menus and a tool bar for quick access to commonly used commands. All window can be moved, resized, undocked from the session window, or changed to/from a tabbed window. For a detailed discussion about how to rearrange the windows, refer to section “[Changing the Window Layout](#)” in the *Catapult C Synthesis User’s and Reference Manual*.

Note

For the purpose of this discussion, Figure 1-2 shows how the session window appears when all of its major features are enabled. Some features, such as the Farm window, are not visible by default in the initial invocation window. Refer “[Creating Custom Operators and Interfaces](#),” for information about using the Library Editor.

You can hide or show the **Toolbar** and **Status Bar** by selecting the corresponding item on the **View** pulldown menu. The View menu is also used to redisplay other windows that have been hidden.

Figure 1-2. Library Builder Session Window



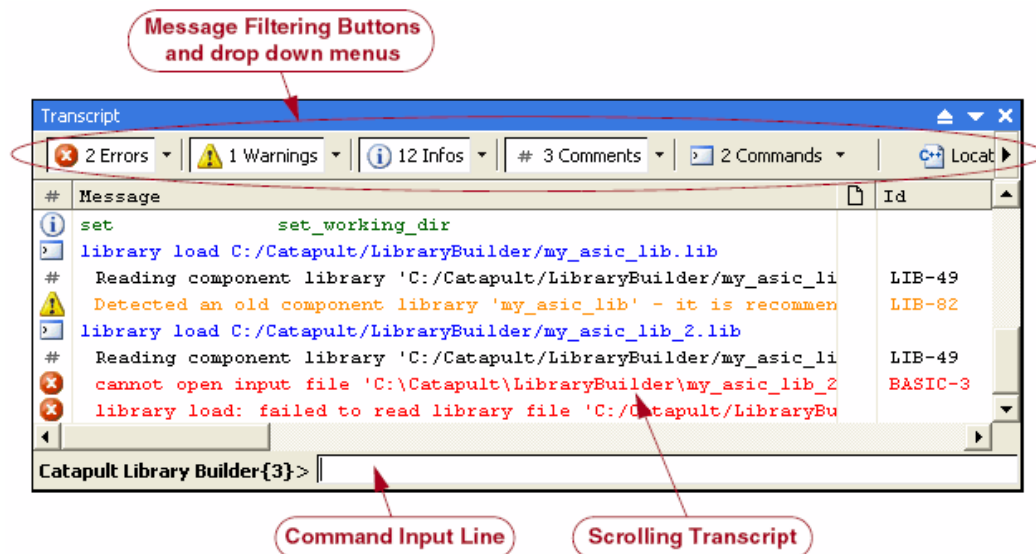
The Task Bar Window

The Task Bar is a convenient command interface to the primary Library Builder tasks (see Figure 1-2). By default, this window is visible at all times.

The Command Input and Transcript Window

This window, shown in Figure 1-3, accepts Tcl commands as input and it displays a scrolling transcript of all activity performed during the session. The message filtering buttons and drop down menus allow you to change the visibility of messages and search for messages by their severity classification.

Figure 1-3. Command Input and Transcript Window



The command input line is a Tcl command interpreter that can access the host operating system shell as well as input commands to Catapult. Refer to chapter 5, “[Commands](#)” for information about all Catapult Tcl commands and the Tcl interface. As commands are entered, a transcript of the commands and any system messages they generate are displayed in the transcript area of the window.

The label next to the command input area shows a count of the number of commands entered. The command input area provides context sensitive command completion. Pressing the TAB key will attempt resolve the text on the command line to a valid command name. One of three outcomes can result:

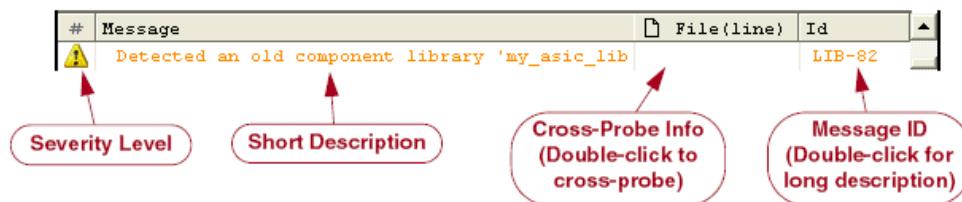
1. If the text can be uniquely matched to one command, that command replaces the text on the command line.
2. If it matches more than one command, the list of possible matches are displayed in the transcript.

3. If no match is possible, no change is made and an audible beep is generated.

Transcript Area

Transcript messages are color coded by type (severity classification). Error messages are red, warning messages are orange, informational messages are green, commands are blue and comments are black. Each message appears on a single line and has four parts, as shown in Figure 1-4. First is an icon that identifies its severity level. Second is the message text in a concise form. Third is cross-probe information consisting of the name and line number of the source code file related to the message. Fourth is the message identifier. For a detailed discussion about message identifiers and how to configure their severity levels, refer to “[Understanding Messages in the Transcript](#)” in the *Catapult C Synthesis User’s and Reference Manual*.

Figure 1-4. Parts of a Message in the Transcript Window



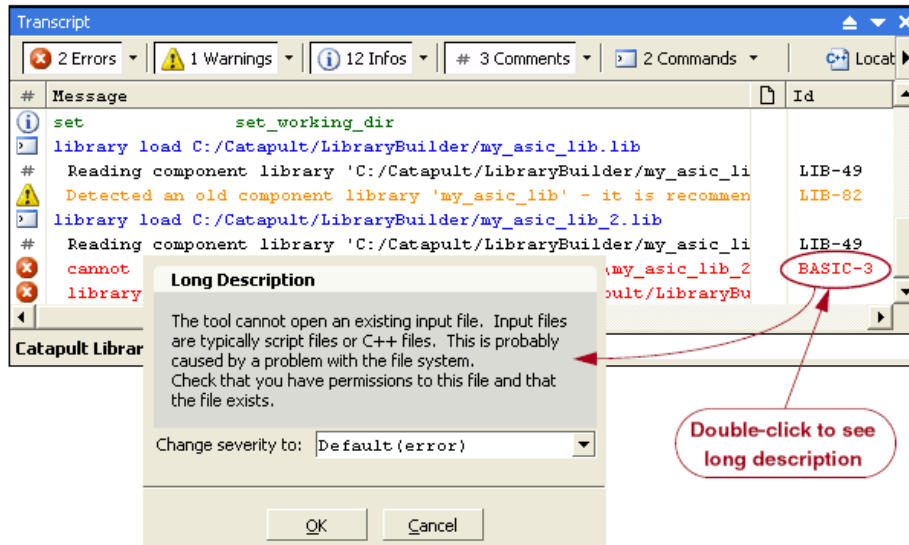
Viewing the Long Description

Message descriptions in the transcript are short and concise. Additional information is available for some messages by double-clicking on the message ID at the end of the line. As shown in Figure 1-5, that action opens the “Long Description” window. In addition to displaying the long description, the window also allows you to change the severity level of that message ID.

You can also use the “[help message](#)” command to display the long description of a message in the transcript window. For example, the following command will display the long description of the “CIN-6” comment:

```
help message CIN-6
# The pragma design top tells the tool where to start synthesis. This
# message tells you that the pragma has been detected.
```

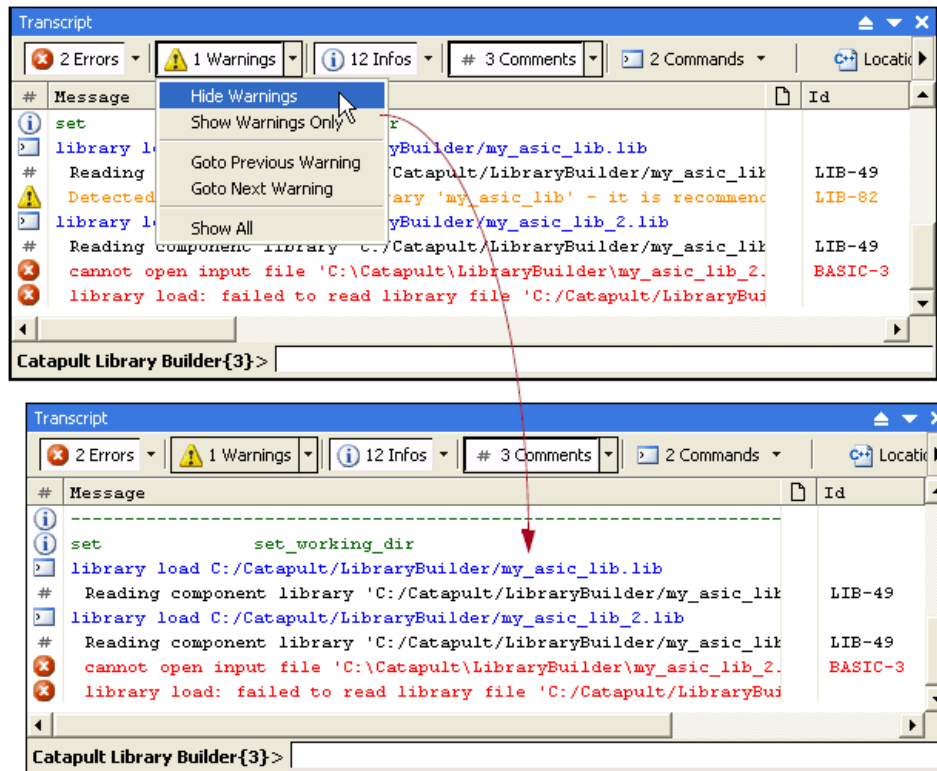

Figure 1-5. Viewing the Long Description of a Message



Filtering Buttons and Drop Down Menus

The message filtering buttons at the top of the window allow you to toggle the visibility of each message type. Each button also has a drop down menu that provides additional filtering and search options.

Figure 1-6. Filtering Comments Out of the Transcript (before and after)



The menu items are as follows (<msg_type> is either Errors, Warnings, Infos or Comments):

- Hide <msg_type> / Show <msg_type>
- Show <msg_type> Only
- Goto Previous <msg_type>
- Goto Next <msg_type>
- Show All
(Makes all messages visible regardless of <msg_type>)

The “Commands” button drop-down menu has the following items:

- Show Only Commands / Show All
- Show Hierarchy / Hide Hierarchy
- Goto Previous Command
- Goto Next Command

In “Show Hierarchy” display mode, all messages generated by a command are subordinate to that command. Each command can be individually expanded or collapsed to show or hide its subordinate messages.

The “Location” button toggles the cross-probe information to display a file icon instead of the file name and line number. The icon takes up less space on the line and allows more of the message description to be seen.

The Library Explorer Window

The Library Explorer window shown in [Figure 1-7](#) allows you to edit and characterize library modules.

Figure 1-7. Library Explorer Window

Library	Status	Area	Delay	IntPwr	LkgPwr	SwgPwr	Slack
mgc_sample-065nm-...	Passed						
mgc_not	Passed						
mgc_and	Passed						
mgc_and(1,2)	Passed						
mgc_and(1,4)	Passed						
mgc_and(1,8)	Passed						
mgc_and(1,12)	Passed						
Fastest	Passed	39.6	0.06	3.8764	0.238199	1.446	-0.06
Smallest	Passed	12.8	0.1	0.766871	0.0469254	0.142662	99999.9
100%	Passed	39.6	0.06	3.8764	0.238199	1.446	-0.06
75%	Passed	30.8	0.07	2.7175	0.162717	1.4984	0.0
50%	Passed	22.4	0.08	1.7066	0.0980857	0.881117	0.0
0%	Passed	12.8	0.1	0.766871	0.0469254	0.142662	99999.9
mgc_and(1,16)	Passed						
mgc_and(1,20)	Passed						
mgc_and(1,24)	Passed						

The following list describes the columns of the Library Explorer window:

- **Library** — Hierarchical display of the currently loaded libraries. Expand a library object to see the component modules (MODs), qualified modules (QMODS), and characterization data sets as shown in [Figure 1-7](#).

A MOD is a library component that implements one or more of the operators available to the Catapult scheduler. Memory is an example of a MOD implementing multiple operators, one operator for read and one operator for write.

A QMOD is a configuration of the module used during characterization to estimate the module (MOD) properties. Library Builder uses several QMODs during the characterization process.

Each QMOD contains relative characterization data sets. Each data set is created by using different constraints on the QMOD to determine the best QofR tradeoffs. For more information, see “[Multi-Point Characterization](#)” on page 76.

- **Status** — Characterization state of each item in the library.

Depending on the selected MOD type and characterization settings, the following columns display characterization results for the data sets. Click on the values to edit them.

- **Area** — Component area characterization.
- **Delay** — For sequential logic, this value is the output register to output value. For combinational logic, this value is the total input to output delay.
- **ClockPeriod** — Clock period characterization constraint.
- **MinClkPrd** — Minimum clock period is the fastest time the component can be scheduled. MinClkPrd only displays when a pipelined component is selected.
- **Slack** — ClockPeriod minus MinClkPrd.
- **InputDelay** — Input to register delay.
- **InputSetup** — Input register setup time.
- **R2RDelay** — Register to register delay.
- **R2RSetup** — Register to register setup time.

Note

For more information on delay properties, see “[Delay Characterization Properties](#)” on page 82.

- **IntPwr** — Internal Power.
- **LkgPwr** — Leakage power.
- **SwgPwr** — Switching power.
- **InputConstraint** — Input delay constraint.

Right-click on items in the Library column to display available commands. Depending on the type of object selected, a subset of the following commands displays:

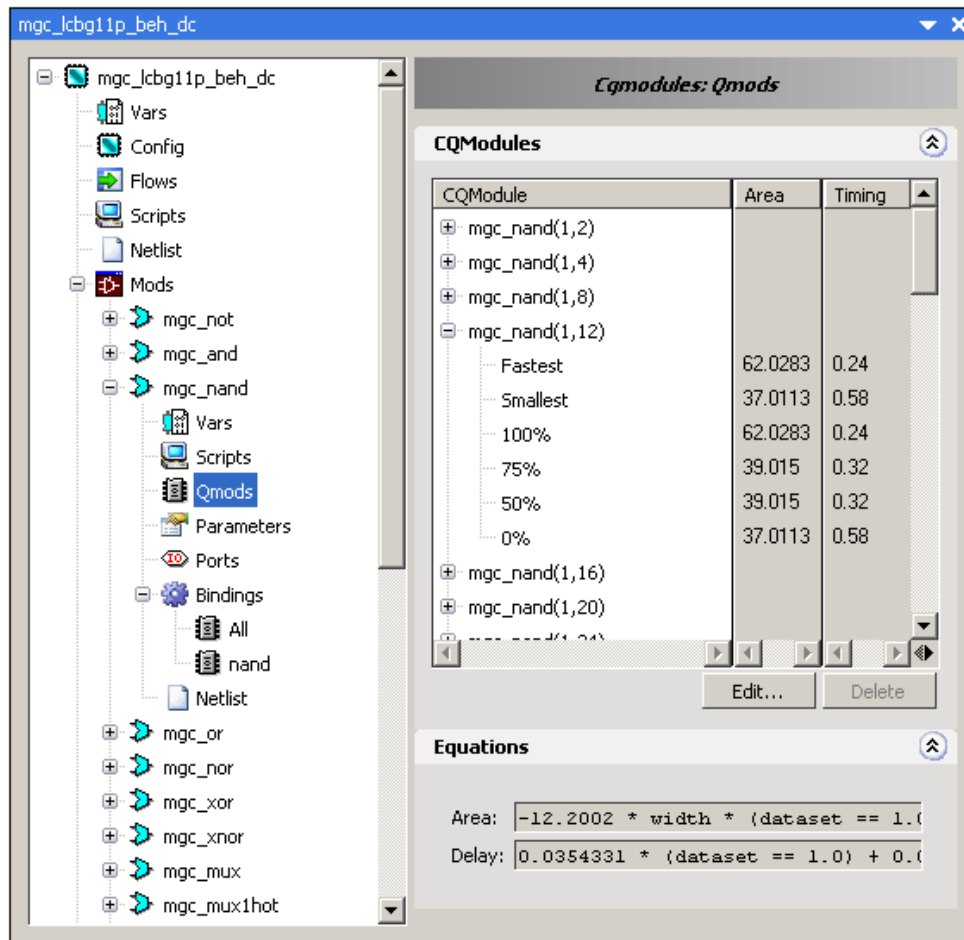
- **Characterize** — Launch the designated RTL synthesis tool to synthesize all of the qualified modules under the selected item. For example, if the library is selected, every QMOD in the library is characterized. If a MOD is selected, only the QMODs for that MOD are characterized. The resulting area and timing data reported by the RTL synthesis tool is stored in the library database. Refer to “[Library Characterization](#)” for more information about the characterization process.

- **Clean** — For all QMODs under the selected item, clears the characterization data from the database. This command resets the status to Pending for all affected QMODs.
- **Edit** — Opens the selected library in the Library Editor. Refer to “[Creating and Editing Libraries](#)” for more information about the Library Editor.
- **Save As...** — Saves the selected library to a file. The default file name is <lib_name>.lib. Refer to “[Saving Libraries](#)” for more information.
- **Save Characterization Commands/Data/Results As...** — Creates a Tcl command file that, when executed in the Library Builder tool, will reconstruct the database of selected library. In other words, this command exports the library database to an ASCII text format file.
- **Plot** — Graphs the characterization data of all QMODs under the selected module. Refer to “[Plotting the Characterization Data](#)” for more information.
- **Report** — Generates a report of the characterization data for selected module, or the entire library.
- **Properties** — Displays the name and working directory for the selected library.

The Library Editor Window

This window is a graphical editing environment that allows you to create and modify all elements in a library. As shown in Figure 1-9, the left side of the window is a hierarchical view of the library elements. The right side of the window is a context sensitive editor. The editor displays the appropriate edit interface for the type of library element that is selected in the hierarchy view. Refer to “[Creating Custom Operators and Interfaces](#),” for more information about how to create and modify library components,

Figure 1-8. Library Editor Window



The Farm Window

The Farm window, shown in Figure 1-9, is used for managing a network of computers that are available to run library characterization tasks. This window allows you to add, remove, and configure a set of host machines to which characterization jobs can be distributed. The **Host** column shows the list of computers available in the Farm and the number of tasks they can accept. The other columns report status and results of tasks that are running.

For more information about the Library Farm, refer to [“Using the Library Farm”](#) on page 90.


Figure 1-9. Library Farm Window

Hosts	Component	Pass/Fail	Status
localhost (1)		0 / 0	Idle
fusion (5)		0 / 0	Idle
Task 1		0 / 0	Idle
Task 2		0 / 0	Idle
Task 3		0 / 0	Idle
Task 4		0 / 0	Idle
Task 5		0 / 0	Idle
snape (10)		0 / 11	Running
Task 1		0 / 0	Idle
Task 2	mgc_add(20,1,4,1,21)[Fastest]...	0 / 1	Running
Task 3		0 / 0	Idle
Task 4	mgc_add(20,0,2,0,21)[Fastest]...	0 / 3	Running
Task 5	mgc_add(20,1,16,0,21)[Fastest]...	0 / 1	Running
Task 6	mgc_add(20,0,20,0,21)[Fastest]...	0 / 3	Running
Task 7		0 / 0	Idle
Task 8		0 / 0	Idle
Task 9		0 / 0	Idle
Task 10	mgc_add(20,0,8,1,21)[Fastest]...	0 / 3	Running

Getting Help

You can access help information by using either context-sensitive links to specific topics, or by using the **Help** pulldown menu to display information on the version of Library Builder or to display Library Builder documentation.

Context-Sensitive Online Help

- The *Help* button on dialog boxes opens help topics specific to the dialog box.
- The *help* button on the session window tool bar  opens help topics about the active window.
- Clicking on *message numbers* in the Transcript window opens a help message pop-up box. See “[Set Messages Options](#)” on page 27 for information about message numbers.

Command-line Help

- Enter a command name and the *-help* argument to display a listing of the options for that command along with brief descriptions.
- Enter the “[help command](#)” or “[help message](#)” commands.

Help Menu

- Select **Help > About Catapult Library Builder...** to display a Library Builder splash screen containing information on the Library Builder version.
- Select **Help > Open Manuals Bookcase** to access all of the Catapult C Synthesis product manual and release notes.

Setting Library Builder Options

This section describes how to configure default settings for the Library Builder system options. All options are initialized with factory default values the first time the software is installed. When you change a default setting, the new setting remains becomes the default for the duration of the Library Builder session. To preserve your new settings for future sessions, refer to [“Saving and Restoring Session Options”](#) on page 47.

To modify the system options, select **Tools > Set Options...** to open the Catapult Library Builder Options window, as shown in [Figure 1-10](#) on page 25, then select a category in the panel on the left to display the options dialog box for that category. You set a variety of options in the following categories:

- [“Set General Options”](#) on page 24
- [“Set Messages Options”](#) on page 27
- [“Set Component Library Options”](#) on page 31
- [“Set Catapult C Synthesis Options”](#) on page 31
- [“Set Farm Options”](#) on page 33
- [“Set Input Compiler Options”](#) on page 35
- [“View Compiler Settings”](#) on page 37
- [“Set Text Editor Options”](#) on page 37
- [“Set Flows Options”](#) on page 38
- [“Set Precision Flow Options”](#) on page 39
- [“Set Design Compiler Flow Options”](#) on page 41
- [“Set RTL Compiler Flow Options”](#) on page 43
- [“Set TalusDesign Flow Options”](#) on page 45

Set General Options

The General options dialog box, as shown in [Figure 1-10](#), allows you to specify default settings for the following features:

- **Startup Directory**

These options are used to set the default current working directory. If not set, then the current working directory is the directory from which the executable is called.

If Library Builder is running on a Windows operating system and you invoke the tool from the Start menu, you will probably want to enable this option. This will cause Library Builder to start up in the last directory in which it was started.

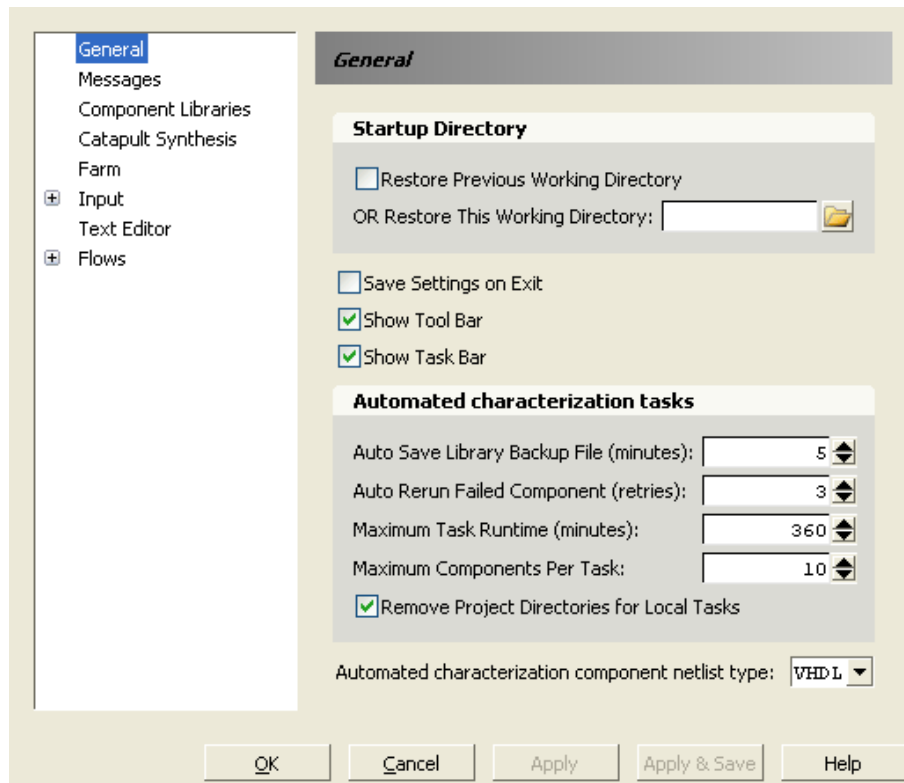
- **Restore Previous Working Directory**

Check this box to restore the working directory from the previous session. If this option is checked, the “**Or Restore This Working Directory**” setting is ignored. You can also set this option using the “`options set General RestoreCWD <true_or_false>`” command.

- **Or Restore This Working Directory:**

Use this option to set the default working directory to one specific directory, causing Library Builder to always start in that directory. Click the **Browse** icon to navigate to the directory that you want to use on startup. You can also set this option using the “`options set General StartInWD <path>`” command.

Figure 1-10. Set Options General Menu



- **Save Settings on Exit**

Check this box to save options settings on exit. The options are saved into the [Catapult C Library Builder Registry](#). This insures that the options settings will appear each time the tool is invoked, regardless of the invocation directory. You can also set this option using the “`options set General SaveSettings<true_or_false>`” command. For more information about saving option settings and about the Catapult registry, refer to the section “[Saving and Restoring Session Options](#)” on page 47.

- **Show Tool Bar**

Uncheck the box to hide the session window Tool Bar on startup. The view menu can be used to display the Tool Bar if it is hidden. You can also set this option using the “`options set General ShowToolBar <true_or_false>`” command.

- **Show Task Bar**

Uncheck the box to hide the Task Bar window on startup. The view menu can be used to display the Task Bar when it’s hidden. You can also set this option using the “`options set General ShowTaskBar <true_or_false>`” command.

- **Automated characterization tasks**

These options are used to set default parameters for the library characterization tasks:

- **Auto Save Library Backup File (minutes)**

Specify the time interval in minutes between each automatic backup of the characterization library. A backup saves the in-memory library to the file `<working_dir>/<lib_name>.char/<lib_name>.wlib`. In the event that you need to recover the backup library, open the library file with the “`library load -recover`” command, which will automatically search for and load the backup files. Alternatively, you can use the **File > Open Library...** menu item to open a backup file directly. You can also set this option by using the “`options set General AutoSaveLibraryBackup <integer>`” command.

- **Auto Rerun Failed Component (retries)**

Specify the number of times Library Builder should attempt to characterize a component that fails to characterize. A setting of zero means no retries will be attempted. If the characterization fails after the specified number of attempts, the component is given a failed status in the library. Invocation failures do not count against the retry count. A component group that times out will be unbundled for the retry.

You can also set this option by using the “`options set General AutoRerunFailedComponent <integer>`” command.

- **Maximum Task Runtime (minutes)**

Specify the maximum amount of time in minutes that each characterization task is allowed to run. Adjust the number of minutes to suit the performance the target host machine. Slower machines need a higher value. Multipliers and modulus require the

most time. If you increase/decrease the “Maximum Components Per Task” setting (see below), you should increase/decrease this setting proportionally. A setting of zero means unlimited runtime. You can also set this option by using the “`options set General MaxTaskRunTime <integer>`” command.

- **Maximum Components Per Task**

Specifies the maximum number of components that can be characterized in a single task. Because each task must invoke a downstream synthesis tool, increasing the number of components per task will reduce the total number of tool invocations across the library. If you increase/decrease the “Maximum Task Runtime (minutes)” setting (see above), you should increase/decrease this setting proportionally. You can also set this option by using the “`options set General MaxComponentsPerTask <integer>`” command.

- **Remove Project Directories for Local Tasks**

A temporary project directory is created for each characterization task (`<working_dir>/<lib_name>.char/<task_name>.proj`). By default, each directory is automatically deleted when the task completes successfully. If any component within the task fails, the project directory is not deleted. Uncheck this option to preserve all project directories regardless of their pass/fail status. You can also set this option using the “`options set General RemoveProjectDirectories <true_or_false>`” command.

- **Automated characterization component netlist type**

Select the netlist format, either **VHDL** or **Verilog**, that will be sent to the downstream synthesis tool. You should select your primary downstream language. You can also set this option using the “`options set General NetlistFormat <VHDL_or_Verilog>`” command.

Set Messages Options


This set of options allows you to override the default severity level of system messages reported by Library Builder. All messages are assigned one of the following severity levels:

- **Error** — Problem that causes Catapult to fail.
- **Warning** — Problem that should be examined.
- **Info** — Informational message.
- **Comment** — Status message.
- **Unclassified** — All other messages.

Using the **Messages** options dialog box, as shown in Figure 1-11, you can change the factory default severity level of individual messages. For example, you can raise the severity of a warning message from **Warning** to **Error**.

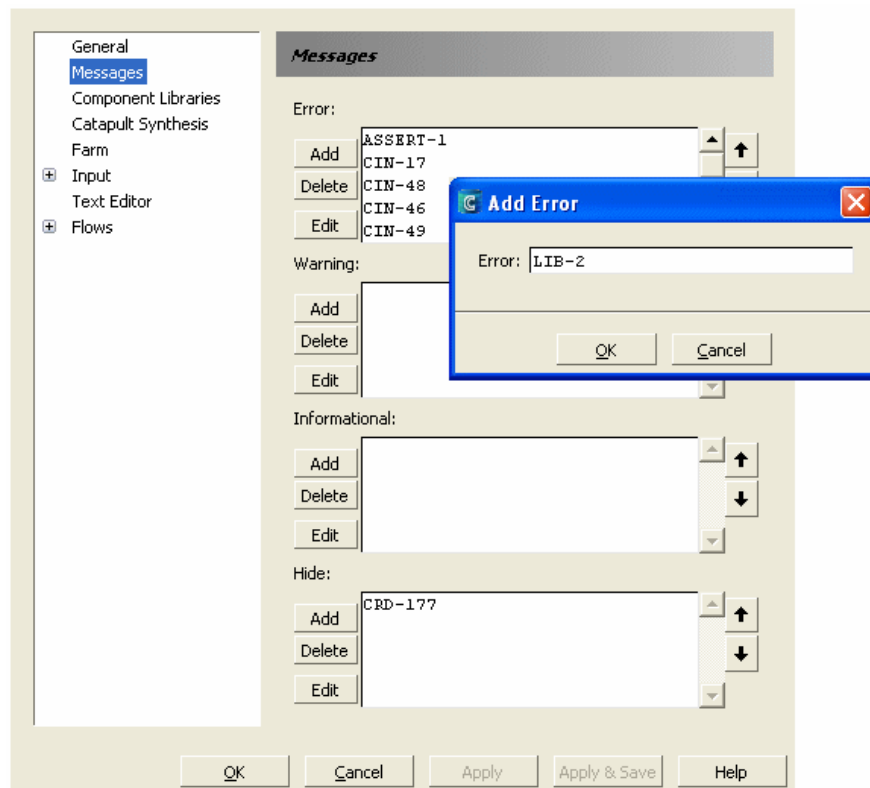
For each severity level, Library Builder provides an override list to which you can add or delete *message numbers* (unique identifiers). The **Messages** dialog box provides an easy interface for modifying these override lists. For each list, use its Add, Edit and Delete buttons to alter its set of messages. For details about how to identify and specify message numbers, refer to the sections “[Message Classifications and Conventions](#)” on page 29, and “[Naming Convention for Messages](#)” on page 30.

Note

 Catapult C Library Builder does not let you create new messages or delete existing system messages. Adding or deleting simply changes the set of messages in the severity level override lists.

You can modify message lists from the command line using the `options set` command with one of the following options `Message/ErrorOverride`, `Message/WarningOverride`, `Message/InformationalOverride`, or `Message/Hide`. These commands can change the priority of a message by adding it to a different list. If a message belongs to multiple lists, then the higher priority list is used. This means that a message may never be downgraded. This command will check if the options are valid and report an error if they are not.

Figure 1-11. Messages Options Dialog Box



- **Adding a Message to a List**

Click the **Add** button next to a message category (Error, Warning, Informational, or Hide) and the Add dialog box displays. Type the ID code of the message that you want to add and click **OK** to accept the entry. The message ID list in the Messages options dialog box updates when you OK the Add dialog box.

- **Editing a Message from a List**

Select the message number you want to edit from the Error, Warning, Informational, or Hide lists and click the **Edit** button to open the Edit dialog box. Modify the value and click **OK** to accept the change.

- **Deleting a Message from a List**

Select the message number you want to delete from the Error, Warning, Informational, or Hide lists and click the **Delete** button. The message number is removed indicating it has been deleted from that list.

The library database is not updated until either the **OK** button or **Apply** button on the Message options dialog box is clicked. At that point, Library Builder reflects the changes in the transcript window, as shown in the example below.

Message Classifications and Conventions

The Library Builder messages are classified by the context in which they appear in the tool. Table 1-1 describes the meaning of the *context codes* that appear at the end of messages in the transcript.

Table 1-1. Types of Library Builder Message

Context Code	Description
Flow Messages	
CRD	C reading
CIN	C SIFgen
HIER	Hierarchical
LOOP	Loop
MEM	Memory and Interface Mapping
ALOC	Allocation
SCHD	Scheduling
FSM	FSM Extraction + Reg Sharing
ASG	Assignment - Component binding and Sharing
Optimization Messages	
OPT	Sequential Design Analysis + Other optimizations

Table 1-1. Types of Library Builder Message (cont.)

Context Code	Description
Other Messages	
BASIC	Low level
VHDL	VHDL Netlisting
VLOG	Verilog Netlisting
NL	General Netlisting
PRJ	Project
SOL	Solution
LIB	Library
LIC	License
READ	SIF Reading
WRITE	SIF Writing
SIFG	SIF Gen (VHDL and Verilog)
SEQ	Sequential Component Analysis
CNS	Constraint Management

Naming Convention for Messages

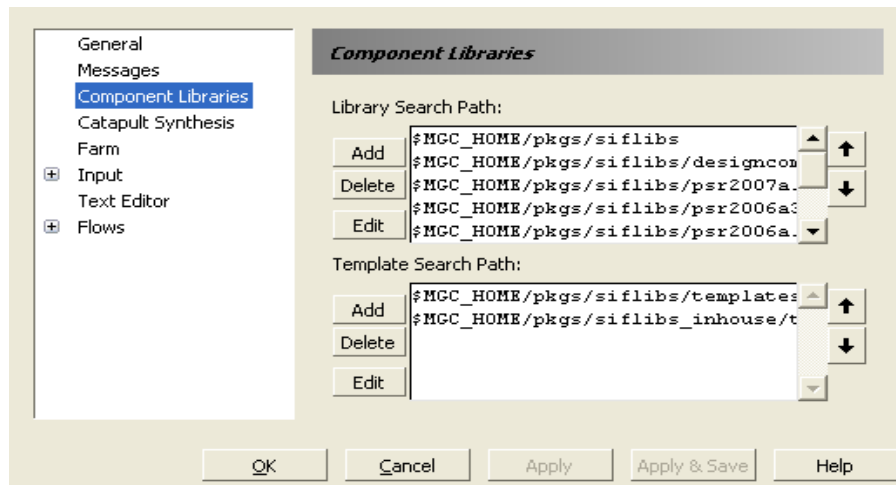
Each qualifying Message in Library Builder is named with the character string for its classification followed by a unique number related to that character string. The message number displays in the transcript window as shown below:

```
# Loading options from registry.  
library load C:/Catapult/my_lib.lib  
# Reading component library 'C:/Catapult/my_lib.lib'... (LIB-49)  
# /LIBS/my_lib
```

The message number associated with the message is at the end of the line. It consists of a context code and a number. In the example above, the tool displays the message number (LIB-49), which is an informational message about reading a component library. For warning or error messages, the word Warning or Error appears at the beginning of the message. Double-click on a message number to display additional help associated with the message.

Set Component Library Options

Figure 1-12. Set Options Component Library



- **Library Search Path**

This is the Library Search path. You can also set this option using the “`options set ComponentLibs SearchPath <list_of_paths>`” command.

- **Template Search Path**

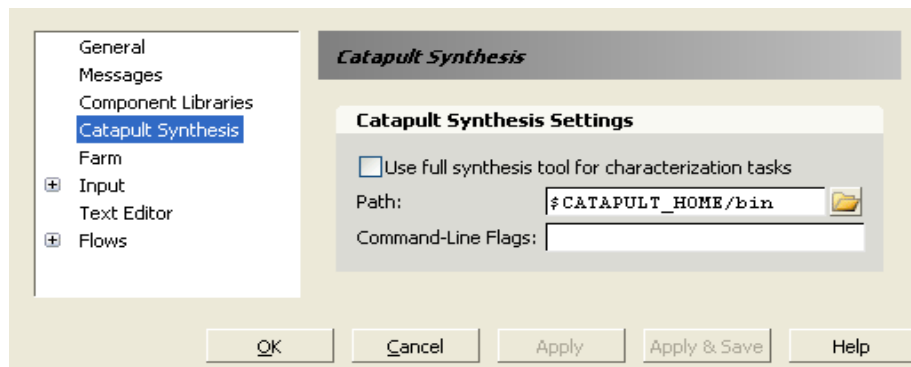
This is the search path used by Library Builder for the templates used to create libraries. You can also set this option using the “`options set ComponentLibs TemplateSearchPath <list_of_paths>`” command.

Use the **Add**, **Edit** and **Delete** buttons to modify the paths in the list. Use the up and down arrows to move a selected path higher or lower in the search order. Click **OK** and changes take effect immediately. To save changes to the area where options were originally read, choose **Tools > Save Options**.

Set Catapult C Synthesis Options

This set of options allow you to customize the default invocation string for the supported synthesis tools.

Figure 1-13. Set Options Catapult C Synthesis Dialog



Catapult Synthesis Settings

- **Use full synthesis tool for characterization tasks**

Enabling this option allows the Library Builder to use Catapult C Synthesis licenses for running characterization tasks. By default, the Library Builder uses dedicated licenses explicitly for characterization tasks. Each Catapult C Synthesis license that is in use by a characterization task will be unavailable to the Catapult C Synthesis tool until the characterization task is complete.

You can also set this option using the “`options set CatapultC UseFullSynthesisTool <true_or_false>`” command.

- **Path**

This path will be checked first to find the Catapult C Synthesis executable. If none is found, then on Unix, the standard `$PATH` variable will be checked. On Windows, the registry will be checked followed by the standard path.

This field will only evaluate the `CATAPULT_HOME` environment variable. You can not use any other environment variable in the path. You can omit the `CATAPULT_HOME` variable and specify an absolute pathname.

Click the **Browse** icon to browse to the directory where Catapult software is installed. You can also set this option using the “`options set CatapultC Path <path>`” command.

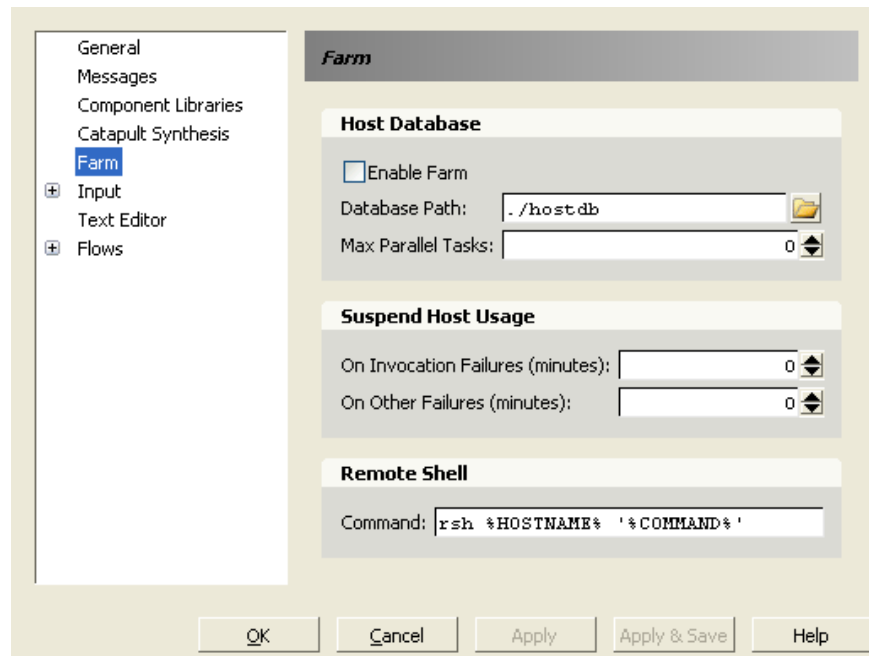
- **Command Line Flags**

This displays the command line flags set for the Catapult C Synthesis tool. You can also set this option using the “`options set CatapultC Flags <list_of_flags>`” command.

Set Farm Options

These options are global settings that apply to all Library Farm hosts and tasks. For information about adding hosts and configuring tasks, refer to “Using the Library Farm” on page 90.

Figure 1-14. Library Farm Set Up



Host Database

- **Enable Farm**

Check this box to enable the Library Farm feature. You can also set this option using the “`options set Farm EnableFarm <true_or_false>`” command.

- **Database Path**

Select the directory containing Library Farm information. Host information will be stored here by the Library Builder. Multiple Library Builders can use the same directory, they will share the available tasks. Click on the **Browse** icon to browse the file system the host database path. You can also set this option using the “`options set Farm HostDatabasePath <path>`” command.

- **Max Parallel Tasks**

Use the scroll arrows to select the maximum number of tasks to run in parallel across all hosts in the farm. Library farm will only run this number of tasks even if more hosts are available in the farm. For example, suppose you have 15 Catapult licenses available, and 20 hosts in your farm. You should set this option to 15 or fewer. Setting it to fewer will keep some licenses available for other to use.

A zero in this field means that tasks are unlimited. You can also set this option using the “`options set Farm MaxParallelTasks <integer>`” command.

Suspend Host Usage

- **On Invocation Failures (minutes)**

Specify the length of time in minutes to suspend usage of a host if the downstream synthesis tool fails to invoke. The host is reactivated after the waiting period. You can also set this option using the “`options set Farm SuspendHostOnInvocationFailure <integer>`” command.

- **On Other Failures (minutes)**

Specify the length of time in minutes to suspend usage of a host if that host fails for reasons other than a failure of the downstream synthesis tool invocation. The host is reactivated after the waiting period. You can also set this option using the “`options set Farm SuspendHostOnFailure <integer>`” command.

Remote Shell

- **Command**

Displays the remote shell command associated with the library farm. You can also set this option using the “`options set Farm RshCommand <rsh_cmd_expr>`” command.

The default command is `rsh`. The internal variables `%HOSTNAME%` and `'%COMMAND%'` are passed as parameters to the remote shell command. The `%HOSTNAME%` parameter contains the name of the target host machine. The `%COMMAND%` parameter contains the library characterization command.

The Remote Shell Command supports the following internal variables:

- **%COMMAND%**
This variable contains the Catapult C Synthesis invocation command, which consists of the path to the `catapult` executable, its command line flags, and the `-shell` option. The values held in the `%COMMAND%` variable are set in the Catapult C Synthesis options dialog. Refer to “[Set Catapult C Synthesis Options](#)” on page 31 for more information.

Note



When the Library Builder is configured to use dedicated licenses for characterization tasks instead of Catapult C Synthesis licenses, you must use the `%COMMAND%` variable in the Remote Shell Command field. Do not enter a literal invocation command line. The variable automatically supplies special command-line flags that are required for the dedicated license.

When used without quotes, each string in the variable is evaluated as separate arguments. For example, if the Catapult C Synthesis Flags option contained the

string “-mgls_license_file 123@licserver,” the variable would expand to the following set of strings:

```
<Catapult_Install>/bin/catapult -mgls_license_file 123@licserver  
-shell
```

When enclosed in double quotes (“%COMMAND%”), the entire contents of the variable is appended to the remote shell command as a single argument. Use the quoted form if the remote shell command is the “rsh” command, which expects the command to be a single argument.

- **%HOSTNAME%**
Evaluates to the target ‘Host’ machine as specified in the Farm tab of the GUI.
- **%COMMANDFILE%**
Evaluates to the command file name to be evaluated by Catapult C Synthesis to synthesize the design and gather characterization details. If %COMMANDFILE% is not specified, the command file will be piped to the standard input of the application.
- **%OUTPUTFILE%**
Evaluates to the transcript file to be generated by Catapult C Synthesis. If %OUTPUTFILE% is not specified, the standard output from the application will be captured in the output file.
- **%CWD%**
Evaluates to the OS-specific form of the pathname of the working directory for the characterization job.

The above variables may be specified zero or more times. In addition, they can be embedded inside of other arguments in the Remote Shell Command. For example, if the Remote Shell Command is:

```
Launch -working_dir=%CWD% -exec %COMMAND%
```

Then the expanded command would be:

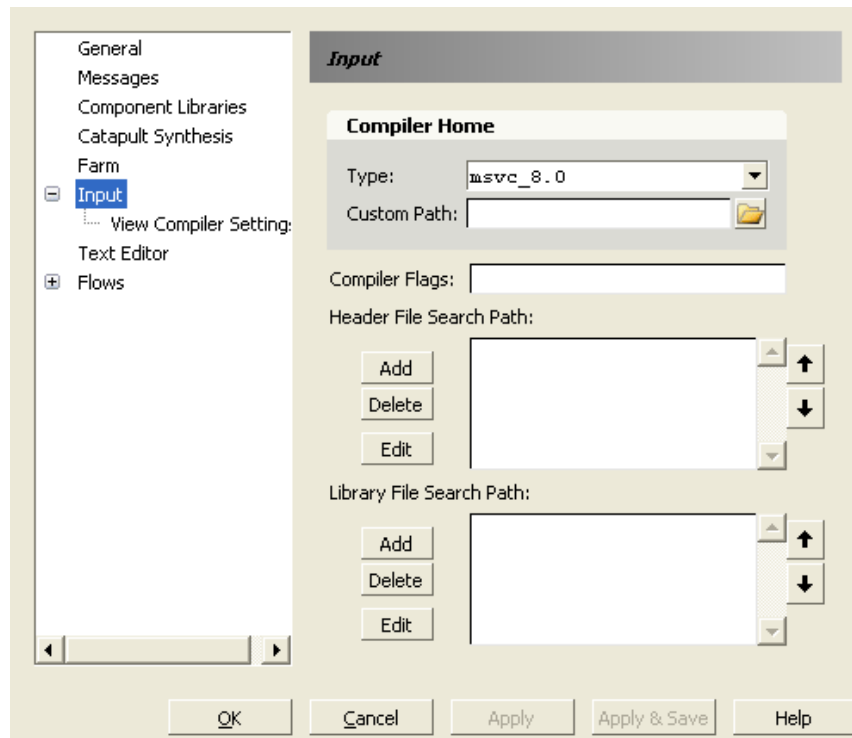
```
Launch -working_dir=/tmp/workdir  
-exec <Catapult_Install>/bin/catapult  
-shell
```

Enter Library Farm set up information and click **OK** or **Apply** to accept the Library Farm settings or **Cancel** to close without saving. See also [“Configuring Library Farm to Use the Load Sharing Facility \(LSF\) software”](#) on page 91.

Set Input Compiler Options

Use the Input dialog box to specify default compiler settings. You can specify a compiler for Catapult to use, compiler command flags, and search paths for header files and library files.

Figure 1-15. Input Options Dialog Box



- **Compiler Home**

The Compiler Home options specify the location of the compiler Catapult will use.

- **Type:** This field provides a list of the compilers Catapult finds installed on your system. On UNIX/Linux systems, the compiler installed in `$MGC_HOME/bin` is listed. On Windows systems, Microsoft compilers found in the Windows registry are listed, followed by the compiler supplied with ModelSim, if installed.

You can either select a compiler from the list, or you can select the **Custom** option to specify a different compiler. You can also set this option using the “`options set Input Compiler <name_or_custom>`” command.

This field works in conjunction with the [View Compiler Settings](#) page described in the next section.

- **Custom Path:** This field is used only when the “Custom” choice is selected in the Type field. Enter the path to top-level directory where the compiler is installed. You can either type the path in the text box or use the file system browser (click on the icon to the right of the text box). You can also set this option using the “`options set Input CompilerHome <path>`” command.

- **Compiler Flags**

Command line flags passed to the compiler during C compilation. Compiler options can also be set by right-clicking on the input file. You can also set this option using the “`options set Input CompilerFlags <list_of_flags>`” command.

- **Header File Search Path**

This search path will be added to the default search path for the Catapult compiler. Use the **Add**, **Edit** and **Delete** buttons to modify the list of directories in the search path. You can also set this option using the “`options set Input SearchPath <list_of_paths>`” command.

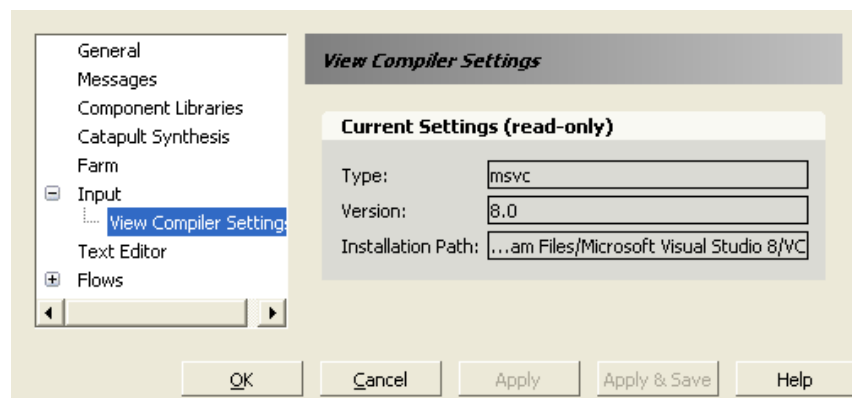
- **Library File Search Path**

This search path will be added to the default search path for the Catapult compiler. Use the **Add**, **Edit** and **Delete** buttons to modify the list of directories in the search path. You can also set this option using the “`options set Input LibPaths <list_of_paths>`” command.

View Compiler Settings

The **View Compiler Settings** page works in conjunction with the “Compiler Home” field on the “Input” options dialog box. This read-only page shows information about the compiler selected in the Compiler Home field. If the “Custom” option is selected and the “Custom Path” value is not valid, the fields on this page will be blank.

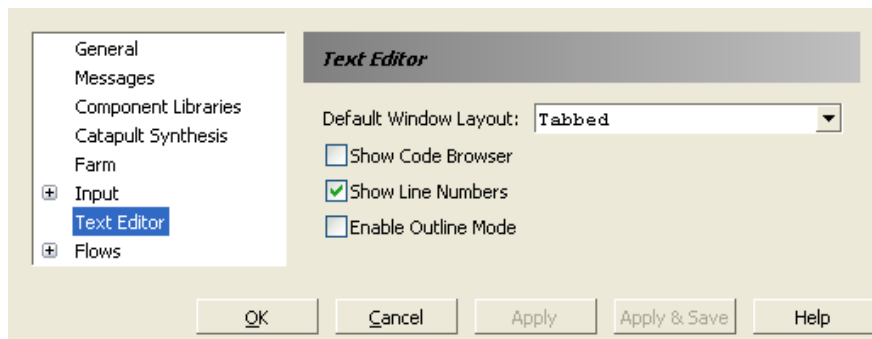
Figure 1-16. Compiler Settings Page



Set Text Editor Options

The Text Editor dialog box lets you set options for the DesignPad editor. You can override the default settings by using the popup menu in the DesignPad editor window.

Figure 1-17. Set Options Text Editor Menu



- **Default Window Layout**

Specify the default type of window to be used for Text Editor documents. Select either tabbed document window or a free floating window. You can also set this option using the “`options set TextEditor WindowLayout <Floating_or_Tabbed>`” command.

- **Show Code Browser**

Specify whether or not the Code Browser feature of the DesignPad editor is visible by default. The code browser allows you to move to selected sections in the code. You can also set this option using the “`options set TextEditor CodeBrowser <true_or_false>`” command.

- **Show Line Numbers**

Specify whether or not the DesignPad editor displays line numbers by default. You can also set this option using the “`options set TextEditor LineNumbers <true_or_false>`” command.

- **Enable Outline Mode**

Specify whether or not the Outline mode is enabled by default in the DesignPad editor. Outline mode provides a hierarchical view of the text in the editor. It allows you collapse/expand sections of the document independently. Outline mode is indicated by plus or minus icons to the left of a text block. Click a plus/minus icon to expand/collapse a block. You can also set this option using the “`options set TextEditor OutlineMode <true_or_false>`” command.

Click **OK** to accept the setting or **Cancel** to close without saving the setting.

Set Flows Options

The Flows dialog box allows you to configure the search paths for user-defined flow. Refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual* for information about user-defined flows.

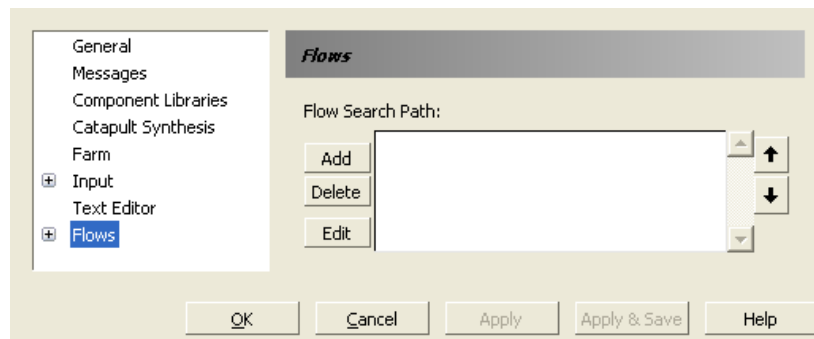
- **Flow Search Path**

Add and delete directory pathnames to user-defined flow package files. During startup, Catapult scans all of the flow package files (.flp) in the Flows Search Path and constructs an index of the packages it finds. The paths are searched in the order in which they appear. If duplicate filenames are found, only the first one is indexed. The default Catapult flow packages are scanned prior to user-defined flow packages.

Use the **Add**, **Edit** and **Delete** buttons to modify the paths in the list. Use the up and down arrows to move a selected path higher or lower in the search order.

You can also set this option using the “`options set Flows FlowSearchPath <list_of_paths>`” command.

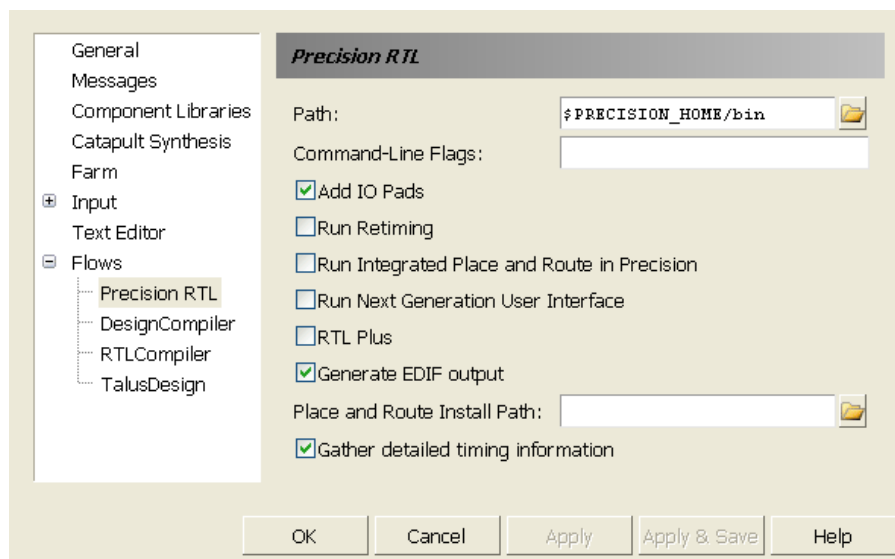
Figure 1-18. Flow Options



Set Precision Flow Options

Set the default library characterization values for the Precision RTL Synthesis flow.

Figure 1-19. Precision Flow Options



- **Path**

Specifies a search path for the Precision RTL Synthesis executable. If none is found, then the standard UNIX \$PATH variable is searched. On Windows, the registry is checked followed by the standard path. Click the folder icon to browse to the directory where the Precision executable is installed. You can also set this option using the “`options set Flows/Precision Path <path>`” command.

This field only evaluates the PRECISION_HOME environment variable. You can not use any other environment variable in the path. You can omit the PRECISION_HOME variable and specify an absolute pathname.

- **Command Line Flags**

Specifies command line switches for the Precision RTL Synthesis executable. You can also set this option using “`options set Flows/Precision Flags <list_of_flags>`” command.

- **Add IO Pads**

Specifies Precision RTL Synthesis option: Optimization - Add IO Pads. You can also set this option using the “`options set Flows/Precision addio <true_or_false>`” command.

- **Run Retiming**

Specifies Precision RTL Synthesis option: Retiming. You can also set this option using the “`options set Flows/Precision retiming <true_or_false>`” command.

- **Run Integrated Place and Route in Precision**

Enables the Precision RTL Synthesis option to automatically place and route the design. You can also set this option using the “`options set Flows/Precision run_pnr <true_or_false>`” command.

- **Run Next Generation User Interface**

Enables the new Precision RTL Synthesis graphical user interface. When this option is disabled, the old GUI is used. You can also set this option using the “`options set Flows/Precision newgui <true_or_false>`” command.

- **RTL Plus**

Runs the RTL Plus version of Precision Synthesis. Precision RTL Plus performs physically aware synthesis. You can also set this option using the “`options set Flows/Precision rltplus <true_or_false>`” command.

- **Place and Route Install Path**

Specifies the path to the default place and route tool. You can also set this option using the “`options set Flows/Precision PlaceAndRouteInstallPath <path>`” command.

- **Output File Folder Name**

Specifies the default name of the output file folder that appears in the Catapult C Synthesis GUI. The hierarchical path to the folder is “<solution_name>/Synthesis/<folder_name>”. You can also set this option using the “`options set Flows/Precision FOLDERNAME <string>`” command. The factory default value for this field is “Precision.”

- **Gather detailed timing information**

Includes the following types of timing data in the Precision timing report:

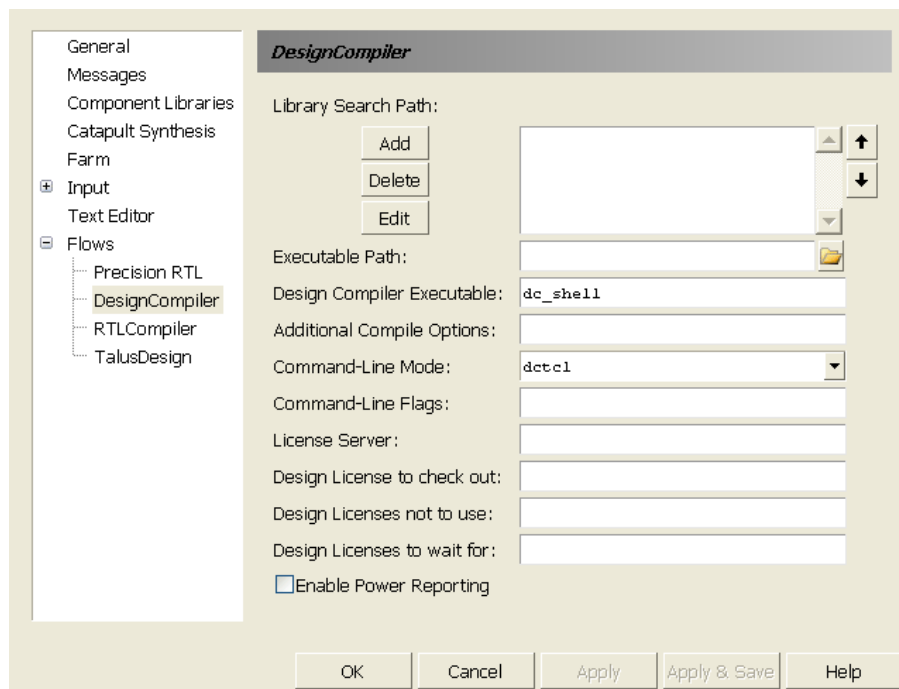
- input to register
- register to register
- register to output

You can also set this option using the “`options set Flows/Precision GatherDetailedTimingData <true_or_false>`” command.

Set Design Compiler Flow Options

Sets the default library characterization values for the Design Compiler (DC) flow.

Figure 1-20. Design Compiler Flow Options



- **Library Search Path**

Adds/deletes pathnames to DC library directories. When DC is invoked, it searches the specified paths in addition to its default library search paths. Use the Add, Edit and Delete buttons to modify the paths in the list. Use the up and down arrows to move a selected path higher or lower in the search order. You can also set this option using the “`options set Flows/DesignCompiler SearchPath <list_of_paths>`” command.

- **Executable Path**

The path to the bin directory containing the DC executable. This search path is checked first, and if the executable is not found, the standard path is checked. Click the folder button to navigate to the desired path. You can also set this option using the “`options set Flows/DesignCompiler Path <path>`” command.

- **Design Compiler Executable**

This option specifies the name of the DC tool to invoke. The default tool is `dc_shell`. You can also set this option using the “`options set Flows/DesignCompiler ShellExe <string>`” command.

- **Additional Compile Options**

Command line switches for the DC executable. You can also set this option using the “`options set Flows/DesignCompiler CompileOpts <list_of_options>`” command.

- **Command-Line Mode**

Enter the environment mode to run the DC executable in. The supported modes are `detcl` and `dcsd`. You can also set this option using the “`options set Flows/DesignCompiler ShellType <mode>`” command.

- **Command Line Flags**

This displays the command line flags set for DC. You can also set this option using “`options set Flows/DesignCompiler Flags <list_of_flags>`” command.

- **License Server**

List of license servers for DC licenses. Catapult assigns the specified list to the FLEXnet environment variable `SNPSLMD_LICENSE_FILE` prior to launching DC. Refer to the [FLEXnet Licensing End User Guide](#) for information about the syntax of FLEXnet environment variables. You can also set this option using “`options set Flows/DesignCompiler LicenseServer <list_of_servers>`” command.

- **Design License to check out**

Specifies a list of Synopsys DesignWare license features to be obtained. The list items are space separated. If the value is set, it is passed directly to the DC command “`get_license`” in the generated DC script so the specified license features are checked out at the beginning of synthesis. (They are held until the `remove_license` command is used

or until the program is exited or until the DC shell is closed.) Refer to the license key file at your site to determine which licensed features are available.

This option overrides any default settings defined in Catapult libraries. For more information about DesignWare settings in Catapult libraries, refer to “The Library Options Tab.”

- **Design Licenses not to use**

Specifies a list of DesignWare licenses that the DC tool is not allowed to use. The list items are space separated. The list is assigned to the DC variable “synlib_dont_get_license” in the generated DC script.

This option overrides any default settings defined in Catapult libraries. For more information about DesignWare settings in Catapult libraries, refer to “The Library Options Tab.”

- **Design Licenses to wait for**

Specifies a list of DesignWare licenses that the DC tool should wait for if they are temporarily unavailable. The list items are space separated. The value of this variable is assigned to the DC variable “synlib_wait_for_design_license” in the generated DC script.

This option overrides any default settings defined in Catapult libraries. For more information about DesignWare settings in Catapult libraries, refer to “The Library Options Tab.”

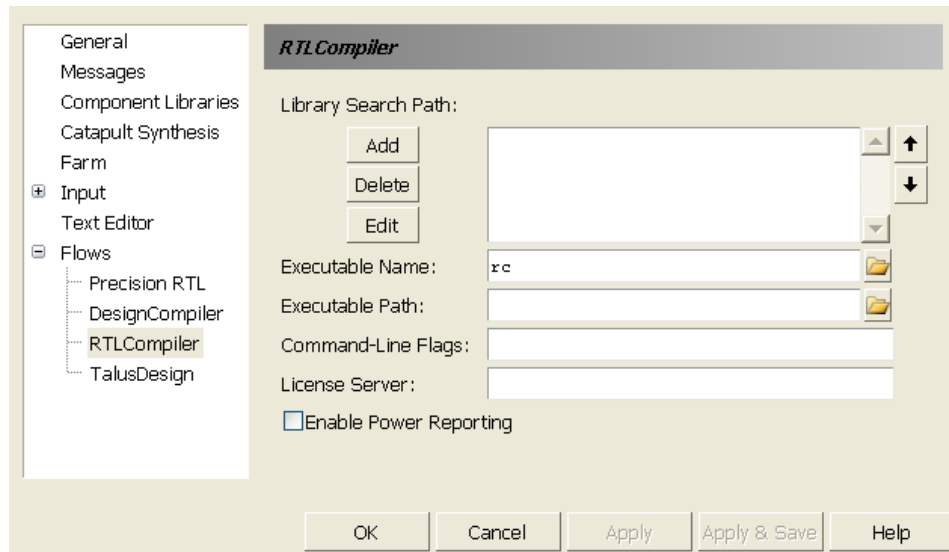
- **Enable Power Reporting**

Directs the DC to generate a power analysis report. You can also set this option using the “`options set Flows/DesignCompiler EnablePowerReporting <true_or_false>`” command.

Set RTL Compiler Flow Options

Sets the default library characterization values for the RTL Compiler flow.

Figure 1-21. RTL Compiler Flow Options



- **Library Search Path**

Adds/deletes pathnames to directories containing RTL Compiler libraries. Use the Add, Edit and Delete buttons to modify the paths in the list. Use the up and down arrows to move a selected path higher or lower in the search order.

You can also set this option using the “`options set Flows/RTLCompiler SearchPath <list_of_paths>`” command.

- **Executable Name**

Specifies the name of the RTL Compiler tool to invoke. The default tool is rc. You can also set this option using the “`options set Flows/RTLCompiler ShellExe <string>`” command.

- **Executable path**

Specifies the full path to the directory containing the RTL Compiler executable. Click the folder button to browse to the directory where software is installed. You can also set this option using the “`options set Flows/RTLCompiler Path <path>`” command.

- **Command-Line Flags**

Specifies additional command line flags to pass to RTL Compiler. You can also set this option using the “`options set Flows/RTLCompiler Flags <list_of_flags>`” command.

- **License Server**

Lists the license servers for the RTL Compiler licenses. List items are separated by spaces. Catapult assigns the specified list to the FLEXnet environment variable `CDS_LIC_FILE` prior to launching the RTL Compiler. Refer to the [FLEXnet Licensing End User Guide](#) for information about the syntax of FLEXnet environment variables. You can also set this option using “`options set Flows/RTLCompiler LicenseServer <list_of_servers>`” command.

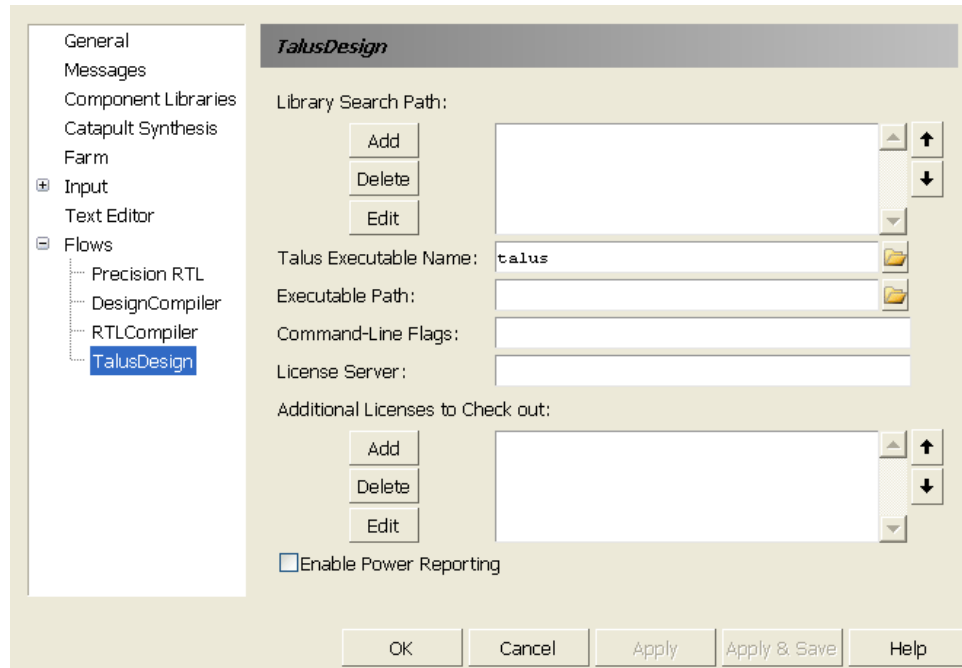
- **Enable Power Reporting**

Directs the RTL Compiler to generate a power analysis report. You can also set this option using the “`options set Flows/RTLCompiler EnablePowerReporting <true_or_false>`” command.

Set TalusDesign Flow Options

Set the default library characterization values for the Magma Talus Design flow. The Talus Design flow is for downstream RTL synthesis and library characterization. The Catapult Library Builder provides library templates for creating compatible libraries but does not currently provide prebuilt and characterized Talus Design libraries. You must create your own libraries with the Catapult Library Builder.

Figure 1-22. Talus Design Flow Options



- **Library Search Path**

Adds/deletes pathnames to directories containing Talus Design libraries. The specified libraries are searched in addition to the Talus Design default library search paths. Use the Add, Edit and Delete buttons to modify the paths in the list. Use the up and down arrows to move a selected path higher or lower in the search order.

You can also set this option using the “`options set Flows/TalusDesign SearchPath <list_of_paths>`” command.

- **Talus Executable Name**

Specifies the name of the Talus Design tool to invoke. The default is talus. Click the folder button to browse to the desired executable. You can also set this option using the “`options set Flows/TalusDesign shellExe <string>`” command.

- **Executable Path**

Specifies a path to search for the Talus Design executable. If none is found, the standard path is checked. Click the folder button to browse to the desired path. You can also set this option using the “`options set Flows/TalusDesign Path <path>`” command.

- **Command-Line Flags**

Specifies the command line switches for the Talus Design executable. You can also set this option using the “`options set Flows/TalusDesign Flags <list_of_flags>`” command.

- **License Server**

List of license servers for Talus Design licenses. Catapult assigns the specified list to the FLEXnet environment variable `MAGMA_LICENSE_FILE` prior to launching Talus Design. Refer the [FLEXnet Licensing End User Guide](#) for information about the syntax of FLEXnet environment variables. You can also set this option using “`options set Flows/TalusDesign LicenseServer <list_of_servers>`” command.

- **Additional Licenses to Check out**

List one or more license names that can be used. List items are separated by spaces. You can also set this option using the “`options set Flows/TalusDesign Licenses <list_of_composites>`” command.

- **Enable Power Reporting**

Directs Talus Design to generate a power analysis report. You can also set this option using the “`options set Flows/TalusDesign EnablePowerReporting <true_or_false>`” command.

Saving and Restoring Session Options

Saving your option settings can serve three purposes. First, saved settings can be loaded automatically each time a new Library Builder session is started. Second, the “**Save Options As...**” command enables you to save any number of alternate configurations. Third, when a new release of Catapult software is installed, option settings that were saved from the previous release are automatically applied to the new release.

By default the options are saved in the Catapult *registry*, unless you explicitly save them to a `catapult.ini` file. Refer to “[Catapult C Library Builder Registry](#)” on page 48 and “[The Catapult Initialization File](#)” on page 48 for more information.

Saving Options

You can use either the **Tools > Save Options** or **Tools > Save Options As...** pulldown menus. Alternatively, you can use the `options save` command. When using **Save Options As...**, name the output file `catapult.ini` if you want Library Builder to load it automatically at startup.

- When saving settings specific to a library project, use **Save Options As...** and save them to a `catapult.ini` file in the project directory.
- **Save Options** will save the settings to the source location from which they were loaded when the session started. The default location is the Catapult C Library Builder registry.

Additionally, you can have Catapult automatically save your settings each time you exit the tools. Refer to the section “[Set General Options](#)” on page 24 for information about this option.

Restoring Options

During startup, Catapult C Library Builder searches for saved options in the following locations, in the order listed. A message displays in the transcript window telling you the source location of the loaded options.

1. A `catapult.ini` file in the current working directory
2. A `catapult.ini` file in the user's HOME directory
3. The Catapult registry

Catapult C Library Builder Registry

On Windows systems, Catapult C Library Builder uses the Windows registry to store the settings. On UNIX systems, the registry is a directory created and maintained by Library Builder at `$HOME/.catapult`. In it, Library Builder keeps separate files for each different platform and software version. Table 1-2 lists the platform identifiers that are embedded in the filenames:

Table 1-2. Platform Identifiers in Registry Filenames

Platform Name	Identifier
Linux	ixl
Solaris	ss5

The Catapult Initialization File

The Library Builder initialization file, `catapult.ini`, is an ASCII text file containing the default settings for all Library Builder system options. [Example 1-1](#) shows the format of the Library Builder `catapult.ini` file. The example is an excerpt of the factory default settings.

Example 1-1. Example `catapult.ini` File

```
[General]
RestoreCWD = false
StartInWD =
SaveSettings = false
ShowToolBar = true
ShowFlowWindow = true
PdfViewer = acroread
AutoSaveLibraryBackup = 5
AutoRerunFailedComponent = 3
MaxTaskRunTime = 360
MaxComponentsPerTask = 10
RemoveProjectDirectories = true
NetlistFormat = VHDL

[Message]
```



```

ErrorOverride = ASSERT-1 CIN-17 CIN-48 CIN-46 CIN-49 CIN-50 CIN-54
HIER-4 SIF-6 CNS-9 HIER-20
WarningOverride =
InformationalOverride =
Hide = CRD-177

[ComponentLibs]
SearchPath = {$MGC_HOME/pkgs/siflibs}
{$MGC_HOME/pkgs/siflibs/designcompiler}
{$MGC_HOME/pkgs/siflibs/psr2006a.112}
{$MGC_HOME/pkgs/siflibs/psr2005c.151}
{$MGC_HOME/pkgs/siflibs_inhouse} {$MGC_HOME/pkgs/ccs_altera}
TemplateSearchPath = {$MGC_HOME/pkgs/siflibs/templates}
{$MGC_HOME/pkgs/siflibs_inhouse/templates}

[CatapultC]
UseFullSynthesisTool = false
Path = $CATAPULT_HOME/bin
Flags =

[Farm]
EnableFarm = false
HostDatabasePath = ./hostdb
MaxParallelTasks = 0
SuspendHostOnInvocationFailure = 0
SuspendHostOnFailure = 0
RshCommand = rsh %HOSTNAME% '%COMMAND%'

[TextEditor]
WindowLayout = Tabbed
CodeBrowser = false
LineNumbers = true
OutlineMode = false

[Flows]
FlowSearchPath =

[Flows/DesignCompiler]
SearchPath =
Path =
Flags =
ShellType = dctcl

[Flows/DesignCompiler/FOLDERNAME]
<type> = string
<description> = Output File Folder Name
<default> = Design Compiler
<value> = Design Compiler

[Flows/Precision]
Path = $PRECISION_HOME/bin
Flags =
addio = false
retiming = false
run_pnr = false

[Flows/Precision/FOLDERNAME]
<type> = string

```

<description> = Output File Folder Name
<default> = Precision
<value> = Precision

Chapter 2

Creating and Editing Libraries

Setting the Working Directory	51
Creating a New Library	51
Editing RAM Library Properties	67
Saving Libraries	73

The Library Builder provides ASIC and FPGA library templates to help you to create and characterize libraries for use in the Catapult C Synthesis tool. The general procedure for characterization is as follows:

1. Create a working directory where the new library will be built (a working area).
2. Invoke the Library Builder and set the working directory within the tool. See “[Setting the Working Directory](#)” on page 51.
3. Create a new library from one of the supplied templates. See “[Creating a New Library](#)” on page 51 and “[Editing RAM Library Properties](#)” on page 67.
4. Use the Library Explorer window or the Farm window to characterize the library. See “[Library Characterization](#)” on page 75 and “[Using the Library Farm](#)” on page 90.
5. Save the library. See “[Saving Libraries](#)” on page 73.

Setting the Working Directory

The Library Builder works with the files in a *working directory* which you create before you invoke the tool. This is the place where all generated *output* files are placed.

1. Click on the **Set Working Directory** task in the Task Bar window or use the **File > Set Working Directory...** menu item to open file system browser.
2. Browse to the directory that you want to set as the working directory and click **OK** to make it your working directory.

Alternatively, you can use the `set_working_dir` command.

Creating a New Library

Catapult C Library Builder provides many different library templates for the various types of libraries you can create. The “Base ASIC Library” template contains all of the basic components needed for synthesis. There are various templates for creating ROM, Custom RAM and

RegisterFile libraries. Finally, the “Blank Library” template is an empty library that you populate with custom component modules and operators.

The procedure for creating a library by using the GUI is as follows:

1. Click the **New Library** task in the Task Bar window, or select the **File > New > New Library** menu item to open the Library Creation dialog box.
2. In the Library Creation dialog box, select the synthesis tool to be used for characterization. For ASIC libraries, choose either the **Cadence RTL Compiler**, **Magma Talus Design**, or **Synopsys Design Compiler**. For FPGA libraries, select **Precision RTL Synthesis**. When a tool is selected, a list of library templates for that tool display below it.
3. Select a library template under the selected synthesis tool.

Fill in the fields on the right side of the Library Creation dialog box. Items in **red** are required. Refer to:

- “[Cadence RTL Compiler Options](#)” on page 53
 - “[Magma Talus Design Options](#)” on page 57
 - “[Precision RTL Synthesis Options](#)” on page 59
 - “[Synopsys Design Compiler Options](#)” on page 61
4. Click OK. The new library is created in the Library Builder database but not saved to disk until you explicitly save it.
 5. Edit the library. In the Library Explorer window, double-click on the newly created library to open the Library Editor. (or right-click and select **Edit** from on the pop-up menu).
 6. Save the new library and close the Library Editor.
 7. Characterize the new library and save the changes. (See “[Characterizing Libraries or Components](#)” earlier in this chapter.)
 8. Add the new library to the Catapult C Synthesis library search path. Use **Tools > Set Options > Component Libraries** in the Catapult C Synthesis application.

The command line interface uses the “**flow run**” command to launch the “library add” flow for the specified synthesis tool. Library Builder provides different flow packages for each of the supported synthesis tools. The flow package names are `DesignCompiler`, `RTLCompiler`, and `Precision`. Run the flow for the target synthesis tool and specify a library template and its required settings. The flow will add the library to the Library Builder database.

The command format is:

```
flow run /<package>/library add <lib_template> <options>
```

For example, the following command creates a “Base ASIC Library” (“base”) for the Design Compiler tool.

```
flow run /DesignCompiler/library add base \  

  -libname my_lib \  

  -libtitle my_lib \  

  -vendor Sample \  

  -technology 180nm \  

  -link_library sample_180nm.db \  

  -target_library sample_180nm.db
```

The set of valid options varies depending on the type of library template. To get a list of the valid template names for a flow package, use the “-help” switch as follows:

```
flow run /<package>/library add -help
```

Similarly, to see the set of valid options for a particular library template, use the following command:

```
flow run /<package>/library add <lib_template> -help
```

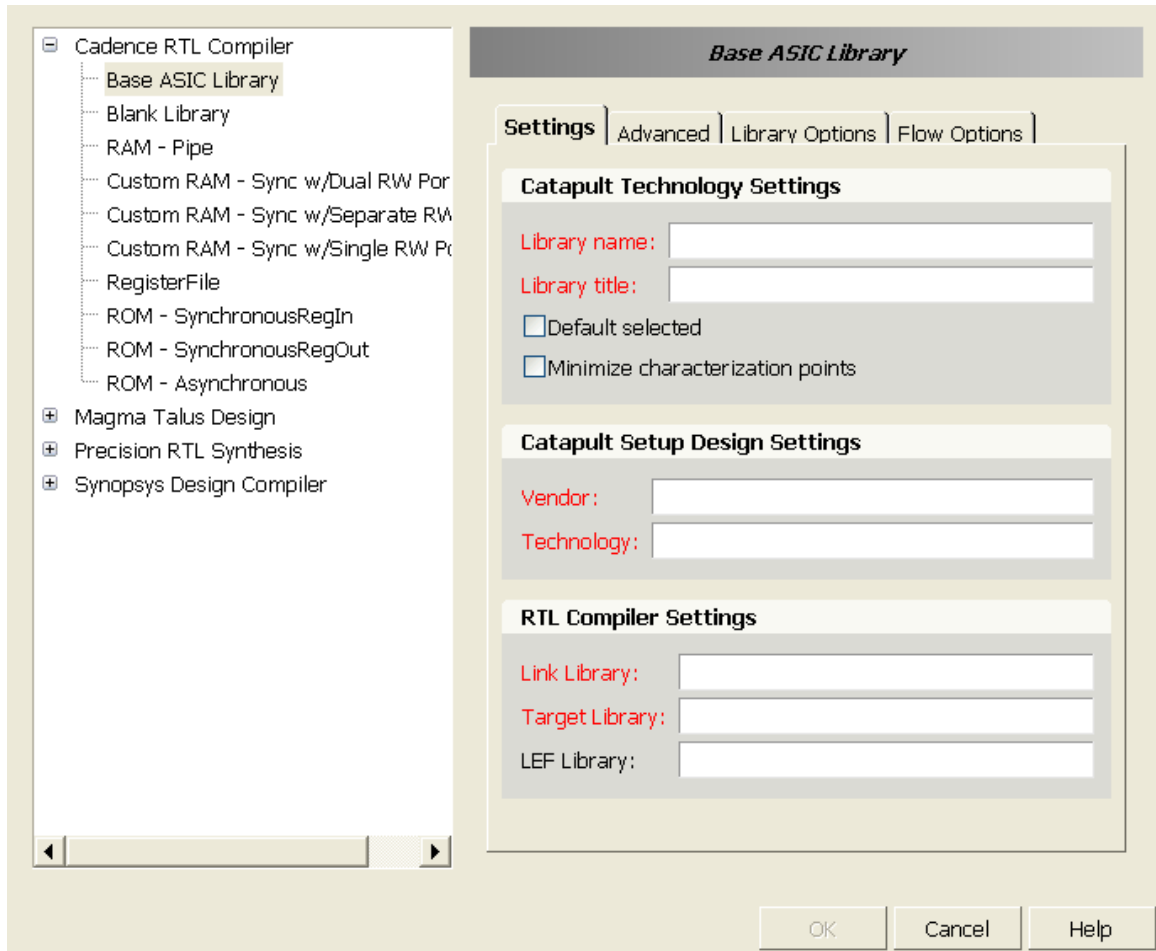
The set of options correspond to the data fields in the Library Creation window. For descriptions of the available options, refer to:

- [“Cadence RTL Compiler Options”](#) on page 53.
- [“Magma Talus Design Options”](#) on page 57,
- [“Precision RTL Synthesis Options”](#) on page 59
- [“Synopsys Design Compiler Options”](#) on page 61,

Cadence RTL Compiler Options

[Figure 2-1](#) shows the **Settings** tab for the Base ASIC Library template. Other templates have the same settings. All data fields on the **Settings** tab are required. All other tabs provide optional settings. Most of the optional fields are the same for all templates, but some are only available for specific types of templates. This section describes all possible fields found on each tab, regardless of which template the field may appear on.

Figure 2-1. Library Creation Dialog Box for Cadence RTL Compiler



The Settings Tab

Enter basic configuration information on the **Settings** tab:

- **Catapult Technology Settings**
 - **Library Name:** Enter a symbolic name for the new library that will appear in the Resource Type list of the Architectural Constraints dialog box in the Catapult C Synthesis tool. For more information, refer to “[Specifying Architectural Constraints](#)” in the *Catapult C Synthesis User’s and Reference Manual*.
 - **Library Title:** This title appears in the Compatible Libraries list of the [Setup Design](#) dialog box in the Catapult C Synthesis tool when the user selects the vendor and technology that are associated with this new library.
 - **Default Selected:** Specifies whether or not the new library will be selected by default when it appears in the [Setup Design](#) dialog box in the Catapult C Synthesis tool.

- **Catapult Setup Design Settings**

- **Vendor and Technology:** The new library is associated with the specified vendor and technology names. When these names are selected in the [Setup Design](#) dialog box in the Catapult C Synthesis tool, your new library will appear in the Compatible Libraries list of the dialog box.

The names must match those in Catapult. For example, vendor name “Sample” and technology name “065nm.”

- **RTL Compiler Settings:**

These values must match the corresponding values in the target RTL Compiler technology file/files (.lib) or a synthesis error occurs when the library is generated. These values are passed directly to RTL Compiler as command line arguments via the command script generated by Catapult.

If the target technology is available in multiple RTL Compiler libraries, such as a *fast* and a *slow* library, you can characterize this Library Builder library using one or more of the libraries; List each RTL Compiler library in the **Link Library** and **Target Library** fields. Library Builder characterizes components using a combination of the technology libraries to give you the best trade off between area and performance.

- **Link Library:** A list of link library names (space separated and enclosed in braces).
- **Target Library:** A list of target library names (space separated and enclosed in braces).

The value is used in the following command line in the RC script:

```
set_attr library sample_065nm.lib
```

- **LEF Library:** A list of LEF (Layout Exchange Format) library names. List items are space separated and enclosed in braces.

The Advanced Tab

Optionally modify the settings on the **Advanced** tab:

- **Library Settings**

- **Time Unit:** Specifies the timescale unit that will be found in RTL Compiler output reports and input constraint files. Catapult always computes and expresses time in nanoseconds internally. When Catapult reads time data from the RTL Compiler report file, it converts the time values to nanoseconds based on the timescale specified in the “Time unit” field. Similarly, when Catapult generates a constraint file, it converts time data from nanoseconds to the target units.

Choose one of the following timescale units: nanoseconds (“ns”), picoseconds (“ps”) or femtoseconds (“fs”). The default setting is nanoseconds.

The following Library Settings apply only to memory templates.

- **Component Area:** The value in this field is assigned to the “area” property on the “All” binding of module(s).
- **Component Delay:** For RAM /ROM templates, the value in this field is assigned to the “delay” property on “read_ram”/“read_rom” binding of the module. For the RAM_pipe template, the “delay” property is on the “All” binding.
- **Default value for Address and Data ports:** The field only applies to RAM templates. Valid values are 0 = active low, 1 = active high, X.
- **Default chip select active:** The field only applies to RAM templates. Valid values are 0 = active low, 1 = active high, U = unused.
- **RTL Compiler Settings**

This group of settings apply only to the “Base ASIC Library” template. Each setting corresponds to a setting in the RTL Compiler tool. Refer to the RTL Compiler documentation for information about each setting.
- **Characterization Settings**

The following Characterization Clock Period settings apply only to the “Base ASIC Library”, “RAM - Pipe” and ROM templates. They allow you to override the default settings for multi-point characterization of library components. Refer to [“Multi-Point Characterization”](#) on page 76 for more information.

 - **Clock Period Fastest:** This value specifies the clock speed setting (typically nanoseconds) for DC when characterizing the fastest clock period (largest area).
 - **Clock Period Smallest:** This value specifies the clock speed setting (typically nanoseconds) for DC when characterizing the slowest clock period (smallest area).
 - **Clock Period Percentages:** This field specifies a set of points within the range of clock speeds bounded by the fastest and smallest settings above. The points are specified as percentages of the range. The default set consists of four points: 100%, 75%, 50% and 0%, where 100 = fastest and 0 = smallest.

The Library Options Tab

- **WireLoad**
 - **Wire Load Model:** This field allows you to specify arguments for the RC attribute `set_attr wireload_model`. If this field is not empty, it is passed directly to RC by the RC script generated by Catapult.
 - **Wire Load Mode:** This variable allows you to specify arguments for the RC attribute `set_attr wireload_mode`. If this field is not empty, it is passed directly to RC by the RC script generated by Catapult.
 - **Wire Load Selection Group:** This field allows you to specify arguments for the RC attribute `set_attr wireload_selection`. If this field is not empty, it is passed directly to RC by the RC script generated by Catapult.

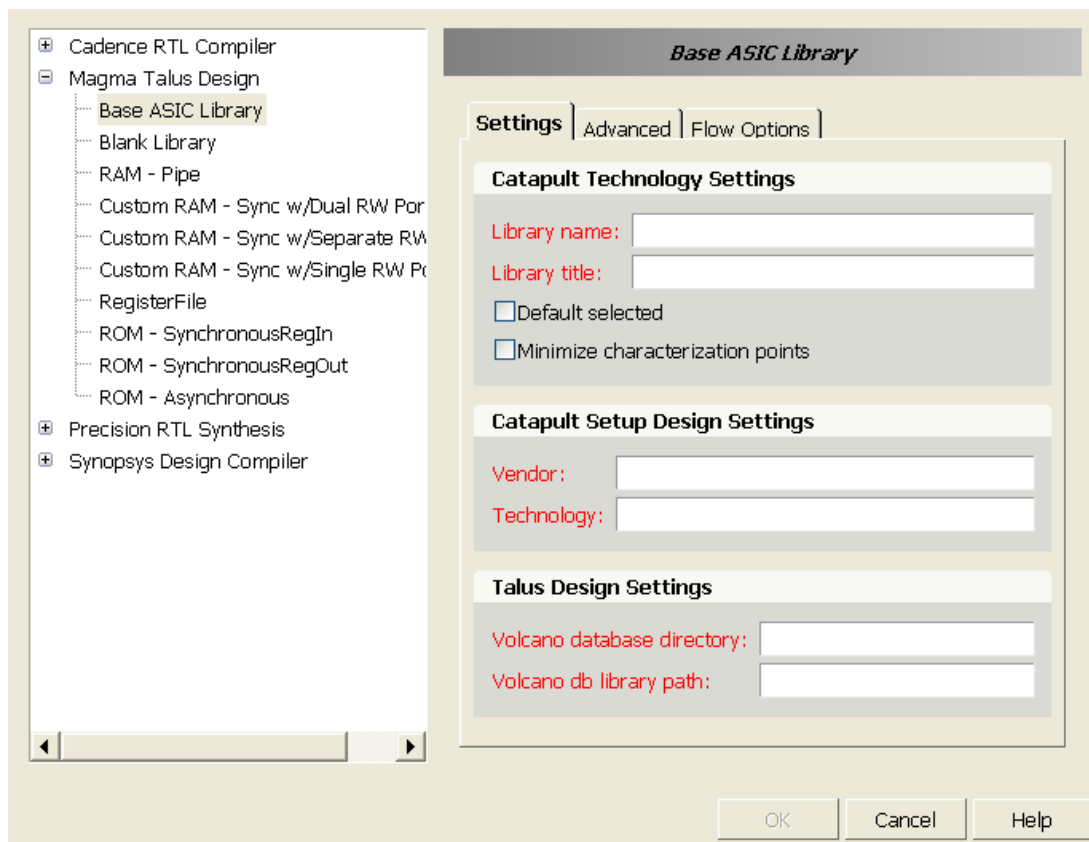
The Flow Options Tab

Optionally modify the default settings for the Cadence RTL Compiler flow options. These flow options are used in the Catapult C Synthesis tool during library characterization. These settings are not saved as part of the library, but are global to the Library Builder session. Refer to “[Set RTL Compiler Flow Options](#)” on page 43 for a description of these options.

Magma Talus Design Options

Figure 2-2 shows the Settings tab for the Base ASIC Library template. Other templates have the same settings except that they do not have the “TalusDesign Settings” group. All data fields on the Settings tab are required. The Advanced tab and Flow Options tab contain optional settings. Most of the optional fields are the same for all templates, but some are only available for specific types of templates. This section describes all possible fields found on each tab, regardless of which template the field may appear on.

Figure 2-2. Library Creation Dialog Box for Talus Design



The Settings Tab

Enter basic configuration information on the **Settings** tab:

- **Catapult Technology Settings**

- **Library name:** Enter a symbolic name for the new library that will be appear in the Resource Type list of the Architectural Constraints dialog box in the Catapult C Synthesis tool.
- **Library title:** This title appears in the Compatible Libraries list of the [Setup Design](#) dialog box in the Catapult C Synthesis tool when the user selects the technology associated with this library.
- **Default Selected:** Specifies whether or not this library is selected by default when it appears in the [Setup Design](#) dialog box in the Catapult C Synthesis tool.

- **Catapult Setup Design Settings**

- **Vendor and Technology:** The new library will be associated with the specified vendor and technology names. When these names are selected in the [Setup Design](#) dialog box in the Catapult C Synthesis tool, the new library will appear in the Compatible Libraries list of the [Setup Design](#) dialog. Example names might be, vendor name “TSMC” and technology name “tsmc18lv.”

- **Talus Design Settings**

The following fields specify settings that will be passed to the Talus Design tool. The values you enter in this section must match the corresponding values of the target Talus Design technology library file or files (`.volcano`). If the fields don't match, you'll get a synthesis error in Talus Design when you try to generate your library.

- **Volcano database directory:** The full path to the Volcano technology database. For example:

```
<your_path>/tsmc18lv_typ_beh_magma.volcano
```

- **Volcano db library path:** The path within the Volcano library specifying the target technology name. For example:

```
/tsmc18lv
```

The Advanced Tab

Optionally modify the characterization settings on the **Advanced** tab. Only the Base ASIC Library and the ROM templates have advanced options.

- **Library Settings**

The following Library Settings apply only to memory templates.

- **Component Area:** The value in this field is assigned to the “area” property on the “All” binding of module(s).

- **Component Delay:** For RAM /ROM templates, the value in this field is assigned to the “delay” property on “read_ram”/“read_rom” binding of the module. For the RAM_pipe template, the “delay” property is on the “All” binding.
- **Default value for Address and Data ports:** The field only applies to RAM templates. Valid values are 0 = active low, 1 = active high, X.
- **Default chip select active:** The field only applies to RAM templates. Valid values are 0 = active low, 1 = active high, U = unused.
- **Talus Design Settings**
 This group of settings apply only to the “Base ASIC Library” template. Each setting corresponds to a setting in the Talus Design tool. Refer to the Talus Design tool documentation for information about each setting.
- **Characterization Settings**
 The following Characterization Clock Period settings apply only to the “Base ASIC Library”, “RAM - Pipe” and ROM templates. They allow you to override the default settings for multi-point characterization of library components. Refer to [“Multi-Point Characterization”](#) on page 76 for more information.
 - **Clock Period Fastest:** This value specifies the clock speed setting for Talus Design when characterizing the fastest clock period (largest area).
 - **Clock Period Smallest:** This value specifies the clock speed setting for Talus Design when characterizing the slowest clock period (smallest area).
 - **Clock Period Percentages:** This field specifies a set of points within the range of clock speeds bounded by the fastest and smallest settings above. The points are specified as percentages of the range. The default set consists of four points: 100%, 75%, 50% and 0%, where 100 = fastest and 0 = smallest.

The Flow Options Tab

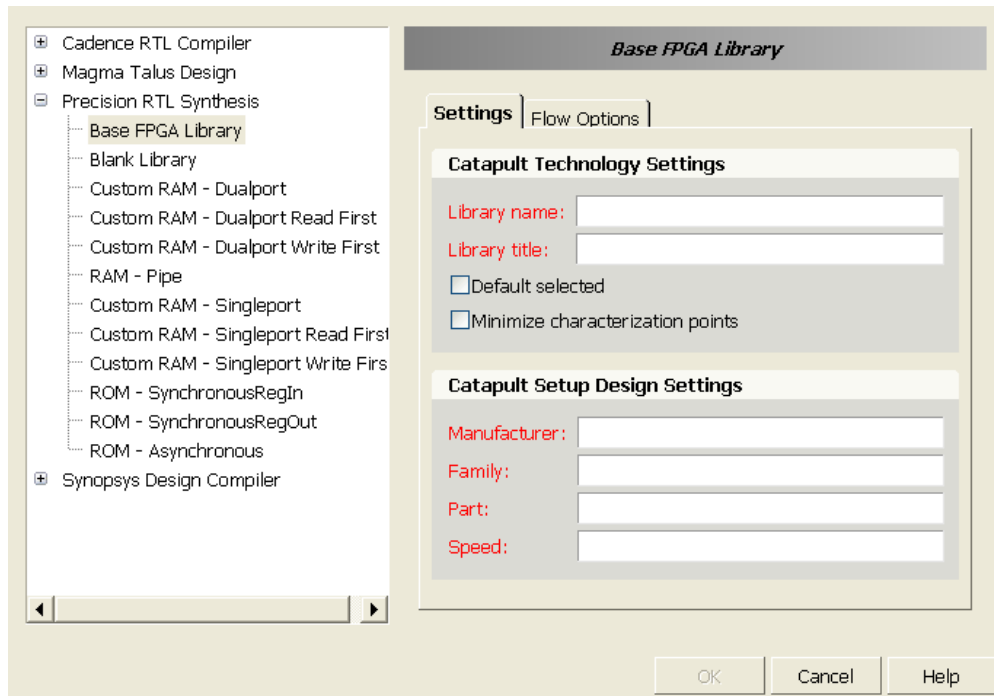
Optionally modify the default settings for the TalusDesign flow options. These flow options are used in the Catapult C Synthesis tool during library characterization. These settings are not saved as part of the library, but are global to the Library Builder session. Refer to [“Set TalusDesign Flow Options”](#) on page 45 for a description of these options.

Precision RTL Synthesis Options

[Figure 2-3](#) shows the Settings tab for the Base ASIC Library template. All data fields on the Settings tab are required.

The Advanced tab and Flow Options tab contain optional settings. Most of the optional fields are the same for all templates, but some are only available for specific types of templates. This section describes all possible fields found on each tab, regardless of which template the field may appear on.

Figure 2-3. Library Creation Dialog Box for Precision RTL



The Settings Tab

Enter basic configuration information on the **Configuration** tab:

- **Catapult Technology Settings**
 - **Library Name:** Enter a symbolic name for the new library that will be appear in the Resource Type list of the Architectural Constraints dialog box in the Catapult C Synthesis tool. For more information, refer to “[Specifying Architectural Constraints](#)” in the *Catapult C Synthesis User’s and Reference Manual*.
 - **Library Title:** This title appears in the Compatible Libraries list of the [Setup Design](#) dialog box in the Catapult C Synthesis tool if the user selects the vendor and technology that are associated with this new library.
 - **Default selected:** Specifies whether or not this library is selected by default when it appears in the [Setup Design](#) dialog box in the Catapult C Synthesis tool.
- **Catapult Setup Design Settings**

The new library is associated with the FPGA technology specified by the fields: **Manufacturer**, **Family**, **Part** and **Speed**. The library parts are characterized for the specified technology. Note that for Custom RAM templates, the **Part** field is optional and appears on the Advanced tab.

The set of valid values for these fields consists of technologies supported by the Precision RTL Synthesis tool. Refer to the section [“Setting Up the Design”](#) in *Catapult C Synthesis User’s and Reference Manual*. The values must match the naming used in Precision RTL Synthesis, although these fields are not case-sensitive.

For example, Manufacturer names include Altera and Xilinx. Example Family names include Cyclone, Stratix and VIRTEX.

The Advanced Tab

The **Advanced** tab values are optional and apply to all RAM/ROM memory templates.

- **Catapult Setup Design Settings**

The **Part** setting specifies the list of library parts that are compatible with this RAM/ROM. The value can be specified as a glob expression or a literal list. The default value is ‘*’, which makes the RAM/ROM compatible with all parts.

- **Library Settings**

The following options apply only to memory templates.

- **Component Area:** The value in this field is assigned to the “area” property on the “All” binding of module(s).
- **Component Delay:** For RAM /ROM templates, the value in this field is assigned to the “delay” property on “read_ram”/“read_rom” binding of the module. For the RAM_pipe template, the “delay” property is on the “All” binding.
- **Default value for Address and Data ports:** The field only applies to RAM templates. Valid values are 0 = active low, 1 = active high, X.

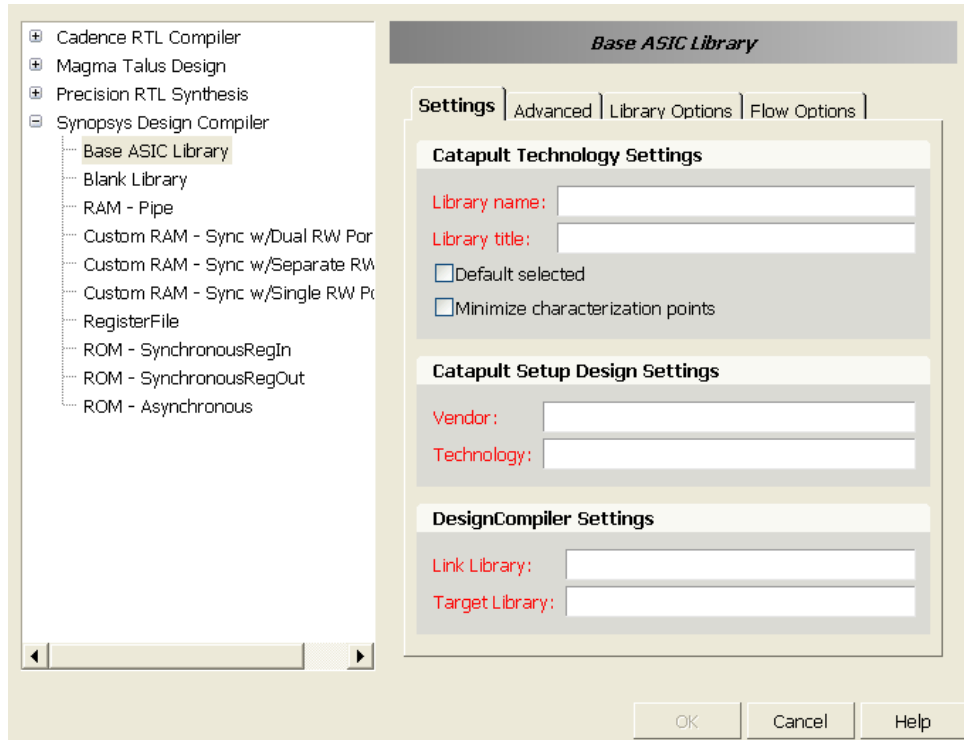
The Flow Options Tab

Optionally modify the default settings for the Precision flow options. These flow options are used in the Catapult C Synthesis tool during library characterization. These settings are not saved as part of the library, but are global to the Library Builder session. Refer to [“Set Precision Flow Options”](#) on page 39 for a description of these options.

Synopsys Design Compiler Options

[Figure 2-4](#) shows the **Settings** tab for the Base ASIC Library template. All other tabs provide optional settings. Some of the optional fields are the same for all templates but some are only available for specific types of templates. This section describes all possible fields found on each tab, regardless of which template the field may appear on.

Figure 2-4. Library Creation Dialog Box for Design Compiler



The Settings Tab

Enter basic configuration information on the **Settings** tab:

- **Catapult Technology Settings**
 - **Library Name:** Symbolic name for the new library that displays in the Resource Type list of the Architectural Constraints dialog box in the Catapult C Synthesis tool. For more information, refer to “[Specifying Architectural Constraints](#)” in the *Catapult C Synthesis User’s and Reference Manual*.
 - **Library Title:** Title that Catapult displays in the Compatible Libraries list of the [Setup Design](#) dialog box when the vendor and technology associated with the new library are selected. The specified string is assigned to the ui_libtitle variable in the library.
 - **Default Selected:** Specifies whether or not the new library is selected by default when it displays in the [Setup Design](#) dialog box in the Catapult.
 - **Minimize characterization points:** Speeds up the characterization process by minimizing the number of data points (QMODS) used during characterization. This option may affect the accuracy of the components in the new library. For more information about QMODS and the characterization process, see “[Library Characterization](#)” on page 75.
- **Catapult Setup Design Settings**

- **Vendor and Technology:** The new library is associated with the specified vendor and technology names. When these names are selected in the [Setup Design](#) dialog box in the Catapult, the new library displays in the Compatible Libraries list.

The names must match those in Catapult. For example, vendor name “LSI Logic” and technology name “lcbg11p.”

- **Design Compiler Settings:**

This group of settings apply only to the “Base ASIC Library” template. Each setting corresponds to a setting in the Design Compiler tool. Refer to the Design Compiler documentation for information about each setting.

These values must match the corresponding values in the target Design Compiler technology file/files (.db) or a synthesis error occurs when the library is generated. These values are passed directly to Design Compiler as command line arguments via the command script generated by Catapult.

If the target technology is available in multiple Design Compiler libraries, such as a *fast* and a *slow* library, you can characterize this Library Builder library using one or more of the libraries; List each design Compiler library in the **Link Library** and **Target Library** fields. Library Builder characterizes components using a combination of the technology libraries to provide the best trade off between area and performance.

- **Link Library:** A list of link libraries (space separated and enclosed in braces). This value is saved in the library variable named link_library and is used later in the following command line in the DC script:

```
set_link_library = {list lib_1.db lib_2.db}
```

- **Target Library:** A list of target libraries names (space separated and enclosed in braces). This value is typically the same as the Link Library value. This value is used in the following command line in the DC script:

```
set_target_library = {list lib_1.db lib_2.db}
```

Note that for the Blank Library template, these are optional settings found on the Advanced tab.

The Advanced Tab

The settings on the **Advanced** tab are optional:

- **Library Settings**

These options apply only to the *Base ASIC Library* template and should match the settings in the DC technology library.

- **Time Unit:** Specifies the timescale unit used in DC output reports and input constraint files. Options include: nanoseconds (ns), picoseconds (ps) or femtoseconds (fs). The default setting is ns.

- **Capacitance Unit:** Specifies the capacitance scale used by DC output reports and input constraints files. Options include: nF (nanofarads), pF (picofarads), and fF (femtofarads).

The following options apply only to memory templates:

- **Component Area:** The value in this field is assigned to the “area” property on the “All” binding of module(s).
- **Component Delay:** For RAM /ROM templates, the value in this field is assigned to the “delay” property on “read_ram”/“read_rom” binding of the module. For the RAM_pipe template, the “delay” property is on the “All” binding.
- **Default value for Address and Data ports:** The field only applies to RAM templates. Valid values are 0 = active low, 1 = active high, X.

Default chip select active: The field only applies to RAM templates. Valid values are 0 = active low, 1 = active high, U = unused.

- **Design Compiler Settings**

This group of settings apply only to the Base ASIC Library template. Each setting corresponds to a setting in DC. Refer to the DC documentation for information about corresponding options.

- **External Capacitive Load (in library units):** Specifies the output load. Corresponds to the set_load command in DC. Best practice is to use the capacitance value from an input pin of a simple cell such as a buffer, inverter, or 2-input AND gate. For example:

```
[expr {1* [load_of CORE65LPHVT/HS65_LH_IVX9/A] }]
```

Value can be an expression or a floating point. If possible, cut and paste the capacitance value from the input pin. For example: 0.001714

- **Operating Conditions:** Specifies operating conditions for the new library. This value supersedes the default value defined in the technology library.

The value is the name of operating conditions. For example, wc_1.10V_125C

- **External Driver Cell Library:** Specifies the library where the external driving cell is located.
- **External Driver Cell:** Specifies the external driver cell name.
- **External Driver Cell Pin:** Specifies the name of the driving pin on the external driver cell.
- **Input Transition (Slew):** Specifies input delay as a number in units (ns or ps) as an alternative to defining an external driving cell.

- **Report Timing Significant Digits:** Specifies the number of decimal places used in the timing measurements. Be sure to specify enough decimal places to effectively capture the data in the library.
- **Enable Scan Registers:** Enables registers with built-in scan chain I/O for synthesis.
- **Set Compile Mode to Ultra:** Enables ultra mode for synthesis (post-characterization). This option improves QofR but increases the synthesis time.
- **Advanced Retiming:** Specifies the *optimize_registers* command instead of the *balance_registers* command for DC. The *optimize_registers* command requires a specific DC license. For more information, see the DC documentation.
- **No Boundary optimization:** Disables boundary optimization in DC.
- **Characterization Settings**
 The following Characterization Clock Period settings apply only to the “Base ASIC Library”, “RAM - Pipe” and ROM templates. They allow you to override the default settings for multi-point characterization of library components. Refer to “[Multi-Point Characterization](#)” on page 76 for more information.
 - **Clock Period Fastest:** Specifies the clock speed setting (typically nano-seconds) for DC when characterizing the fastest clock period (largest area).
 - **Clock Period Smallest:** Specifies the clock speed setting (typically nano-seconds) for DC when characterizing the slowest clock period (smallest area).
 - **Clock Period Percentages:** Specifies a set of points within the range of clock speeds bounded by the fastest and smallest settings above. The points are specified as percentages of the range. The default set consists of four points: 100%, 75%, 50% and 0%, where 100 = fastest and 0 = smallest.

The Library Options Tab

The settings on the **Library Options** tab are optional and only available on the *Base ASIC Library* template.

- **DesignWare**

DesignWare refers to Synopsys DesignWare Building Block IP, which is a technology-independent, microarchitecture level library that is tightly integrated into the Synopsys synthesis environment. The DesignWare libraries *standard.sldb*, *dw_foundation.sldb*, *dw01.sldb*, *dw02.sldb*, *dw03.sldb* and so on, come with DC, and DC is more efficient in inferring typical microarchitectures such as adders, multipliers and such, if the licensed DesignWare architectures are enabled. In general, you can expect better QofR when using DesignWare libraries, as well as shorter DC runtimes.

DesignWare microarchitectures from the latter libraries require a DesignWare license. In order for the DC tool to use licensed DesignWare libraries, a list of the DesignWare libraries to be used must be provided to DC. Use the “**Synthetic Library(ies)**” option to

specify that list. Three other Catapult variables, “get_license”, “dont_get_license” and “wait_for_design_license”, can be used to control the DC tool’s access to DesignWare licenses. A DesignWare license is required for inferring licensed DesignWare blocks. If the DesignWare license feature is not present, the blocks that require that feature are not inferred. If the feature is present but currently unavailable, the DC tool errors out. That behavior can be modified to make the process wait for certain temporarily unavailable license features or not to use certain license features.

- **License(s) to check out:** Specifies a list of license features to be obtained. The list items are space separated. If the value is set, it is passed directly to the DC command “get_license” in the generated DC script so the specified license features are checked out at the beginning of synthesis. (They are held until the remove_license command is used or until the program is exited or until the DC shell is closed.) Refer to the license key file at your site to determine which licensed features are available. The command syntax for editing this variable in an existing library is:

```
library set /LIBS/<lib_name>/VARS/get_license --  
-VALUE <list_of_licenses>
```

- **Synthetic Library(ies):** Specifies a list of DesignWare synthetic libraries that the DC tool should use during characterization and synthesis. The list items are space separated. The command syntax for editing this variable in an existing library is:

```
library set /LIBS/<lib_name>/VARS/synthetic_library --  
-VALUE {<list_of_libs>}
```

- **Design Licenses not to use:** Specifies a list of DesignWare licenses that the DC tool is not allowed to use. The list items are space separated. The command syntax for editing this variable in an existing library is:

```
library set /LIBS/<lib_name>/VARS/dont_get_license --  
-VALUE <list_of_licenses>
```

The value of this variable is assigned to the DC variable “synlib_dont_get_license” in the generated DC script.

- **Design Licenses to wait for:** Specifies a list of design licenses that the DC tool should wait for if they are temporarily unavailable. The list items are space separated. The command syntax for editing this variable in an existing library is:

```
library set /LIBS/<lib_name>/VARS/wait_for_design_license --  
-VALUE <list_of_licenses>
```

The value of this variable is assigned to the DC variable “synlib_wait_for_design_license” in the generated DC script.

- **WireLoad**

- **Wire Load Model:** This field allows you to specify arguments for the DC command set_wire_load_model. If this field is not empty, it is passed directly to DC by the DC script generated by Catapult. When specifying the arguments, you must include

the option name and its value. For example, to include the “-name” option with a value of “tsmc18_wl20,” you would enter the following string in the Wire Load Model field:

```
-name tsmc18_wl20
```

The example above will add the command line to the DC script:

```
set_wire_load_model -name tsmc18_wl20
```

- **Wire Load Mode:** This variable allows you to specify arguments for the DC command `set_wire_load_mode`. If the value is anything other than “none”, it is passed directly to DC by the DC script generated by Catapult. The command syntax for editing this variable in an existing library is:

```
library set /LIBS/<lib_name>/VARS/wire_load_mode --  
-VALUE <none | top | enclosed | segmented>
```

- **Wire Load Selection Group:** This field allows you to specify arguments for the DC command `set_wire_load_selection_group`. If this field is not empty, it is passed directly to DC by the DC script generated by Catapult. The command syntax for editing this variable in an existing library is:

```
library set /LIBS/<lib_name>/VARS/wire_load_selection_group  
-- -VALUE <string>
```

For example, setting this variable to “-lib myWireLib.db -max myGroup” will add the following command line to the DC script:

```
set_wire_load_selection_group -lib myWireLib.db -max myGroup
```

The Flow Options Tab

Optionally modify the default settings for the Design Compiler flow options. These flow options are used in the Catapult C Synthesis tool during library characterization. These settings are not saved as part of the library, but are global to the Library Builder session. Refer to [“Set Design Compiler Flow Options”](#) on page 41 for a description of these options.

Editing RAM Library Properties

This section describes how to edit a RAM library. It covers the following tasks:

- [“Editing the RAM Library Variables”](#) on page 68
- [“Editing RAM Components Parameters”](#) on page 69
- [“Editing RAM Formula Information”](#) on page 70
- [“Editing RAM Timing”](#) on page 71
- [“Editing RAM Ports”](#) on page 72

Library Builder comes with the following templates for creating RAM library components:

- **Sync w/Dual RW Ports:** Contains RAM templates with dual read/write ports, one address line that controls both the read and write port.
- **Sync w/Separate RW Ports:** Contains RAM templates with separate read/write ports, one address line for the read port and one for the write port.
- **Sync w/Single RW Ports:** Contains RAM templates with a single read/write port.

You can modify pre-existing RAM templates as follows:

1. Select **New Library** from the Task Bar or the file menu. This will open the Library Creation dialog box from which you select one of the RAM templates.
2. Select a RAM template and set the fields in red on the “Settings” tab to match your Library. Refer to [“Creating a New Library”](#) on page 51 detailed information about filling in the Library Creation dialog box.
3. Optionally select the “Advanced” tab and set other types of data fields, such as area and delay values, and default port values.
4. Then click **OK** on the Library Creation dialog box to open the template in Library Editor.
5. If you did not set the area and delay values in step 3 above, you must edit the RAM template in the Library Editor and define the following three *Bindings* properties: *area* and both *read_ram delay* properties.
 - a. Expand the folders **Mods > RAM * > Bindings**.
 - b. Select the **All** binding and edit the *area* property.
 - c. Select each **read_ram** binding and edit the *delay* properties.
 - d. Save the RAM template. After the template is edited, it can be saved to a new library name. For more information, see [“Saving Libraries”](#) on page 73.
6. Add the new library to the Catapult library search path. If the new library is in the library search path for Library Builder, it will be available the next time you choose the **File > Open Library...** menu.

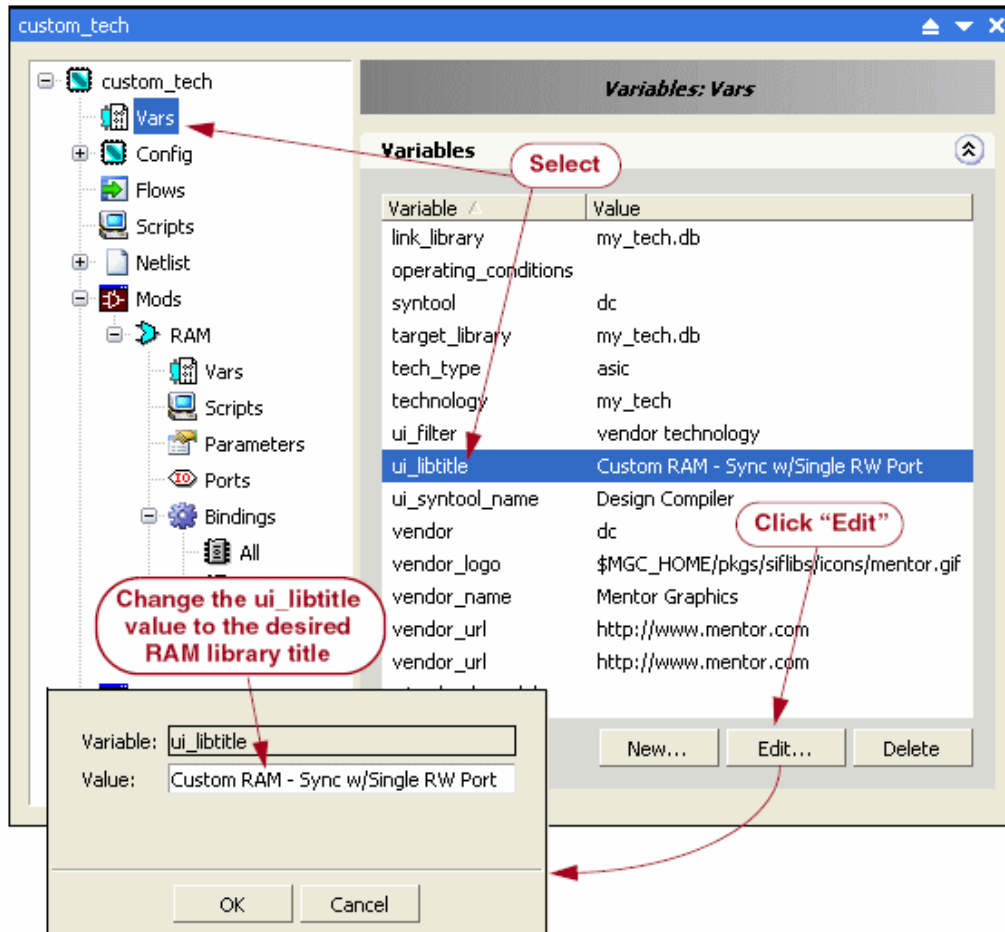
Editing the RAM Library Variables

You can add a new variable, edit an existing variable, or delete a variable. For example, you can edit the RAM library `ui_libtitle` variable, whose title string displays in the Setup Design Technology window of the Catapult C Synthesis tool.

1. Click on the “VARS” folder in the hierarchy view on the left. This displays all of the variable names and values on the right.

2. Select the `ui_libtitle` variable and click the **Edit** button to display the **Edit Variable** dialog box. It displays the variable name and its current value. Enter the desired value and click **OK** to accept the change.

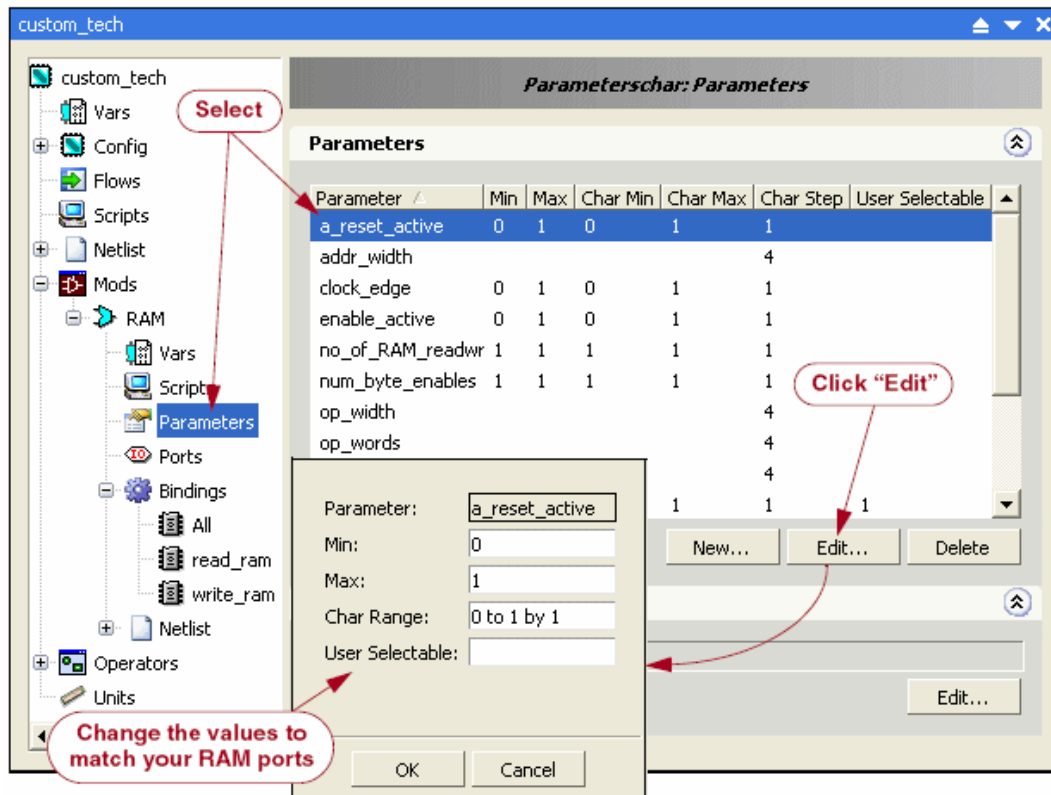
Figure 2-5. Editing the RAM Library Title Variable



Editing RAM Components Parameters

Open the MODS folder and then select "Parameters". This is where Library Builder controls the number of ports on the RAM. You can add new RAM information, edit existing information, or delete information.

Figure 2-6. Editing RAM Parameter Information

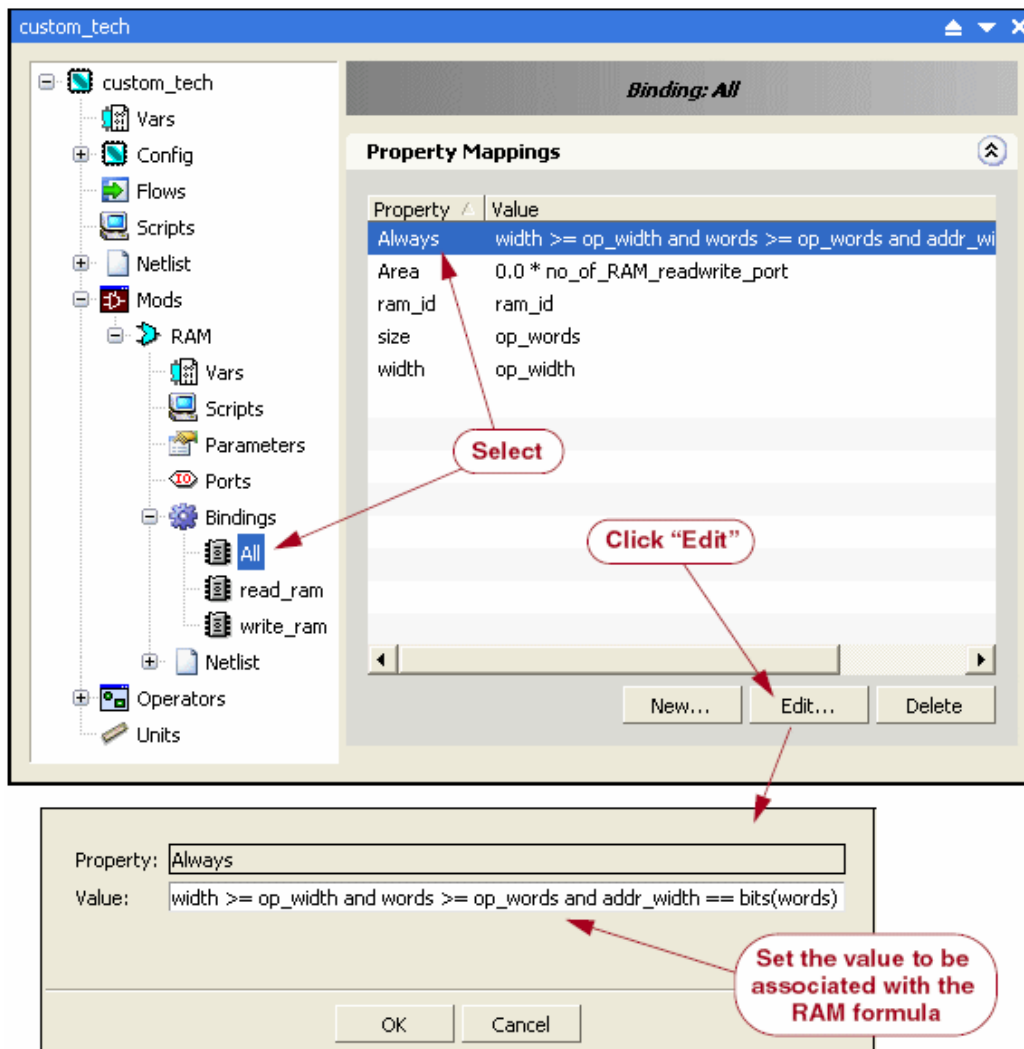


Editing RAM Formula Information

The BINDINGS folder is where you edit the formula for the area of the RAM. To set RAM formulas, open the BINDINGS folder and click on the "All" binding. The formula can use any of the PARAMETERS as variables, but usually only the number of words and the bitwidth control the area of a RAM.

The "user selectable" property can be set to true (1=true, 0=false) on individual parameters to make them editable by designers in the Catapult Constraint Editor. In a Catapult session, when editing the architectural constraints of a resource, any "user selectable" parameters of the library component is editable in the "Resource Options" field of the Constraint Editor. Each parameter field has a drop-down list of valid values to choose from.

Figure 2-7. Editing RAM Formula Information

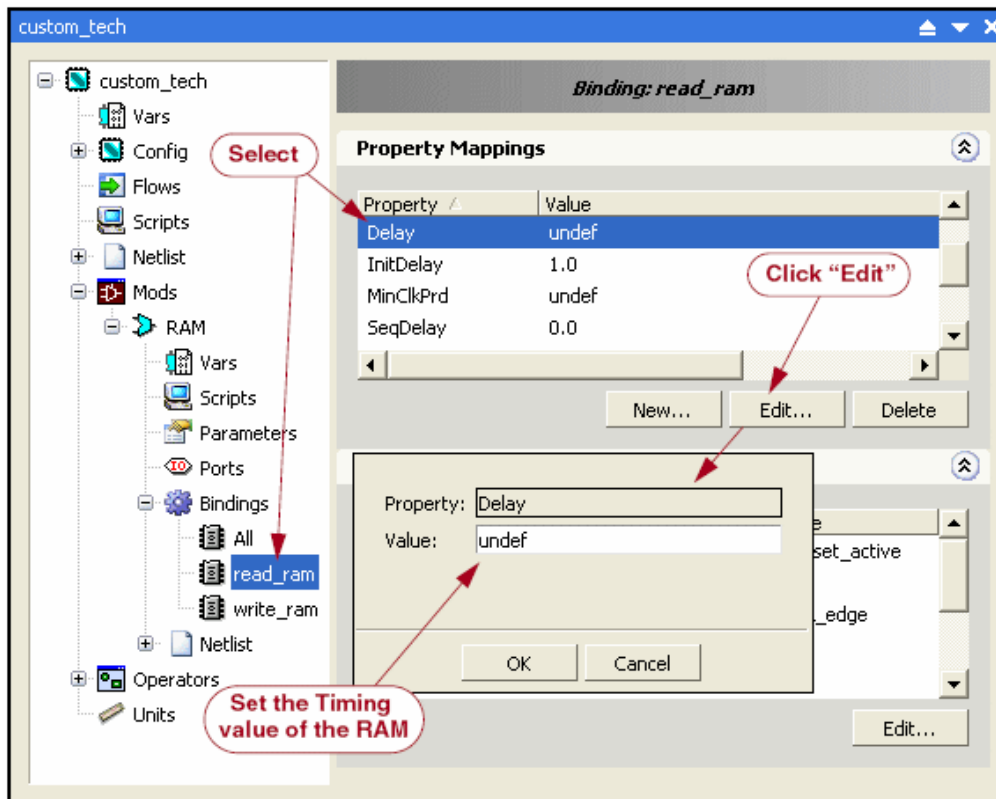


Editing RAM Timing

To edit the timing for the read/write operations of RAM, go to the BINDINGS folder and click on the **read_ram** or **write_ram** binding and modify the following timing properties:

- SeqDelay Number of Cycles that the operation takes. If the ports associated with that operation have input registers, this property is needed. If the ports are asynchronous, delete this property.
- InitDelay Number of cycles between accesses to this port. Default is SeqDelay + 1, but it can be set smaller for pipelined operation.
- Delay Combinational delay.

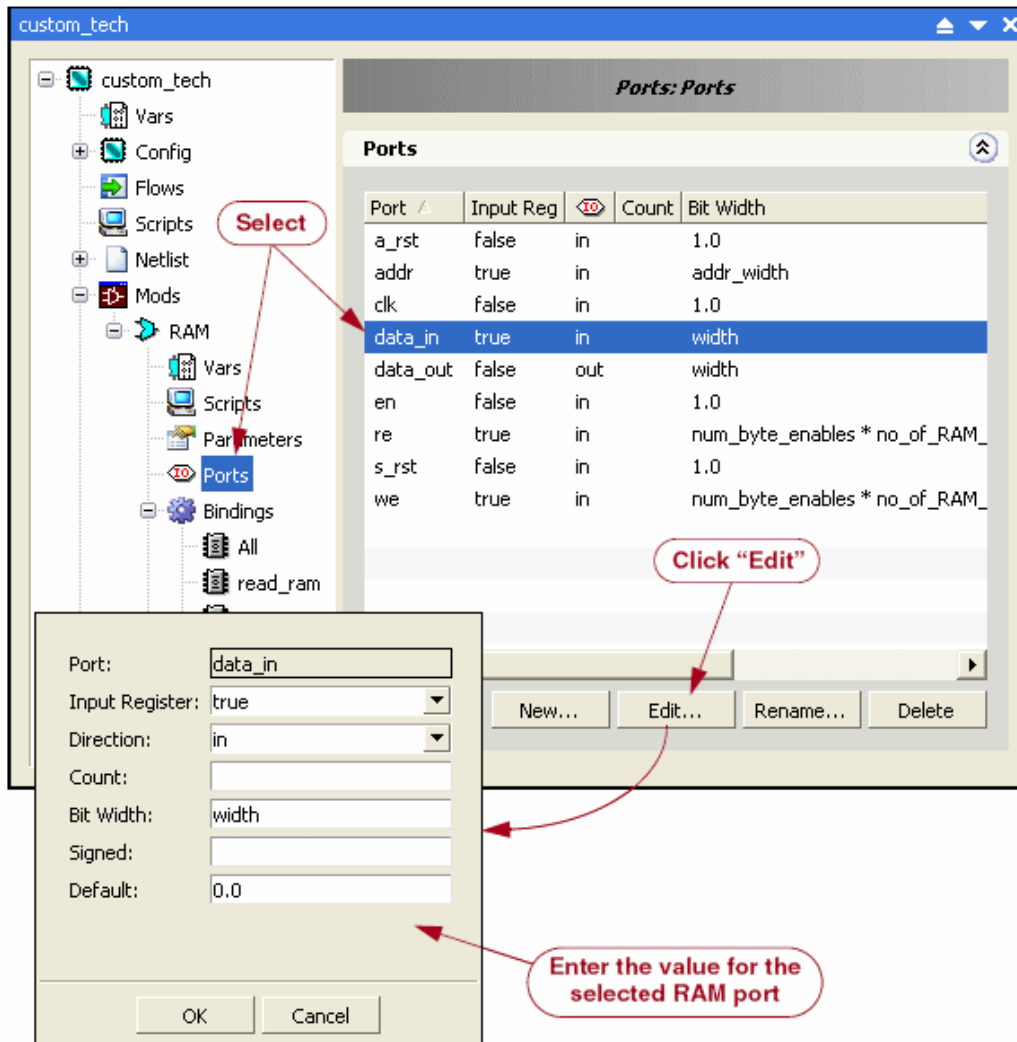
Figure 2-8. Editing RAM Timing Values



Editing RAM Ports

Open RAM component and select Ports. If you have a RAM with an asynchronous read, then you should remove the input register for the read address. The clock, reset and enable should never have an input register.

Figure 2-9. Editing RAM Ports



Saving Libraries

Library edits are not saved until you explicitly save the library to disk with one of the following options:

- Select “**Save All Libraries As**” from the “Library Tasks” window or the File pulldown menu. This command saves all open libraries together as a single file. The default file name is `Catapult.lib`.
- From within the Library Explorer window, right-click on a library and select the **Save As...** menu item. The default file name is the name of the selected library.

In both cases a file system browser is provided for you to navigate to the location where you want the file saved. Place the library in the Catapult search path to make available to Catapult C Synthesis sessions. If you save it in an alternate location, you must add that path to the

Catapult C Synthesis library search path (Use **Tools > Set Options > Component Libraries** in the Catapult C Synthesis application).

Chapter 3

Library Characterization

Characterization is the process of synthesizing the components in a library in order to obtain their area and timing characteristics. The synthesis is performed by one of the supported synthesis tool (Synopsys Design Compiler, Cadence RTL Compiler, or Magma Talus Design) and the area/timing characteristics are then saved with the components in the library. Components in the Catapult “Base ASIC Library” template are characterized at multiple performance points, producing a robust library with which Catapult Synthesis can deliver highly accurate area estimates. See [“Multi-Point Characterization”](#) on page 76 for more information.

Note



A *module* (MOD) is a library component implementing one of the operators available to the scheduler. For example the MODs `mgc_mul` and `mgc_mul_pipe` (multiplier and pipelined multiplier) implement the functionality of the multiplication operator MUL.

A QMOD is module with all characterization parameters explicitly specified. For example `mgc_and(1,2)` is a qualified configuration of `mgc_and` with one output and two inputs (a single AND2 gate).

Library characterization consists of the following steps:

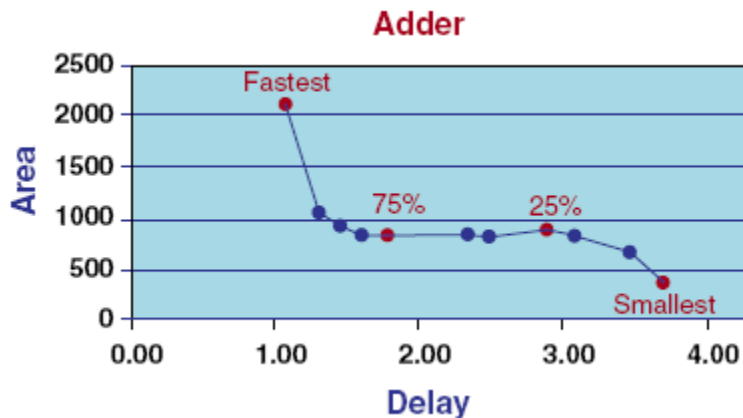
1. In the Library Explorer window, select the items to characterize. If the top level is selected, the entire library is characterized. Expand the tree to select individual MODs, QMODs or QMOD datasets.
2. Launch the characterization process on the selected items. See [“Characterizing Libraries or Components”](#) on page 78.
3. After characterization is complete, use the Plot window, reports and transcript files to view and analyze the characterization data and resolve any failures.
4. If errors occurred during characterization, use the characterization transcript to diagnose the problem. Refer to [“Troubleshooting Library Failures”](#) on page 88.

During the characterization process, the Catapult C Library Builder commands are displayed in the Transcript window, giving you a record of the entire session. You can scroll through the transcript and print or save it.

Multi-Point Characterization

During the characterization process, Catapult C Library Builder collects multiple sets of characterization data (area and timing) for each component. You can think of each data set as a point along an area versus delay graph of the component. For example, Figure 3-1 shows how the graph of a 16-bit Adder might look.

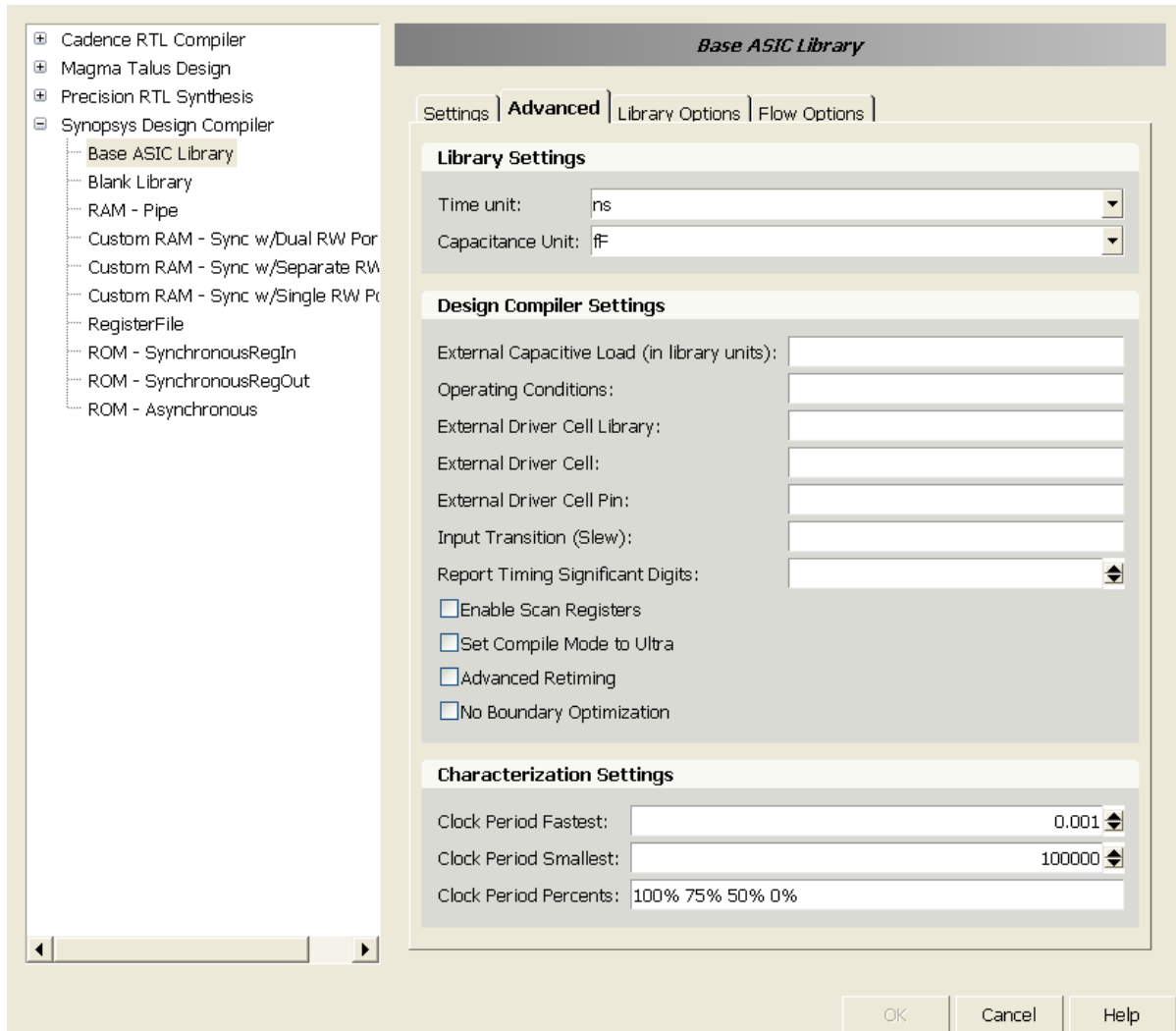
Figure 3-1. Adder Characteristics Graph



Typically the characteristics of the two end points (Fastest and Smallest) are rarely used in designs. So the Library Builder allows you to specify multiple points between the end points at which area and timing data will be collected. The target points are specified as percentages of the range bounded by the fastest and smallest points.

By default, four data sets are collected and their distribution is: 100%, 75%, 50% and 0% (“Fastest” = 100% and “Smallest” = 0%). You can specify a different number of data sets and/or different percentages when creating a new library. Those options are on the Advanced tab of the Library Creation dialog box, shown in Figure 3-2. Refer to [“Creating a New Library”](#) on page 51 for detailed information.

Figure 3-2. Multi-Point Characterization Settings on Library Creation Dialog



The characterization process works as follows for each component:

- Obtain “fastest” data set:
Run the synthesis tool with a target delay constraint specified by the **Clock Period Fastest** setting. (default is 0.01 ns.)



Caution

The clock periods must be able to generate the fastest design, i.e. the period must be small enough to constrain a simple logic gate such that it does not meet timing. If the tool is able to easily meet timing the curve described on the previous page will not be accurate.

- Obtain “smallest” data set:
Run the synthesis tool with a target delay constraint specified by the **Clock Period Smallest** setting. (default is 100,000 ns.)

- Obtain each intermediate data set:
Use the actual delay values obtained from the “fastest” and “smallest” characterizations to calculate the target delay constraints to be used for the intermediate characterization runs. Then run the synthesis tool for each intermediate point. Intermediate points are specified by the **Clock Period Percentages** setting.

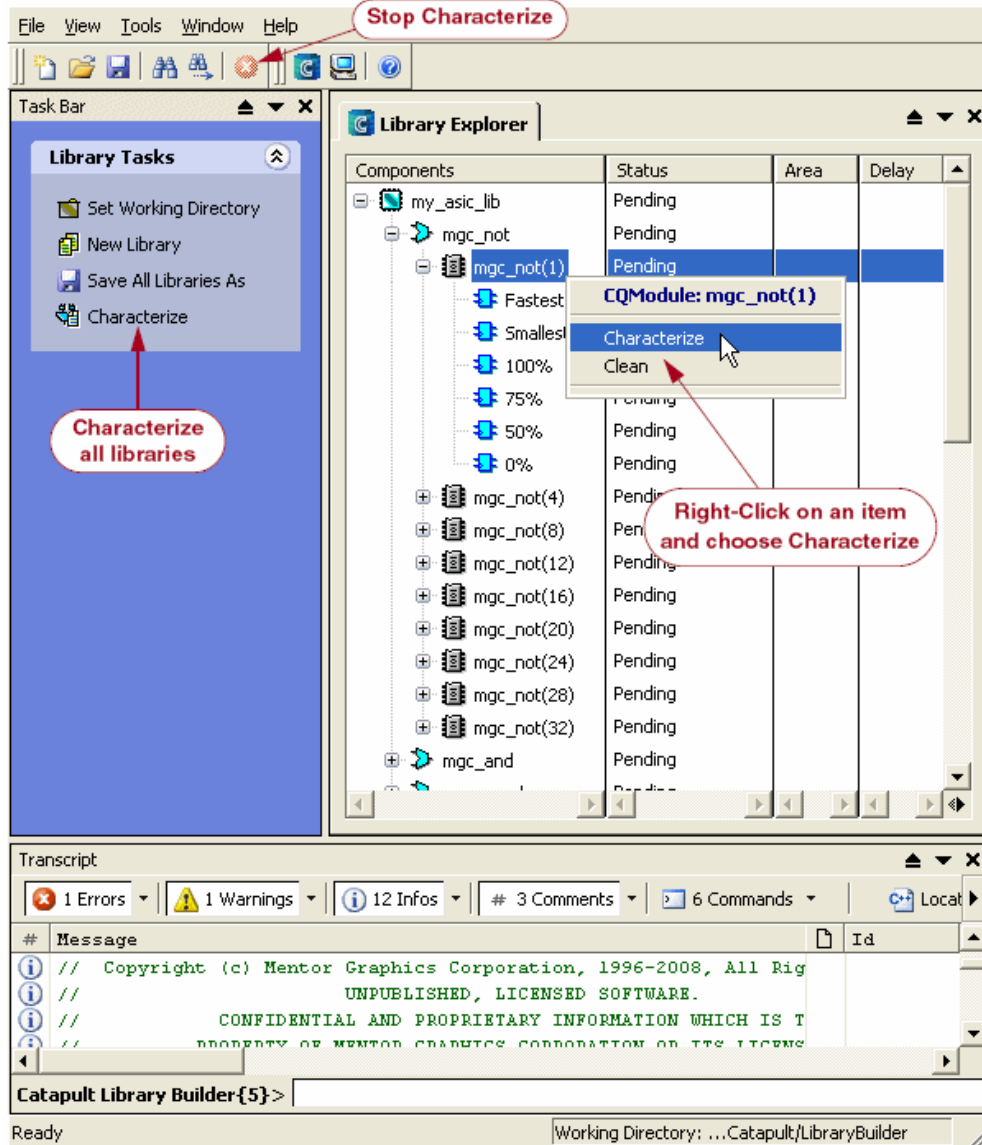
For ASIC libraries, the Library Explorer window displays “Status”, “Area”, “Delay”, “Clockperiod” and “Minclkprd” columns. The ClockPeriod column is the clock period to which the design was constrained for characterization. The MinClkPrd column is for sequential components, this includes registers. The minimum clock period is the fastest time that the component can be scheduled. The delay is the maximum, **input to register**, or **register to register** delay. The Delay column for these components refers to the **final register to output** delay. The “Minclkprd” column will not appear until a pipelined component is selected in the “Components” column.

Characterizing Libraries or Components

Once the library is loaded into the Library Explorer window, right-click on the object you want to characterize and choose the “**Characterize**” command. The characterize command operates on all objects hierarchically contained under the selected object. For example, if a module is selected, each qualified module it contains will be characterized. Similarly, if the library object is selected, the entire library will be characterized.

Select “**Characterize**” on the Task Bar to characterize all libraries in the Library Explorer window. Figure 3-3 illustrates how to initiate the characterization command.

Figure 3-3. Characterizing the Library or Component



As the characterization process runs, it displays information about its progress in the Status column of the Library Explorer window, as well as in the Transcript window. A full transcript of the characterization task is automatically saved in the project directory (see “[Viewing the Characterization Transcript](#)” on page 87).

To stop a characterization job in progress, click on the **Stop** button on the tool bar. It usually take a minute or so for the job to terminate. After it terminates, the component(s) that were being characterized will have a “**Failed Generate**” status.

When characterizing a large number of components, the Library Builder queues a series of characterization “tasks”, and each task will contain a multiple components. The maximum

number of components that can be grouped into a task is configurable. To adjust the default setting, edit the “**Maximum Components Per Task**” option (Refer to “[Set General Options](#)” on page 24).

Figure 3-4 shows an example of the how the Library Explorer window might appear while a characterization job is in progress. For this example, all of the “mgc_equal” components are being characterized, and the “Maximum Components Per Task” option has been set to four. The figure shows the first group of four components is in progress, indicated by the “Generating” status. The remaining mgc_equal components have a “Queued” status, and will be processed in turn.

Figure 3-4. Characterization Status Information

Components	Status	Area	Delay	Clockperiod	Minclkprd
mgc_reg_neg(32,1)	Passed				
Fastest	Passed	1139.1	0.38	0.001	0.0
Smallest	Passed	1139.1	0.38	100000.0	0.0
100%	Passed	1139.1	0.38	0.38	0.0
75%	Passed	1139.1	0.38	0.38	0.0
50%	Passed	1139.1	0.38	0.38	0.0
0%	Passed	1139.1	0.38	0.38	0.0
mgc_equal	Generating				
mgc_equal(4)	Generating				
Fastest	Generating			0.001	
Smallest	Generating			100000.0	
100%	Queued				
75%	Queued				
50%	Queued				
0%	Queued				
mgc_equal(8)	Generating				
mgc_equal(12)	Queued				
mgc_equal(16)	Queued				
mgc_equal(20)	Queued				
mgc_equal(24)	Queued				
mgc_equal(28)	Queued				
mgc_equal(32)	Queued				
mgc_decode	Pending				
mgc_shift_l	Failed Generate				
mgc_shift_r	Failed Invocation				
mgc_shift_bl	Pending				

Figure 3-4 also shows some of the other status values for characterization, such as “Passed”, “Pending” and “Failed Generate”. Table 3-1 defines the meaning of the status values that may appear in the **Status** column:

Table 3-1. Component Characterization Status

Status	Description
Pending	Component currently has no characterization data and is not currently queued for characterization.
Queued	Component is in the queue for automatic characterization.
Configuring	Initial state of the automatic characterization process which configures the design constraints and generates the characterization task script.
Generating	Characterization task is running. The component is being synthesized and the resulting area and timing characteristics are being collected.
Passed	Characterization task completed successfully and area/timing data was obtained.
Failed Analyze	An error was encountered when parsing the transcript to collect timing and area estimates for the design.
Failed Generate	Characterization task returned a non-zero exit status or did not obtain timing and area information.
Failed Invocation	The downstream synthesis tool failed to invoke.

Queuing Multiple Libraries for Multi-day Run

You can also set up a queue of libraries to characterize as follows:

1. Create all of the new libraries using the steps described previously in this section.
2. Click the **Characterize** task on the Task Bar to run the characterization process on all open libraries.

Library Characterization Results

If the library characterization passes, the status is set to “Passed” and the area and timing results display in the Library Explorer windows shown in [Figure 3-5](#).

This component was characterized using four-point characterization (default setting). So the display shows separate line items for each target point, plus the fastest and smallest points. Refer to [“Multi-Point Characterization”](#) on page 76 for information about configuring multi-point settings.

Figure 3-5. Characterization Passes

Library	Status	Area	Delay	IntPwr	LkgPwr	SwgPwr	Slack
mgc_sample-065nm-...	Passed						
mgc_not	Passed						
mgc_and	Passed						
mgc_and(1,2)	Passed						
mgc_and(1,4)	Passed						
mgc_and(1,8)	Passed						
mgc_and(1,12)	Passed						
Fastest	Passed	39.6	0.06	3.8764	0.238199	1.446	-0.06
Smallest	Passed	12.8	0.1	0.766871	0.0469254	0.142662	99999.9
100%	Passed	39.6	0.06	3.8764	0.238199	1.446	-0.06
75%	Passed	30.8	0.07	2.7175	0.162717	1.4984	0.0
50%	Passed	22.4	0.08	1.7066	0.0980857	0.881117	0.0
0%	Passed	12.8	0.1	0.766871	0.0469254	0.142662	99999.9
mgc_and(1,16)	Passed						
mgc_and(1,20)	Passed						
mgc_and(1,24)	Passed						

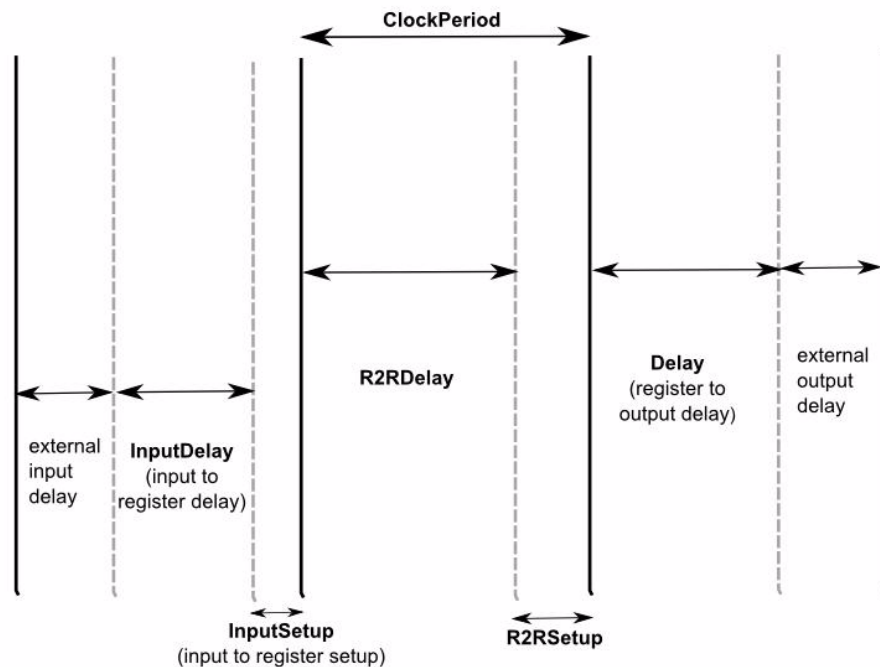
Delay Characterization Properties

After a successful characterization of a library/component, the Library Explorer window displays delay property values for each data set. These delay properties reflect the maximum values, regardless of the path. Depending on the component type, the following values display:

- **Delay** — If sequential, this value is the register to output value. If combinational, this value is the total input to output delay.
- **ClockPeriod** — Clock period constraint.
- **MinClkPrd** — Minimum clock period is the fastest time a pipelined component can be scheduled. It is either InputDelay + InputSetup, R2RDelay + R2RSetup, or Delay, whichever is largest.
- **Slack** — If sequential, this value is ClockPeriod minus MinClkPrd. If combinational, this value is ClockPeriod minus Delay.
- **InputDelay** — Input to register delay.
- **InputSetup** — Input register setup time.
- **R2RDelay** — Register to register delay.
- **R2RSetup** — Register to register setup time.

Figure 3-6 illustrates how these timing delay values are determined.

Figure 3-6. Sequential Timing Delay Diagram



Resetting the Data Before Another Characterization

Before you can rerun characterization on a library that has passed characterization, you must clear the values. Right-click on the target object and select **Clean** from the popup menu. However you can recharacterize a single component without having to clean it beforehand.

Once the values are removed from the Area and Timing fields, right-click on the target object and select **Characterize** from the popup menu.

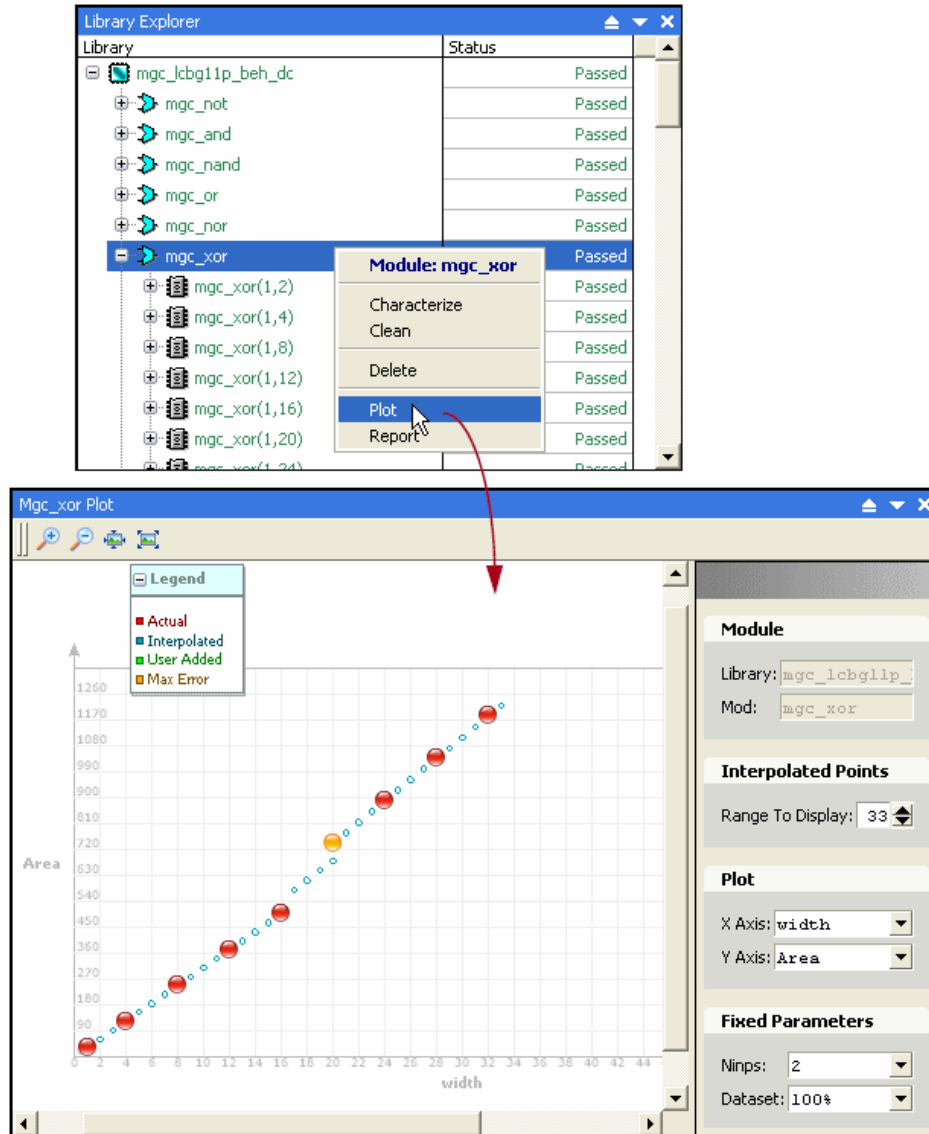
Plotting the Characterization Data

The "Plot" window provides a graphical view of the interpolated and measured characterization values for a user-specified range of qualified modules (QMODs) in a library module (MOD). The user interface of the Plot window allows you to dynamically plot different ranges of QMODs by selecting different combinations of properties and parameter values. It also allows you to add, remove and characterize QMODs.

As shown in Figure 3-7, the Plot window is accessed from the Library Explorer window by right-clicking on a MOD and selecting "Plot" from the popup menu. The main area of the Plot window contains a graph of the characterization data. The red and gold circles on the graph represent characterized QMODs, their *measured* values obtained from the downstream RTL synthesis tool. The blue circles are estimated characterization values for valid QMOD configurations that do not yet exist in the MOD. The gold color circle indicates the QMOD

whose measured value deviates the most from the Library Builder estimated value for it. In other words, the gold QMOD has the largest margin of error (Max Error) relative to all other measured QMODs on the graph.

Figure 3-7. Opening a Plot Window for the `mgc_xor` Module

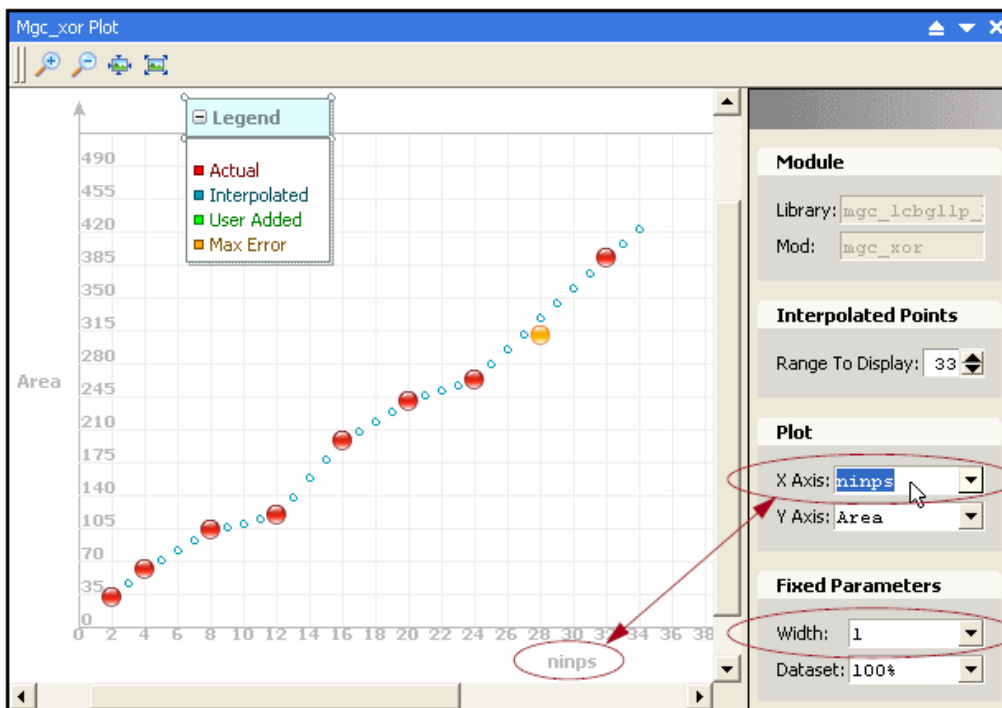


The Y-axis is the range of values for one of the MODs characterization properties. The X-axis is the range of valid values for one of its parameters. All other parameters are set to fixed values within their respective valid ranges. By default, the first property listed in the MOD’s “All” bindings is the initial Y-axis setting, and the first parameter listed is the initial X-axis setting. Similarly for each fixed parameter, the first valid value in its range is the default setting.

Use the interface panel on right side of the window to plot different configurations. For example, using the drop-down menu of X-Axis field, select the `ninps` parameter. The plot

automatically updates to graph *ninps* on the X-axis and *width* as a fixed value, shown in Figure 3-8.

Figure 3-8. Updated Plot with Ninps along X-Axis



In Figure 3-8, of all the QMODs in *mgc_xor* MOD, only the following nine measured QMOD configurations are plotted because they are the only ones that match the “Plot” and “Fixed Parameters” settings. The first parameter, “width,” is fixed at the value 1, and all values of the “ninps” parameter are plotted along the X-Axis. :

<code>mgc_xor(1,2)</code>	<code>mgc_xor(1,20)</code>
<code>mgc_xor(1,4)</code>	<code>mgc_xor(1,24)</code>
<code>mgc_xor(1,8)</code>	<code>mgc_xor(1,28)</code>
<code>mgc_xor(1,12)</code>	<code>mgc_xor(1,32)</code>
<code>mgc_xor(1,16)</code>	

The “Range To Display” field allows you to change the number of estimated data points (blue circles) that are plotted. Generally, the Range To Display number should be close to the X-axis value of the highest measured QMOD. For example, in Figure 3-8, the Range To Display is 33, and the highest “ninps” value in the range of QMODs is 32 (`mgc_xor(1,32)`).

Adding/Removing QMODs

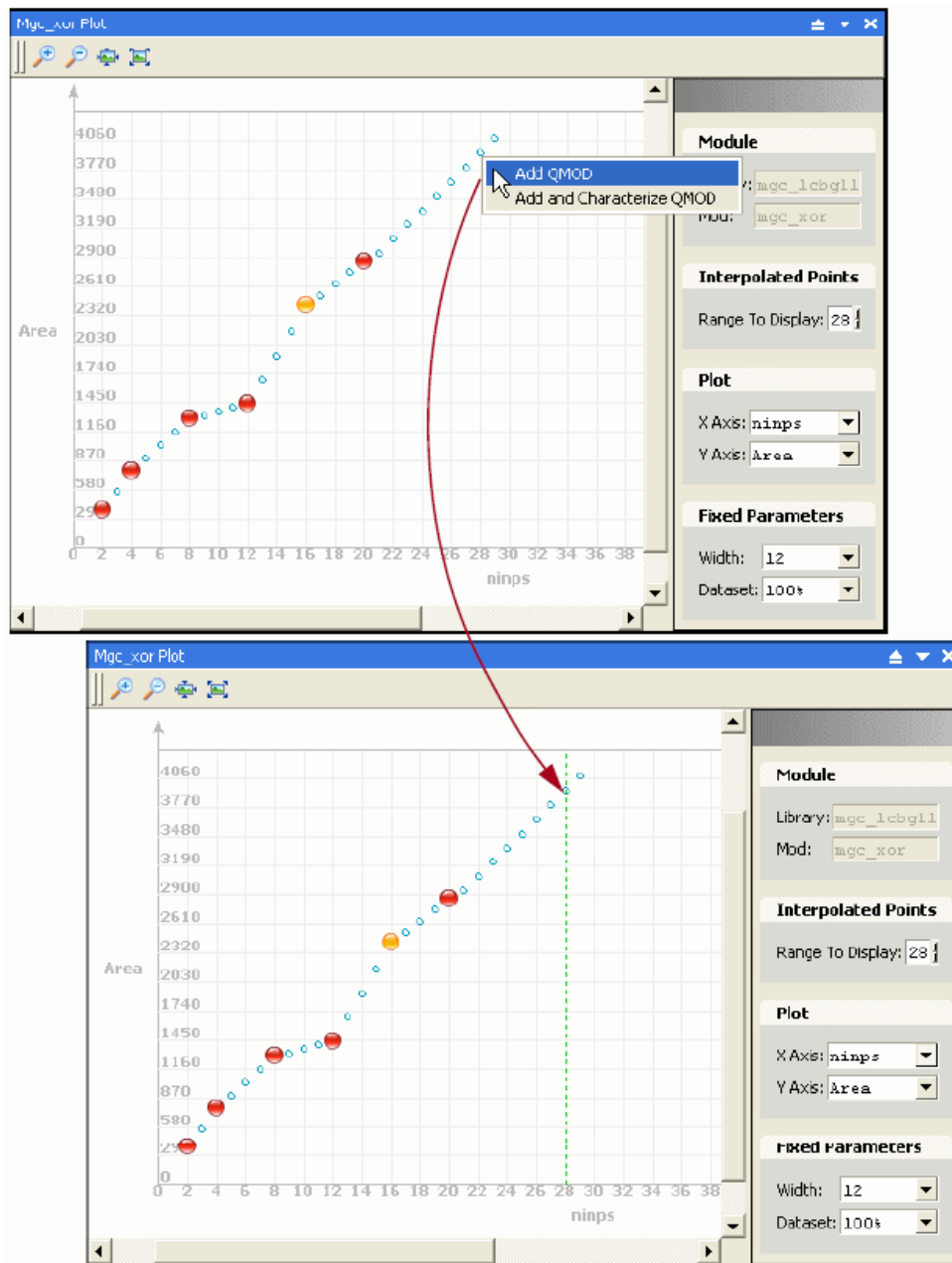
You can use the Plot window to add new QMODs to the MOD by right-clicking the blue circles and selecting “Add QMOD” (or “Add and Characterize QMOD”) from the popup menu. The new QMOD will be immediately added to the library and have the same parameter values as the

selected blue circle. If it is added but not characterized, it is represented on the graph as a green vertical line (unknown Y-Axis value). After it is characterized, it will become either a red or gold circle.

Figure 3-9 illustrates the procedure by adding the new QMOD `mgc_xor(12, 28)`. The Library Builder automatically updates the QMOD list in the Library Explorer window and in the module's "Qmods" section in the Library Editor window.

To remove a QMOD, right-click on it and choose "Remove QMOD" from the popup menu.

Figure 3-9. Adding QMOD from the Plot Window

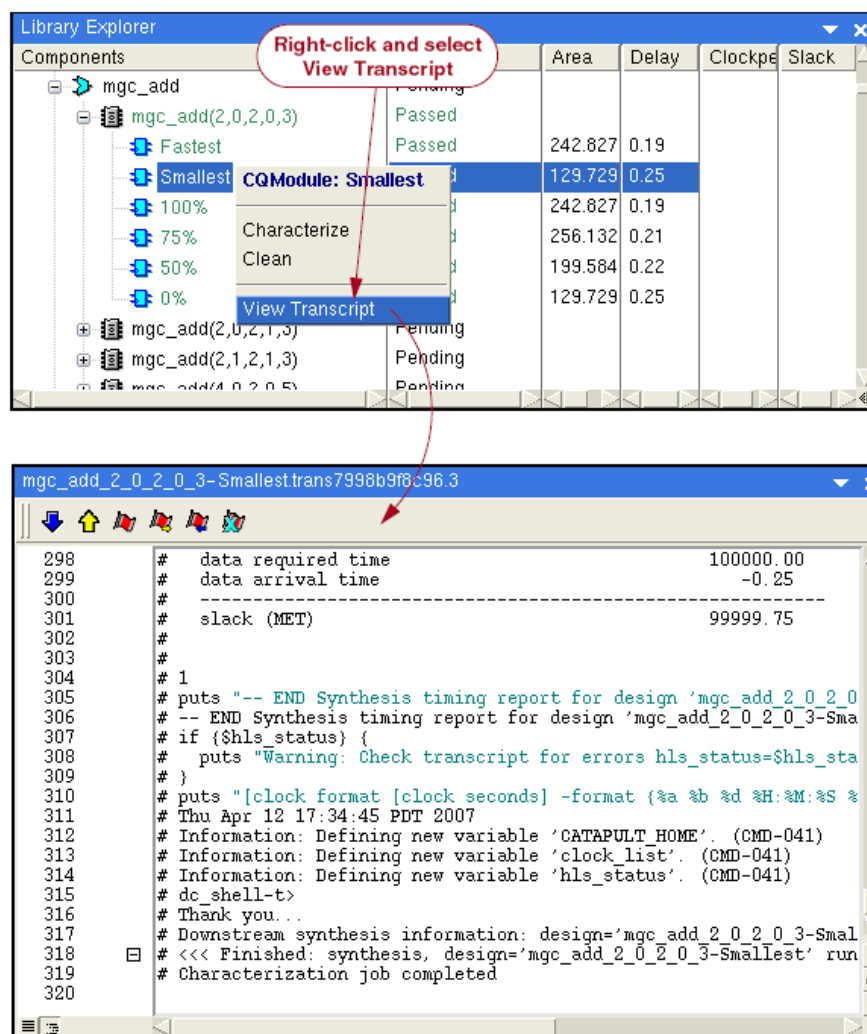


Viewing the Characterization Transcript

Once the library characterization starts, progress and commands being run are displayed in the Transcript window. Double-click on a component (or right-click and select **View Transcript** as shown in Figure 3-10) to see a transcript of the commands and tasks performed by Catapult C Library Builder.

If you run a “grouped” job, the transcript will collapse certain sections so that only the most relevant part are visible. It may be desirable to see others by expanding the +/- hierarchy points, or use the expand all.

Figure 3-10. Viewing the Characterization Transcript



Troubleshooting Library Failures

If the library characterization fails, Library Builder displays a message about the failure. If this occurs, you can launch Catapult C Synthesis to further investigate the failure. Use Catapult C Synthesis to modify the component and synthesize it. The area and delay data obtained in Catapult C Synthesis can be copied into your library in Library Builder.

Note



This option is not available if you are using Library Farm.

Procedure:

1. If the “**Remove Project Directories for Local Tasks**” option is enabled, disable it and run the “**Characterize**” command on the object again.

Library Builder always creates a Catapult C Synthesis project directory for each characterization task. By default, the project directories are deleted when the task is finished. Disabling this option will preserve the directory.

To disable the option, refer to “[Set General Options](#)” on page 24, or use the following command:

```
options set General RemoveProjectDirectories false  
# false
```

2. Launch Catapult C Synthesis by right-clicking on the failed library object in the Library Explorer window and select **Open Catapult Project** from the popup menu. This will open a Catapult C Synthesis session, load the failed module into the project and generate RTL. Refer to Figure 3-11.
3. Modify the component as needed to fix the problem and generate RTL.
4. Double-click on the “Synthesize rtl.vhdl” makefile to launch Design Compiler. Alternatively, right-click on “Synthesize rtl.vhdl” and select the **Launch DesignCompiler** command from the popup menu, as shown in Figure 3-11.
5. After obtaining the area and delay data in Catapult C Synthesis, use the Library Explorer window to manually enter the data for the failed object. Double-click on a data field to edit, such as **Area**, **Delay** or **Clockperiod** and enter the value. If a valid value is entered in the field, the status for the component will be changed to “Passed.” Refer to Figure 3-12.

Figure 3-11. Opening Failed Object in Catapult C Synthesis

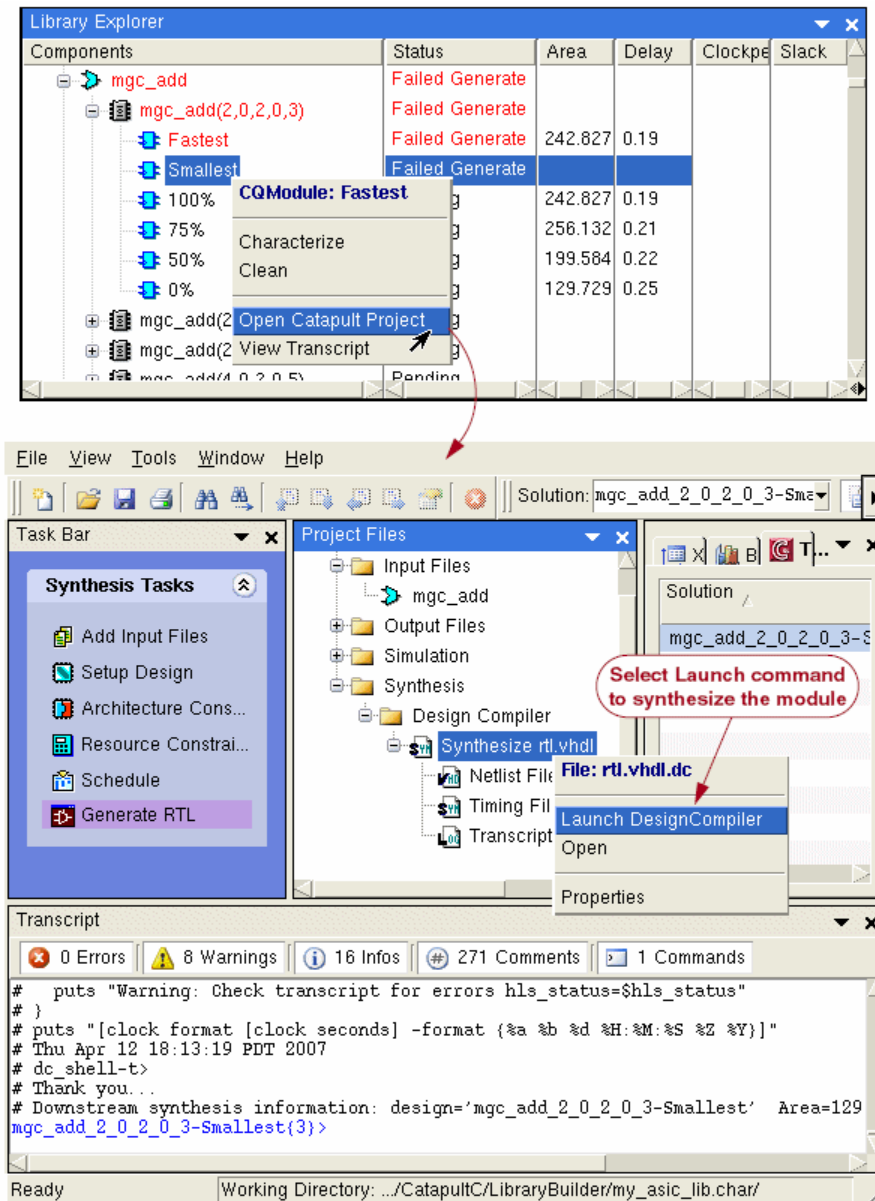
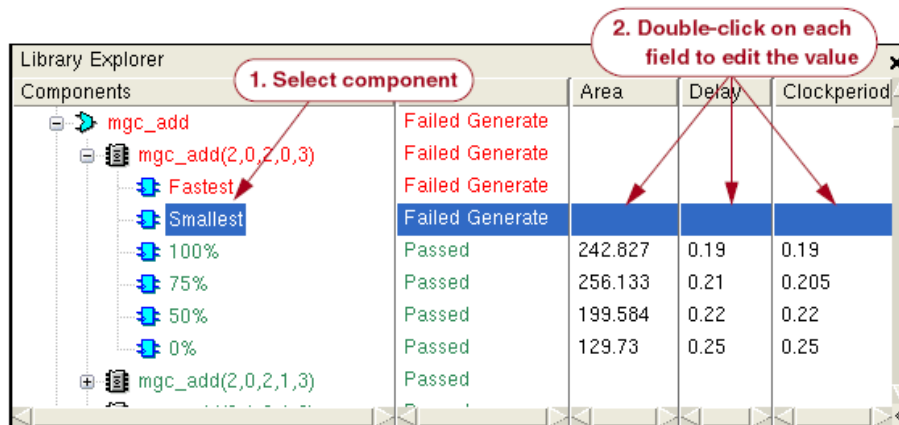


Figure 3-12. Entering New Area and Timing Values



Using the Library Farm

Library Builder contains a Library Farm tool that can be used distribute library characterization tasks to host computers on your network, thereby speeding up library characterization processing by allowing the tasks to run in parallel. Use the **Farm** window to assign local and remote hosts to the Farm and specify the task load for each host. After hosts are added as shown in [Figure 3-13](#) on page 92. The Library Farm window displays the list of hosts and the status of each task, such as “idle,” “passed” or “failed.”

The Farm can also work in conjunction with other load balancing software on your network, such as Load Sharing Facility (LSF) software. Refer to [“Set Farm Options”](#) on page 33 for detailed information about configuring the **Remote Shell Command** option.

Note



You should verify that the .cshrc sources all scripts required to enable the RTL synthesis tool and Catapult C Synthesis, so rsh is able to just start running the tools. Make sure to check the number of RTL synthesis tool licenses available or the Farm may consume more than are available.

Enabling and Configuring Library Farm Options

The factory default settings have the Farm disabled and the Farm window hidden. To make the window visible, select the “View > Farm” menu item. To enable the Farm, right-click in the Farm window and choose **“Enable Farm”** from the popup menu.

If you want the Farm to be enabled by default, modify the Farm option settings. Refer to the section [“Set Farm Options”](#) on page 33 for more information.

Configuring Library Farm to Use the Load Sharing Facility (LSF) software

To configure Library Farm to work with LSF, modify the **Remote Shell Command** setting on the Farm options page and set to an appropriate LSF command line. The format of the command line will include LSF command switches and Catapult internal variables as arguments. For information about these internal variables, refer to “[Set Farm Options](#)” on page 33. You can also set the Remote Shell Command option by using the “`options set`” command as shown in the examples below.

Note



You must add dummy “hosts” to the Library Farm in order to specify the number of jobs to submit to LSF. Any name may be used, except “localhost,” which is a reserved name. For example, you might add a host named “LSF” and a task limit of 5. That would imply that five LSF jobs may be submitted concurrently. You may adjust the number of tasks while Library Builder is characterizing.

Example 1: This example shows how the LSF `bsub` command line can be specified to run on a local machine:

```
options set Farm RshCommand {bsub -o %OUTPUTFILE% -q long  
-R "select[rh30_32b==1]" -L /bin/csh -K %COMMAND% -file %COMMANDFILE%}
```

- The `-o %OUTPUTFILE%` option specifies that upon completion of the job the standard output should be placed in the file named by `%OUTPUTFILE%` on the local host.
- The `-q long` option specifies the queue name for the job. The queue is named “long” in this example.
- The `-R "select[rh30_32b==1]"` option specifies the resource requirements of the job.
- The `-L /bin/csh` option starts a login shell and configures the environment with the user’s login scripts. This only matters if you are launching between different OS platforms. The `-L <arg>` option may be omitted if you are using the same OS for the submitted job.
- The `-K` option specifies that the `bsub` command should not return until the job is completed.
- The `%COMMAND%` option specifies the Catapult C Synthesis command to be invoked

Note



When the Library Builder is configured to use dedicated licenses for characterization tasks instead of Catapult C Synthesis licenses, you must use the `%COMMAND%` variable in the **Remote Shell Command** field. Do not enter a literal invocation command line. The variable automatically supplies special command line flags that are required for the new license.

- The `-file %COMMANDFILE%` option is the Catapult C Synthesis command line option to supply a script file.

Example 2: This example shows how to specify the LSF command to run from remote LSF “gate” machine:

```
options set Farm RshCommand {rsh %HOSTNAME% "cd %CWD%; bsub -o  
%OUTPUTFILE%  
-q long -R \"select [rh30_32b==1]\" -L /bin/csh -K %COMMAND% -file  
%COMMANDFILE%"} }
```

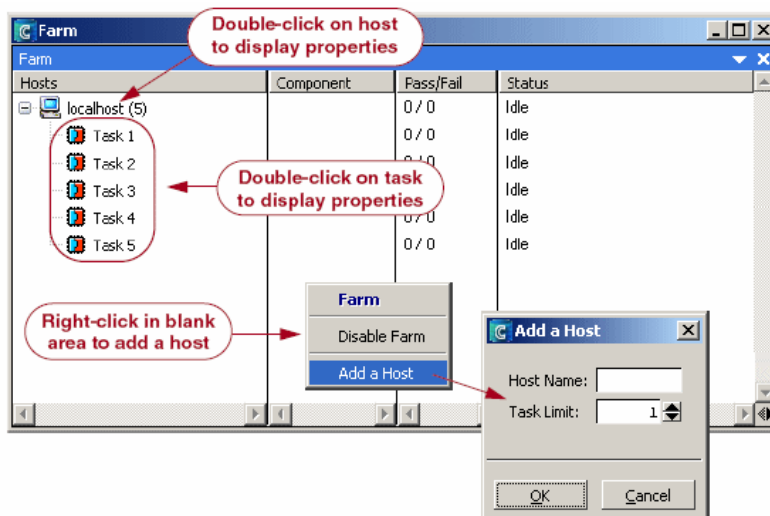
In this example the entire `bsub` command is passed as an argument to an `rsh` command. The `rsh` command takes two arguments, the name of the host machine (`%HOSTNAME%`) and the command string to be executed by `rsh`. Notice that the second argument is enclosed in double quotes and any nested quotes within it (such as `-R \"select [rh30_32b==1]\"`) must be escaped with a backslash.

Setting Up Library Farm Hosts

The initial Farm window is empty. Right-click in the blank area on the Farm window and choose **Add a Host** from the pop-up menu to open the Add a Host dialog box, as shown in Figure 3-13. Enter the name of the host computer and specify the number of tasks for that host, then click **OK**.

The host name “localhost” is a reserved name for the local machine. Jobs sent to localhost are run on the local machine without any network requirements. This can be useful for multiprocessor machines where you would like to run additional characterization tasks.

Figure 3-13. Library Farm Window



Host Menu: Right-click on a host name to open its pop-up menu. The menu allows you to add another task, remove the host, or see properties associated with the host.

- If you select **Add another Task**, a new task is added to the list in the Farm window. The pass/fail values are zero and the status is set to idle.
- If you select **Remove**, the host is removed from the Farm list.
- If you select **Properties**, the Farm Properties dialog box opens, with which you can change the number of tasks for that host. Use the scroll-arrows, or type the desired number of tasks and click **OK** or **Cancel**.

Task Menu: Right-click on a task to open its pop-up menu. The menu allows you to remove the selected task or view its properties.

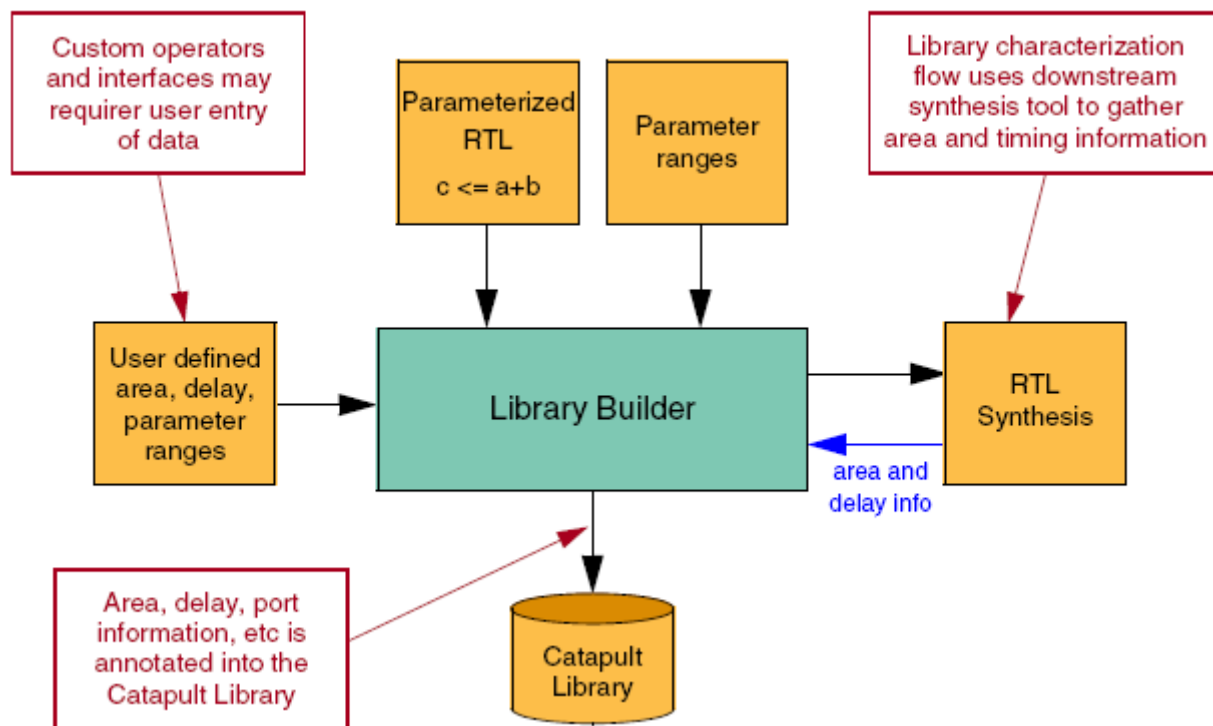
Chapter 4

Creating Custom Operators and Interfaces

Introduction	95
Creating the Custom Operator C++ Function	96
Creating a Library for the Custom Operators	96
Importing Custom Operators from C++ and HDL	97
Editing Libraries	105
Verifying the Custom Operator RTL and Custom C++ Function	127

Introduction

Catapult C Synthesis provides the user with built-in interface libraries that support simple protocols such as wire interface, two-way handshake, and memory interfaces. Similarly the base libraries for both ASIC and FPGA provide all of the operators required to schedule an algorithm. Although this combination of interfaces and base operators is usually sufficient, there are instances when a designer may wish to leverage some custom operator IP (MAC, Multi-Add, Wallace-tree multiplier, etc.) or interface (AMBA, APB, Avalon, PCIx, etc.). Library Builder allows user to integrate existing RTL IP into the Catapult design flow.



Creating the Custom Operator C++ Function

Catapult base libraries contain built-in operators that allow a C++ algorithm to be synthesized to RTL. These library operators correspond to the *, +, -, etc. operations in the C++ code. Custom operators are different in that they map an entire C++ function to a block of RTL. This can lead to huge performance gains in both ASIC and FPGA technologies when mapping to design ware or DSP blocks.

Custom operators require a corresponding C++ function call. This C++ function **MUST** have the same functionality as the operator RTL. Otherwise this will cause a simulation failure when verifying the C++ design against the RTL design.

A special pragma is used to indicate that the C++ function should be directly replaced with the custom operator RTL implementation.

```
#pragma map_to_operator "<operator name>"  
<C++ function>
```

For example:

```
#include <ac_int.h>  
#pragma map_to_operator "mul_pipe"  
ac_int<16> mul_pipe(ac_int<8> a, ac_int<8> b){  
    return a*b;  
}  
  
#pragma design top  
void mul_pipe_test(ac_int<8> a, ac_int<8> b, ac_int<16> &c){  
    c = mul_pipe(a,b);  
}
```

The function arguments must match operator ports and the only allowed output is the function return value. The operator name **MUST** match the operator name defined in the library.

Creating a Library for the Custom Operators

The first step in creating custom interfaces or operators is to create a blank ASIC or FPGA library. This can be done either in the Library Builder GUI or from the command line using “flow” commands.

Creating an ASIC Blank Library

Usage:

```
flow run /DesignCompiler/library add blank \  
    -libname <lib file name> \  
    -libtitle <UI lib title> \  
    -vendor <ASIC Vendor> \  
    -technology <Process technology> \  
    \
```



```
-link_library <Path to *.db> \  
-target_library <Path to *.db> \  
-selected <true/false>
```

Example:

```
flow run /DesignCompiler/library add blank \  
-libname example \  
-libtitle example \  
-vendor {LSI Logic} \  
-technology lcbg11p \  
-link_library lcbg11_wc.db \  
-target_library lcbg11p_wc.db \  
-selected true
```

Creating an FPGA Blank Library

Usage:

```
flow run /Precision/library add blank \  
-libname <lib file name> \  
-libtitle <UI lib title> \  
-manufacturer <Vendor name> \  
-family <Device family> \  
-part <Device package> \  
-speed <Device speed grade> \  
-selected <true,false>
```

Example:

```
flow run /Precision/library add blank \  
-libname example \  
-libtitle example \  
-manufacturer Altera \  
-family {Stratix II} \  
-part * \  
-speed 3 \  
-selected true
```

Importing Custom Operators from C++ and HDL

Library builder provides the “library import” command that can read in custom operators from C++ source code, and their corresponding modules from RTL netlists. The command parses C++ code to extract the interface of the operator (port names, port directions and bitwidths) and automatically annotate that data in the “Operator” section of the library. It parses the RTL netlist to extract the module port names, port directions, and generics or parameters, then annotates that information in the “Mods” section of the library.

The user must edit the library after importing to specify the bindings between operator and the module, as well as provide additional information such as latency and throughput information,

area, characterization/parameter ranges, etc.. Refer to “[Editing Libraries](#)” on page 105 for more information.

The import command has the following usage:

```
library import ?<switches>? ?<files>?

<switches>          Valid switches: (Optional)
-module <string>    Name of module to import
-operator <string>  Name of operator to import
-vhdl               Import VHDL netlist
-vhdl_libmap <name> <path>
                   VHDL library mapping (Required)
-property_map <propname> <propval>
                   Adds the property propname, propval to
                   the All bindings for all imported mods
                   (Required)
-port_default <portname> <default>
                   Adds the default value 'default' to the
                   named port (Required)
-non_char_param <parameter>
                   Makes the named parameter an HDL generic only (not
                   a characterization param)
-input_register <portname>
                   Adds the INPUT_REGISTER flag to the named
                   port (must be an input port) (Required)
-char_range <param> <range_str>
                   Adds the named range to the CHAR_RANGE of
                   the named parameter (Required)
-add_variable <varname,> <value>
                   Adds the named variable / value to the VARS
                   the named parameter (Required)
-vhdl_option <string>
                   Option to pass to VHDL parser
-verilog            Import Verilog netlist
-verilog_option <string>
                   Option to pass to Verilog parser
-get_tops           List top level modules
-libname <string>   Name of library to import to
-mod_type <ram|rom|inport|outport|inoutport|userop|userop_withstate>
                   Module type
    ram             ram
    rom             rom
    inport          inport
    outport         outport
    inoutport      inoutport
    userop          userop
    userop_withstate userop_withstate
<files>            files to import (Optional)
```

Importing Operators from C++ Functions

Use the “-library <string>” switch to specify a target library to annotate. Use the “-operator <string>” switch to specify the name of the custom operator function in the C++ file. The complete import flow is as follows:

1. Create a library or import into an existing library.
2. Import the custom operator function by using the “library import” command.
 NOTE: The operator must be imported before any modules that would be bound to that operator.
3. Import the corresponding HDL module.
4. Manually set up the pin bindings between the module and the operator.
5. Save the library

Custom operator functions must adhere to the following conditions in order to import properly:

- Return only one value
- Return value cannot be a struct

The following example illustrates the operator import flow. Consider the custom operator function defined in the file MAC8X8.cpp shown below:

```
#include "ac_int.h"

ac_int<32> MAC8X8(ac_int<8> a, ac_int<8> b ) {
    return 0;
}
```

The following script creates a new library named MY_LIB, imports the operator and corresponding VHDL module named MAC8X8, and then sets the pin bindings and other properties.

```
# Create the library
flow run /Precision/library add blank -libname MY_LIB -libtitle MY_LIB \
    -manufacturer Altera -family {Stratix II} -part * -speed *

# Import the operator
library import -libname MY_LIB -operator MAC8X8 MAC8X8.cpp

# Import the module
library import -module MAC8X8 -vhdl -vhdl_libmap work work \
    -vhdl_libmap altera_mf altera_mf -libname MY_LIB \
    -mod_type userop_withstate MAC8X8.vhd

# Add all the bindings and props manually
library add /LIBS/MY_LIB/OPERATORS/MAC8X8/PARAMETERS/opid \
    -- -MIN {} -MAX {}
library add /LIBS/MY_LIB/MODS/MAC8X8/BINDINGS/1:MAC8X8/PROPERTY_MAPPING \
    -- -opid opid
library add /LIBS/MY_LIB/MODS/MAC8X8/BINDINGS/1:MAC8X8/PROPERTY_MAPPING \
    -- -SeqDelay 1
library add /LIBS/MY_LIB/MODS/MAC8X8/BINDINGS/1:MAC8X8/PIN_MAPPING/aclr0 \
    -- -PINASSOC_TYPE SIGNAL -SIGNAL {[A_RST]} -PHASE 1
```

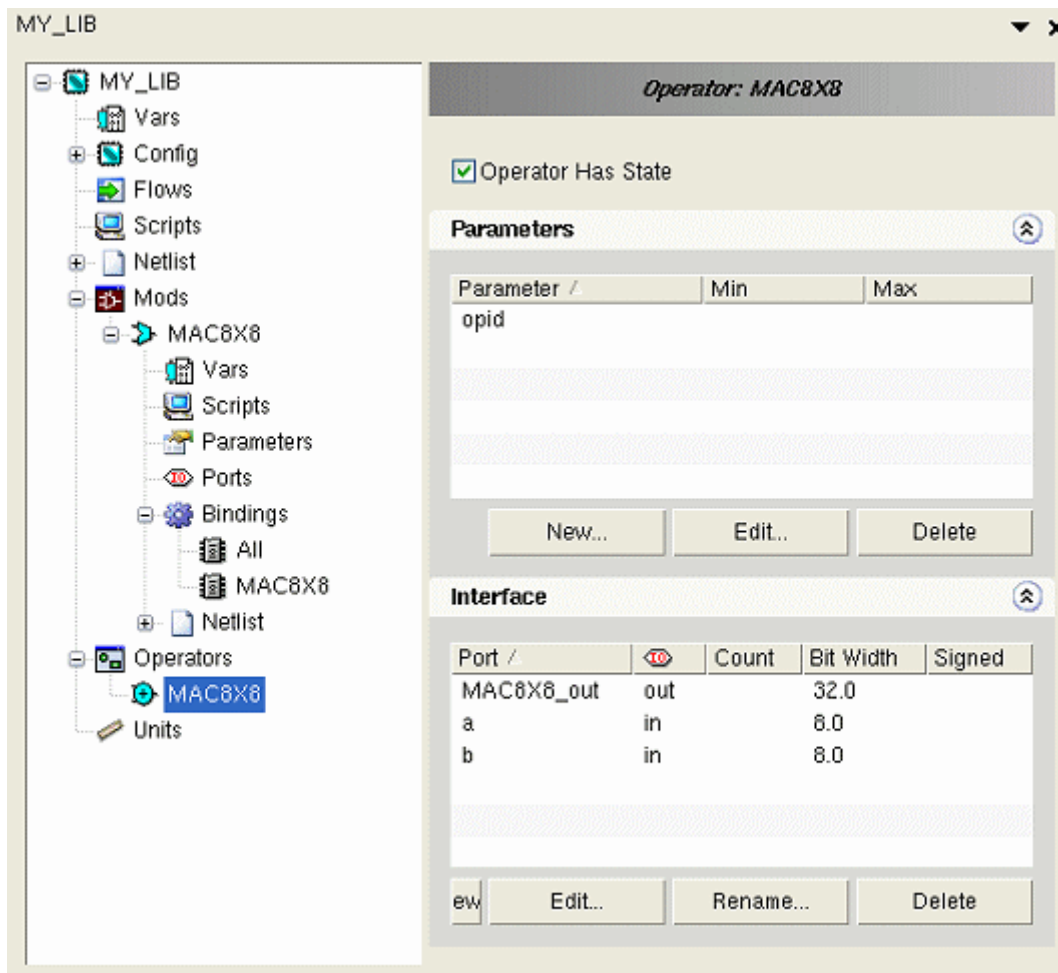
```

library add /LIBS/MY_LIB/MODS/MAC8X8/BINDINGS/1:MAC8X8/PIN_MAPPING/clock0
\
  -- -PINASSOC_TYPE SIGNAL -SIGNAL {[CLOCK]} -PHASE 1
library add /LIBS/MY_LIB/MODS/MAC8X8/BINDINGS/1:MAC8X8/PIN_MAPPING/dataa
\
  -- -PINASSOC_TYPE OPERATOR_PIN -OPERATOR_PIN a
library add /LIBS/MY_LIB/MODS/MAC8X8/BINDINGS/1:MAC8X8/PIN_MAPPING/datab
\
  -- -PINASSOC_TYPE OPERATOR_PIN -OPERATOR_PIN b
library add /LIBS/MY_LIB/MODS/MAC8X8/BINDINGS/1:MAC8X8/PIN_MAPPING/result
\
  -- -PINASSOC_TYPE OPERATOR_PIN -OPERATOR_PIN MAC8X8_out
library set /LIBS/MY_LIB/MODS/MAC8X8/PORTS/dataa \
  -- -INPUT_REGISTER true -SIGNED 1
library set /LIBS/MY_LIB/MODS/MAC8X8/PORTS/datab \
  -- -INPUT_REGISTER true -SIGNED 1

library save /LIBS/MY_LIB -filename MAC8X8.lib

```

Figure 4-1. Library with Imported Operator



Importing Netlists

The module type can be divided into three categories, interface, operator, and memories, as listed in Table 4-1.

Table 4-1. Categories of Module Types

Module type	Category
ram	Memory
rom	
inport	Interface
outport	
inoutport	
userop	Operator
userop_withstate	

Handling Memories

Library Builder needs to know the number of read/write ports on memories since data is expected to be concatenated onto a single RTL port. This is also true for the address and control. To see an example of how memories need to be structured for Catapult look at the install tree `$MGC_HOME/pkgs/siflibs` to see the built-in VHDL and Verilog models.

The special parameter `no_of_<module_name>_<port suffix>` is required by Library Builder to indicate the number of read/write ports. This must also be declared as a generic/parameter on the RTL. Library builder will look for these generics when importing the memory RTL. The `port_suffix` and order of the generics tells library builder whether the memory is single-port or multi-port.

The parameters and generics for the different flavors of memories are as follows:

- Singleport/dualport
 - `no_of_<module name>_readwrite_port`
 - Generic's range controls max number of ports. For example, a range of 1 to 1 is singleport, 1 to 2 is dual port, and so on.
- RAM with Separate Read/Write Ports
 - Library builder expects the generics/parameters in the following order
 - i. `no_of_<module name>_read_port`
 - ii. `no_of_<module name>_write_port`

- ROM
 - no_of_<module name>_read_port

Example - Importing a Singleport Ram

This ram uses a package called “ram_singleport_pkg” that is expected to be in the library “mgc_hls.”

Entity:

```
library ieee;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.all ;
USE ieee.std_logic_unsigned.all ;
LIBRARY mgc_hls;
USE mgc_hls.ram_singleport_pkg.all;

entity singleport_ram_rst is
  generic (
    words           : integer range 2 to 1000 := 2;
    width           : integer range 2 to 1000 := 2;
    addr_width      : integer range 2 to 1000 := 2;
    a_reset_active  : integer range 0 to 1;
    s_reset_active  : integer range 0 to 1;
    enable_active   : integer range 0 to 1 := 1;
    clock_edge      : integer range 0 to 1 := 1;
    no_of_singleport_readwrite_port : integer range 1 to 1 := 1
  );
  port (
    data_in  : in  std_logic_vector(width - 1 downto 0) ;
    addr     : in  std_logic_vector(addr_width - 1 downto 0) ;
    we       : in  std_logic;
    data_out : out std_logic_vector(width - 1 downto 0);
    clk      : in  std_logic;
    a_rst    : in  std_logic;
    s_rst    : in  std_logic;
    en       : in  std_logic
  );
end singleport_ram_rst ;
```

Import Script:

```
#Create the Library
flow run /DesignCompiler/library add blank -libname example \
  -libtitle example -vendor LSI -technology lcbg11p \
  -link_library lsi_lgbg11p_wc.db -target_library lsi_lgbg11p_wc.db

#Import the Netlist
library import -module singleport_ram_rst -vhdl -mod_type ram \
  -vhdl_libmap mgc_hls mgc_hls -libname example \
  -port_default we 0 -port_default a_rst {1.0 - a_reset_active} \
  -port_default s_rst {1.0 - s_reset_active} -input_register data_in \
  -input_register addr -input_register we \
  ram_singleport_fpga.vhd
```

Example - Importing an Input Interface

This interface is an input slave interface. There is no package or library mapping so the default work library is used.

Entity:

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;

entity fsl_slave is
    generic (
        C_DWIDTH      : integer := 32
    );
    port (
        -- Slave FSL Signals
        FSL_S_Read      : out std_logic;
        FSL_S_Data      : in  std_logic_vector(0 to C_DWIDTH-1);
        FSL_S_Exists    : in  std_logic;
        --Catapult side signals
        data_out        : out std_logic_vector(C_DWIDTH- 1 downto 0);
        fsl_rdy         : out std_logic;
        fsl_rd          : in  std_logic
    );
end fsl_slave;
```

Import Script:

```
#Create the Library
flow run /Precision/library add blank \
    -libname fsl_slave \
    -libtitle fsl_slave \
    -manufacturer Xilinx \
    -family * \
    -part * \
    -speed 3 \
    -selected true

#Import the Netlist
library import -module fsl_slave -vhdl -mod_type inport \
    -libname fsl_slave fsl_slave.vhd
```

Example - Importing a user operation

Entity:

```
LIBRARY IEEE ;
USE IEEE.std_logic_1164.ALL ;
USE IEEE.std_logic_arith.ALL ;
USE IEEE.std_logic_signed.ALL ;

use work.example_pkg.all;

ENTITY MAC IS
```

```
GENERIC(width: natural range 1 to 100:= 2;
        ph_arst : natural := 1);
PORT (
    clk      : IN STD_LOGIC ;
    rst      : IN STD_LOGIC ;
    a : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
    b : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
    c : OUT STD_LOGIC_VECTOR(width*2-1 DOWNTO 0)
) ;
END MAC ;
```

Import Script:

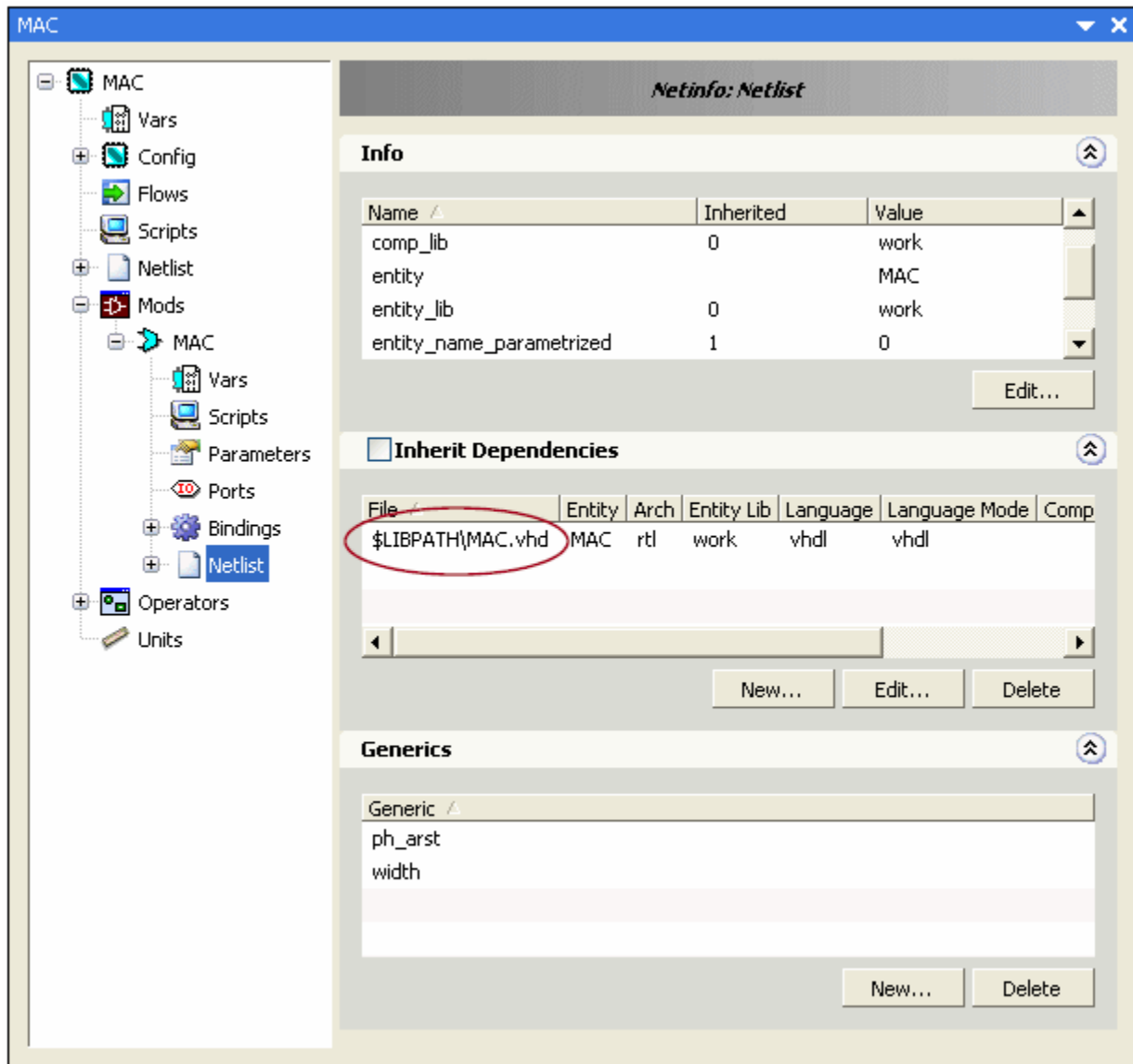
```
#Create the Library
flow run /DesignCompiler/library add blank \
    -libname MAC -libtitle MAC -vendor LSI -technology lcbg11p \
    -link_library lsi_lgbg11p_wc.db -target_library lsi_lgbg11p_wc.db

#Import the Netlist
library import -module MAC -vhdl -mod_type userop_withstate \
    -libname MAC MAC.vhd
```

Netlist Dependencies

Library builder will create the required netlist dependencies when the RTL netlist is imported. It is assumed that the RTL netlist will be stored in the same directory as the Catapult library. Library builder creates a netlist dependency using the \$LIBPATH variable which is set to point to the library (Figure 4-2).

Figure 4-2. Dependencies Imported from Netlist

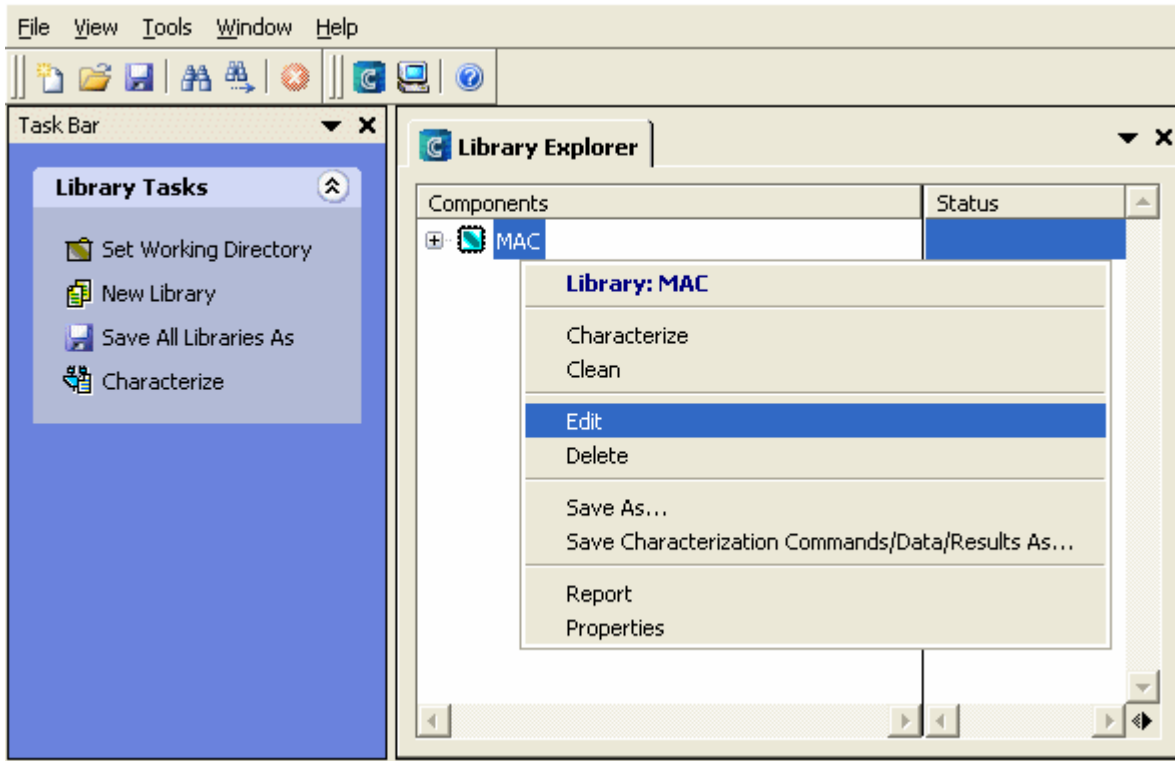


If the user wants the RTL in some other location, the netlist dependency must be edited in library builder.

Editing Libraries

After Library Builder has successfully imported the RTL netlist you must edit the library to provide additional information in order to build a valid library. This information consists of

things like latency and throughput behavior, parameter ranges, and so on. To edit the library, right-click on the library and select edit.



Modules

Modules tell Catapult about how to hook-up the RTL block. The module consists of the following:

- [Parameters](#)
- [Ports](#)
- [Bindings](#)
- [Pin Associations](#)
- [Property Mappings](#)

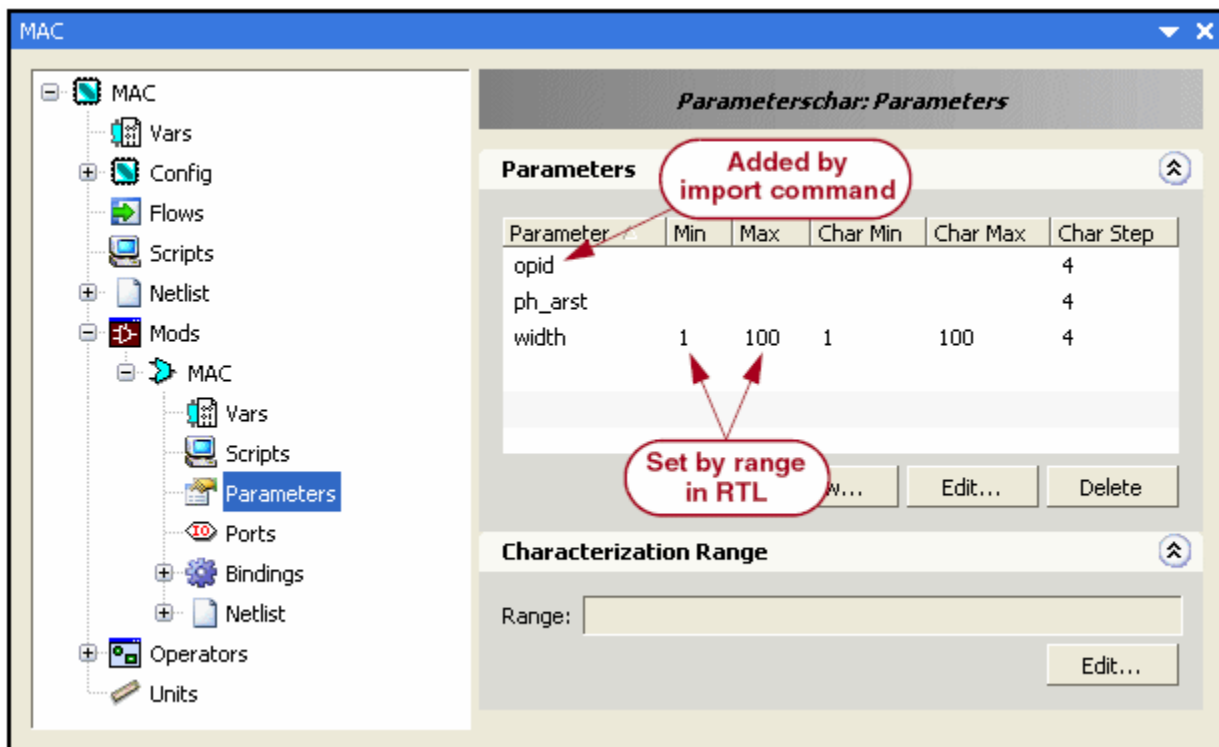
Parameters

Module parameters typically correspond to the generics/parameters on the RTL. However the import command will add an additional parameter for interfaces, memories, and operators with state. For memories, a parameter called “ram_id” is added. For interfaces, a parameter called “rscid” is added. For custom operators with state, a parameter called “opid” is added. The user does not need to modify these parameters. For all other parameters the min/max range and min/max characterization range must be set. The min/max range tells Catapult the allowable

range of the parameter. This controls things like how wide can a port be set on an interface, or how many words can a memory have. If the RTL generic/parameter has a range set on it, the import command will annotate this information for the min/max and characterization ranges. If the range is not specified in the RTL the user must set it manually. For example:

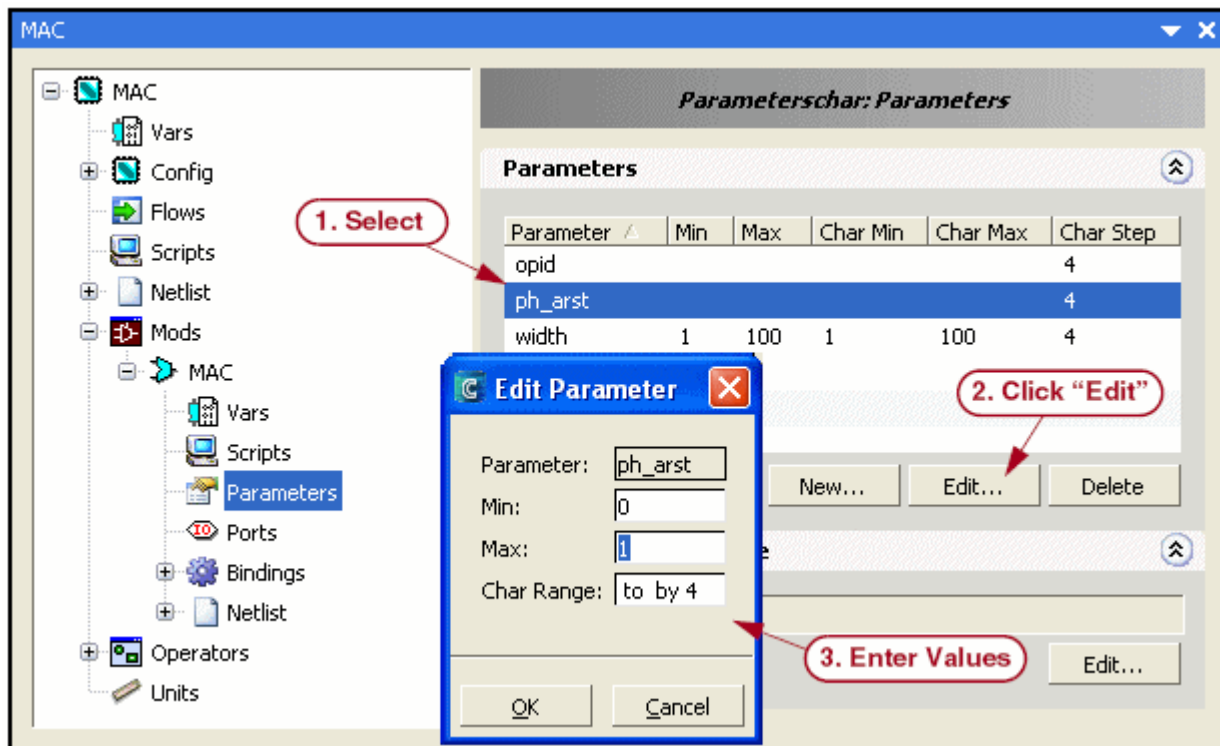
```
ENTITY MAC IS
GENERIC(width: natural range 1 to 100:= 2;
        ph_arst : natural := 1);
PORT (
    clk      : IN STD_LOGIC ;
    rst      : IN STD_LOGIC ;
    a : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
    b : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
    c : OUT STD_LOGIC_VECTOR(width*2-1 DOWNTO 0)
) ;
END MAC ;
```

Figure 4-3. Parameters Imported from Netlist



Parameters can be set manually by double-clicking on the parameter or selecting edit.

Figure 4-4. Parameters Set Manually



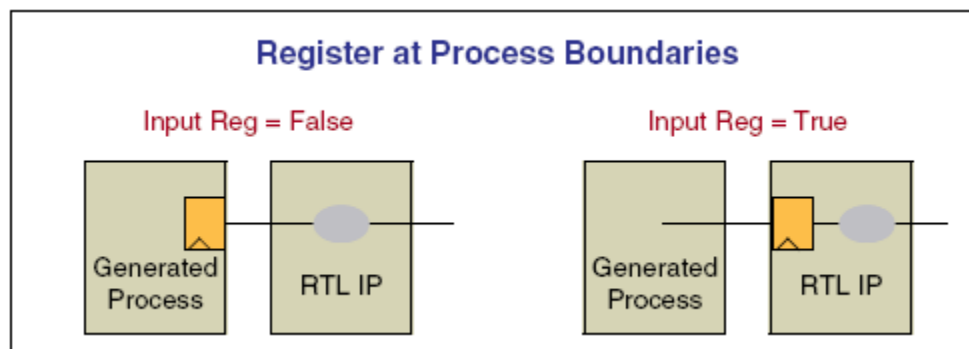
Ports

Ports correspond to the ports on the RTL. Most of the port information is added during the RTL import, but there are some fields that must be set by the user.

- **Input Register Setting**

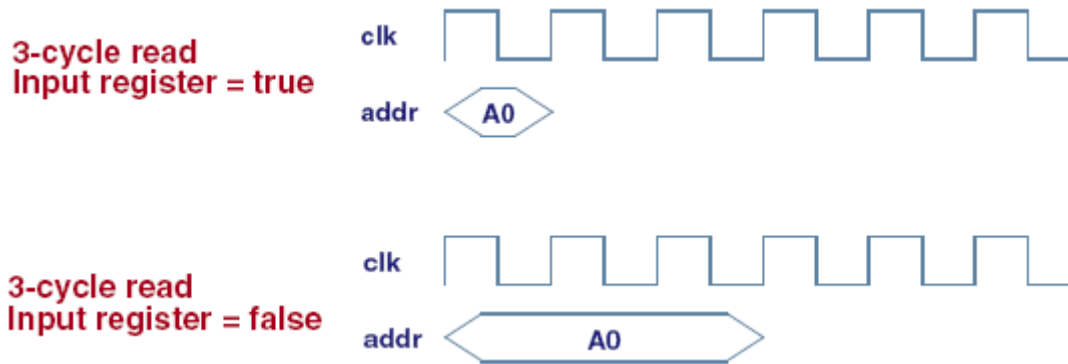
Catapult requires that either the outputs of the core process (RTL) that it generates or the inputs of the interface/operator RTL IP are registered. If input register is set to true on the module port, then Catapult does not put a register on the corresponding output of the core process. This is illustrated in Figure 4-5.

Figure 4-5. Input Register Setting



There are cases where you may wish to set input register to false even though the interface/operator RTL has input registers. If you require that the data and control inputs to the interface/operator RTL should be driven for multiple clock cycles, set the input register to false. This is only done for multi-cycle interface/operators.

Figure 4-6. Effects of Input Register Setting



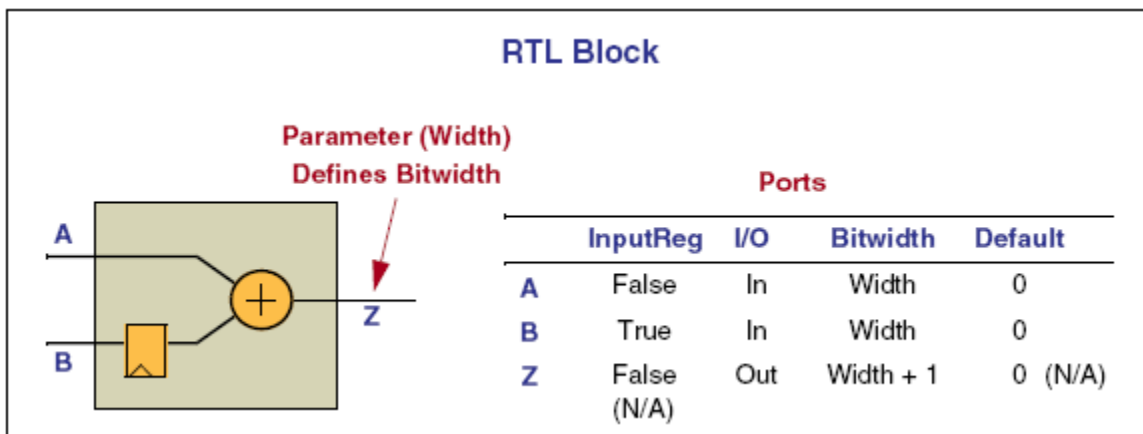
- **Sign Bit**

The port sign bit indicates that the port is signed or unsigned (1 == signed, 0 == unsigned). Catapult will sign extend inputs and outputs of signed ports. Single bit ports should leave the sign bit unassigned. Setting the sign bit on a single bit port results in `std_logic_vector(0 downto 0)`.

- **Default Value**

The default value port setting indicates what value should be driven on an interface/operator input port when the interface/operator is not being read/written. This setting is used for ports like memory write enables, resets, and so on.

Figure 4-7. Port Settings Overview

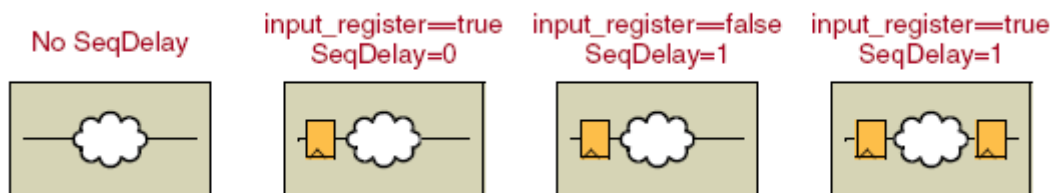


Bindings

Bindings tell Catapult how to connect to the RTL interface/operator. They also allow timing and area information to be specified. There are usually at least two bindings for every interface/operator, the “All” binding, and the operator binding. Operator bindings typically consist of read_port/write_port bindings for interfaces, read_ram/write_ram bindings for memories, built-in operators (add, mul, etc.), and user-defined for custom operators. The “All” binding is typically used to set area and timing information. This is done via a “property mapping.” Property mappings allow module parameters to be set, or to map operator parameters to module parameters. The following can be specified on the All binding:

- **Area**
 - Area of the module.
 - Can be specified as an equation.
- **Delay**
 - Combinatorial delay, or clock-to-out time of sequential components.
- **SeqDelay**
 - Sequential delay. Indicates the number of clock cycles required to complete the operation. For example:
 - $\text{SeqDelay} = \text{Number of Component Regs} - (\text{input_reg} == \text{true})?1:0$
 - For components with `input_reg == true`, `SeqDelay = 0`. That means that Data available after first clock edge.
 - No sequential delay means the component is combinatorial.

Figure 4-8. Effect of Input Register on SeqDelay



- **InitDelay**
 - Indicates how much a component can be pipelined. For example, `InitDelay = 2` means that the component can be pipelined down to `II=2`, but not `II=1`.
- **Always**
 - Design rule. Used to compute equations.

Area, Delay, SeqDelay, and InitDelay property mappings are case sensitive. They can be set from the command line (example below) or from the GUI (Figure 4-9).

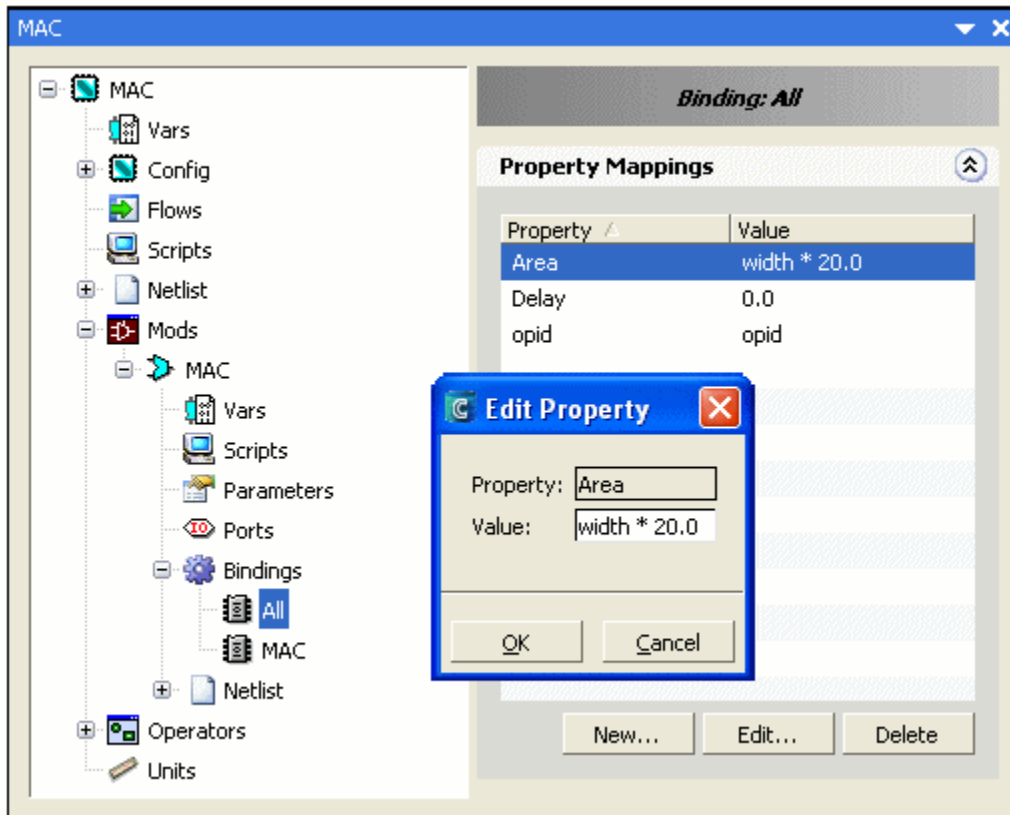
```
library add \  

  /LIBS/<library_name>/MODS/<module_name>/BINDINGS/all/PROPERTY_MAPPING\  

  \  

  -- -Area width*20 -Delay 1.2 -SeqDelay 1 -InitDelay 1
```

Figure 4-9. Using the GUI to Set Properties



Pin Associations

Pin associations are used to connect operator ports to RTL ports and to tell Catapult how to bind the RTL component when netlisting the final VHDL or Verilog output. The pin associations are set on the operator binding. [Table 4-2](#) describes the pin association types.

Table 4-2. Pin Associations

Pin	Description
Unbound	Left unconnected
OPERATOR_PIN	Binds to port on operator
CLOCK	Clock from Catapult C process.
ENABLE	Clock enable from Catapult C process.
S_RST	Synchronous reset from Catapult C process
A_RST	Asynchronous reset from Catapult C process.

Table 4-2. Pin Associations

Pin	Description
[EXTERNAL]	Connects the port of a bound internal or interface resource to an external port. Interface resources with EXTERNAL bindings must be bound.
DIRECT	Connects the port of a bound internal resource to an external port. DIRECT bindings are removed for interface resources.
GLOBAL	Connects the port of bound internal resources to a single external port when the internal port names are the same. GLOBAL bindings are removed for interface resources.
CONSTANT	Drives a constant value to the component port.
WAITON	Used for handshaking. The Catapult C process will wait for the component port to be driven high.

When the RTL netlist is imported, the appropriate binding is automatically added based on the module type setting. If the module type is set to “userop” the operator ports must be defined before the pin associations can be made. If the inport, outport, ram, or rom module type is set, the operator binding is added and you must specify the pin associations. [Table 4-2](#) describes the pins on each operator.

Table 4-3. Operator Pin Bindings

Binding	Data Port	Address
read_port	D	n/a
write_port	D	n/a
read_ram	D	I
write_ram	D	I

Using the fsl_slave example shown below, use the following steps to add the pin associations.

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;

entity fsl_slave is
  generic (
    C_DWIDTH      : integer := 32
  );
  port (
    -- Slave FSL Signals
    FSL_S_Read    : out std_logic;
    FSL_S_Data    : in  std_logic_vector(0 to C_DWIDTH-1);
    FSL_S_Exists  : in  std_logic;
    --Catapult side signals
```



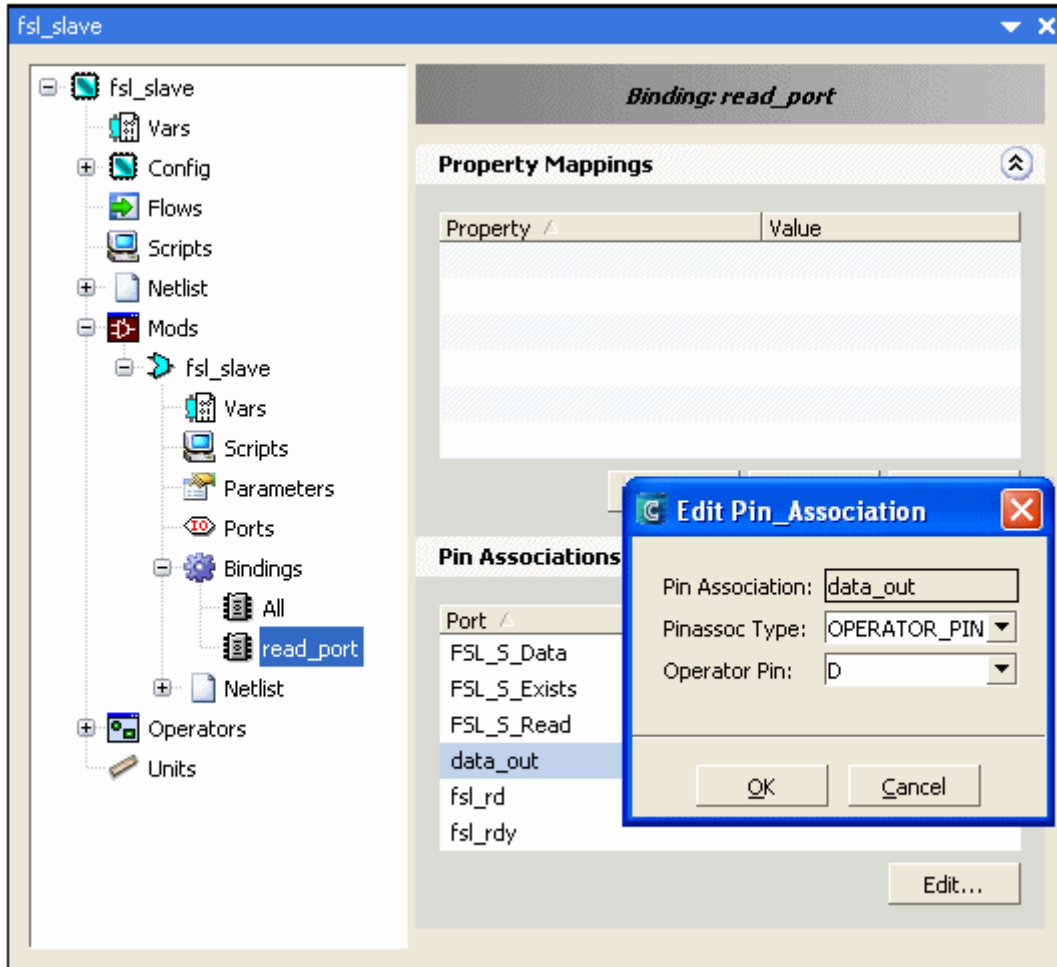
```

data_out      : out std_logic_vector(C_DWIDTH- 1 downto 0);
fsl_rdy       : out std_logic;
fsl_rd        : in std_logic

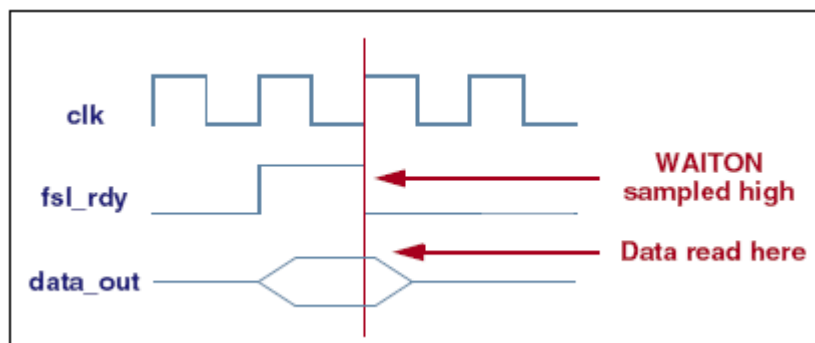
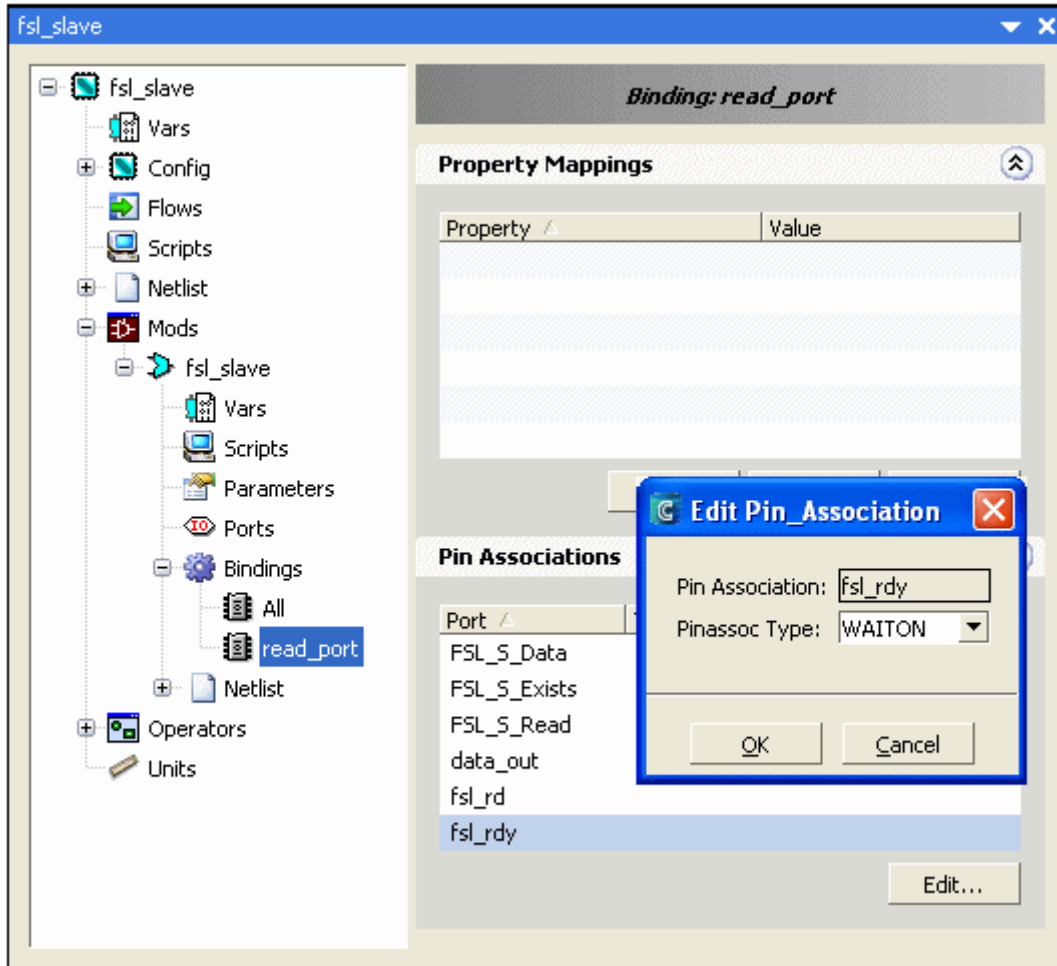
);
end fsl_slave;

```

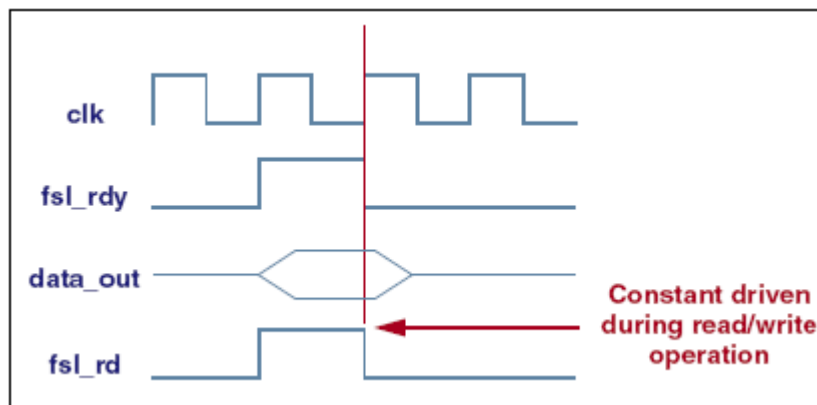
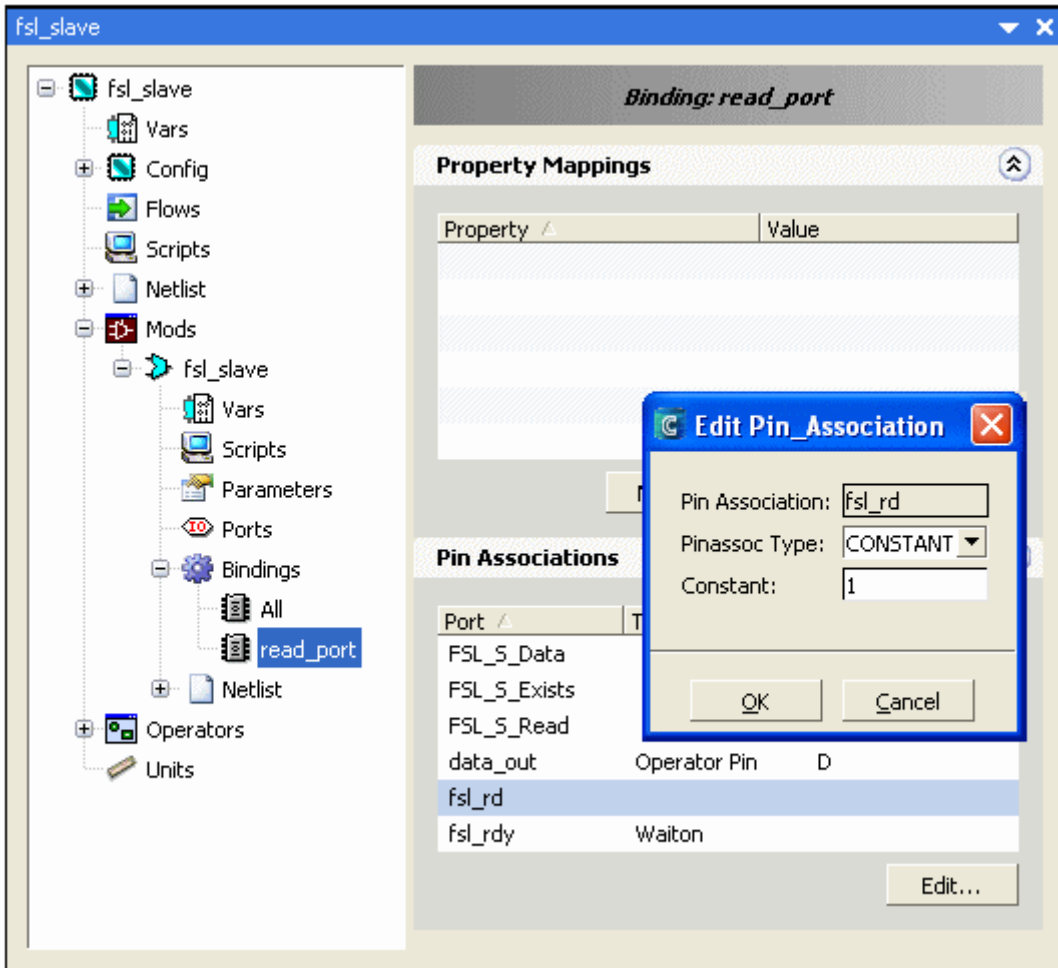
1. Connect the operator data port:



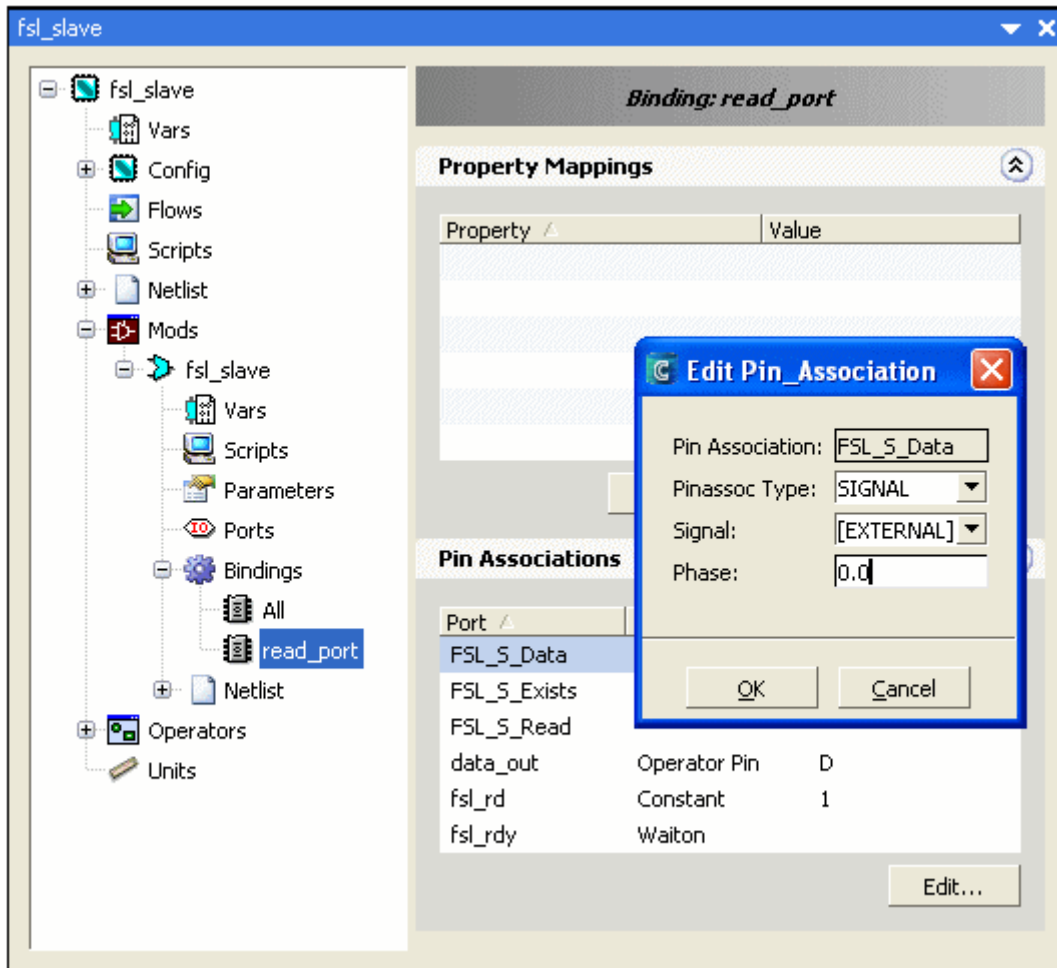
2. Add a handshake:



3. Drive a constant value to the component:



4. Connect all of the external ports:



Property Mappings

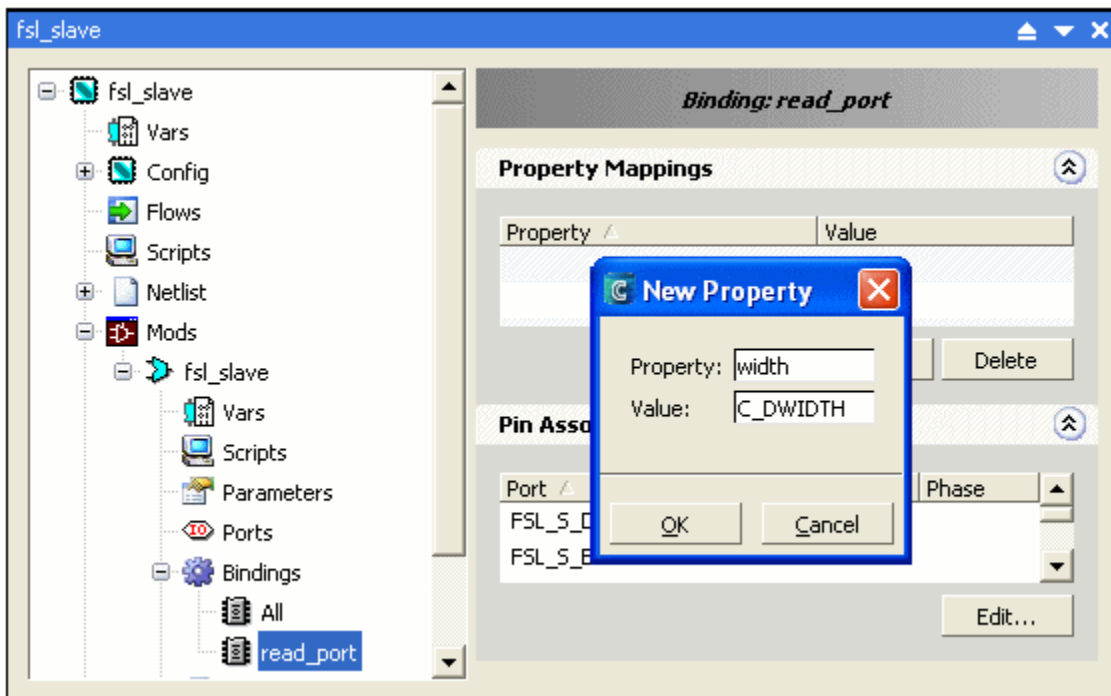
Property mappings allow operator parameters to be mapped to Module parameters which in turn may correspond to generics on the RTL. This allows an operator to control things like RTL port width, number of words in a memory, and so on. The built-in interface and memory operators have the following parameters. The “[library import](#)” command automatically maps the necessary operator parameters to their respective module parameters (such as width, id, etc.). If you are creating a library component manually, and that component uses built-in operators such

as read_port or write_port, you must manually map the operator parameters to the module parameter in the bindings.

Table 4-4. Property Mapping

Param	read_port	write_port	read_ram	write_ram	Custom operator	Comment
width	X	X	X	X		Set based on interface or memory width arch constraint
size			X	X		Set based on number of array elements
ramid			X	X		Set for each ram instance
id	X	X				Set for each interface instance
opid					X	Set for operators with state

For the fsl_slave example on [page 112](#), C_DWIDTH is used to set the RTL port width. Thus we want to create a property mapping between the “width” parameter of the read_port operator and C_DWIDTH



Creating RAM without a Reset

By default, the Library Builder provides reset pin associations on the module RAM bindings. To create a RAM without a reset, set the following properties on the *ALL* bindings to unconditionally suppress the asynchronous and synchronous reset pin association types:

- `no_s_reset = 1`
- `no_a_reset = 1`

The property value can also be set to the name of a module parameter (`width`, `en_active`). When the value of the specified parameter equals 1, the reset is suppressed.

You must modify the RTL model before simulation.

The `library add` command can also be used to suppress the reset on a RAM.

For more information on pin associations and setting properties, see the following topics:

- [“Pin Associations”](#) on page 111
- [“Property Mappings”](#) on page 116

Programmable Reset Polarity and Multiple Resets

Catapult can support both asynchronous and synchronous resets with either active high or active low polarity. Using a reset that is not described in the module causes Catapult C Synthesis to display an error when trying to bind the component. The error message in the transcript is similar to the following:

```
# Warning: Couldn't find library component for operator 'mul_pipe(8)' - no
available component (SIF-4)
# Error: incomplete component selection
```

Resets and control signals can be parameterized to set the desired polarity. This requires a special function to test the reset polarity.

```
FUNCTION active(lval: std_logic; ph: INTEGER RANGE 0 TO 1) RETURN BOOLEAN;
END example_pkg;

PACKAGE BODY example_pkg IS

    FUNCTION active(lval: std_logic; ph: INTEGER RANGE 0 TO 1) RETURN
    BOOLEAN IS
    BEGIN
        CASE lval IS
            WHEN '0' | 'L' =>
                RETURN ph = 0;
            WHEN '1' | 'H' =>
                RETURN ph = 1;
            WHEN OTHERS =>
```

```

        RETURN true;
    END CASE;
END active;

END example_pkg;

```

The reset polarity can then be passed to the RTL as a generic/parameter and the “special” function can be used to generate reset for the correct polarity. This same methodology can be used to handle clock enable polarity as well.

```

LIBRARY IEEE ;
USE IEEE.std_logic_1164.ALL ;
USE IEEE.std_logic_arith.ALL ;
USE IEEE.std_logic_signed.ALL ;
library work;
use work.example_pkg.all;

ENTITY mul_pipe IS
GENERIC(width: natural := 2;
        ph_arst : natural := 1);
PORT (
    clk      : IN STD_LOGIC ;
    rst      : IN STD_LOGIC ;
    a : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
    b : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
    c : OUT STD_LOGIC_VECTOR(width*2-1 DOWNTO 0)
    ) ;
END mul_pipe ;

ARCHITECTURE rtl OF mul_pipe IS

SIGNAL tmp : STD_LOGIC_VECTOR(width*2-1 DOWNTO 0) ;

BEGIN

PROCESS (clk)
    BEGIN
    IF active(rst,ph_arst)then
        c <= (others => '0');
    ELSIF(clk'EVENT AND clk = '1') THEN
        c <= a * b;
    END IF ;
    END PROCESS;

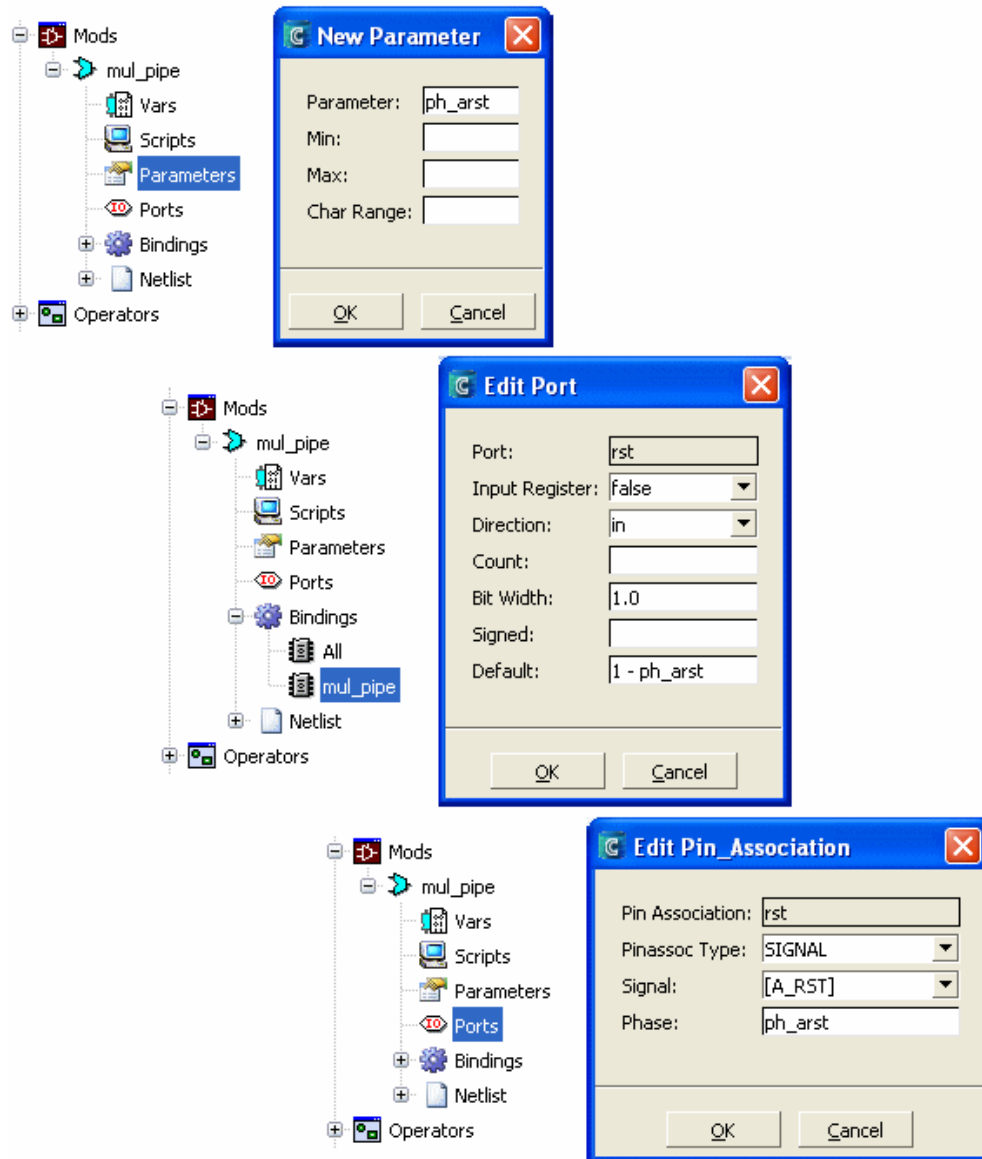
END rtl;

```

The library must also be modified to handle programmable reset polarity. The following steps must be taken:

- Add `ph_arst` parameter to module parameter
- Set the `rst` port default opposite the phase

- Use `ph_arst` to set the polarity on the binding



Manually Defining Custom Operators

The custom operator is created during the netlist import. But if the “library import” command was not used to import the operator port and parameter information from the custom C++ function, the user must edit the library to add that information. Consider the following pipelined multiplier RTL example that we wish to map to directly from a C++ function:

```
ENTITY mul_pipe IS
  GENERIC(width: natural := 2;
          ph_arst : natural := 1);
  PORT (
```



```

        clk      : IN STD_LOGIC ;
        rst      : IN STD_LOGIC ;
        a : IN STD_LOGIC_VECTOR(width-1 DOWNT0 0) ;
        b : IN STD_LOGIC_VECTOR(width-1 DOWNT0 0) ;
        c : OUT STD_LOGIC_VECTOR(width*2-1 DOWNT0 0)
        ) ;
    END mul_pipe ;

    ARCHITECTURE rtl OF mul_pipe IS

        SIGNAL tmp : STD_LOGIC_VECTOR(width*2-1 DOWNT0 0) ;

    BEGIN

        PROCESS (clk)
            BEGIN
                IF active(rst,ph_arst) then
                    c <= (others => '0');
                ELSIF (clk'EVENT AND clk = '1') THEN
                    c <= a * b;
                END IF ;
            END PROCESS;
        END rtl;
    
```

We first create the library and import the netlist with the module type set to user operation.

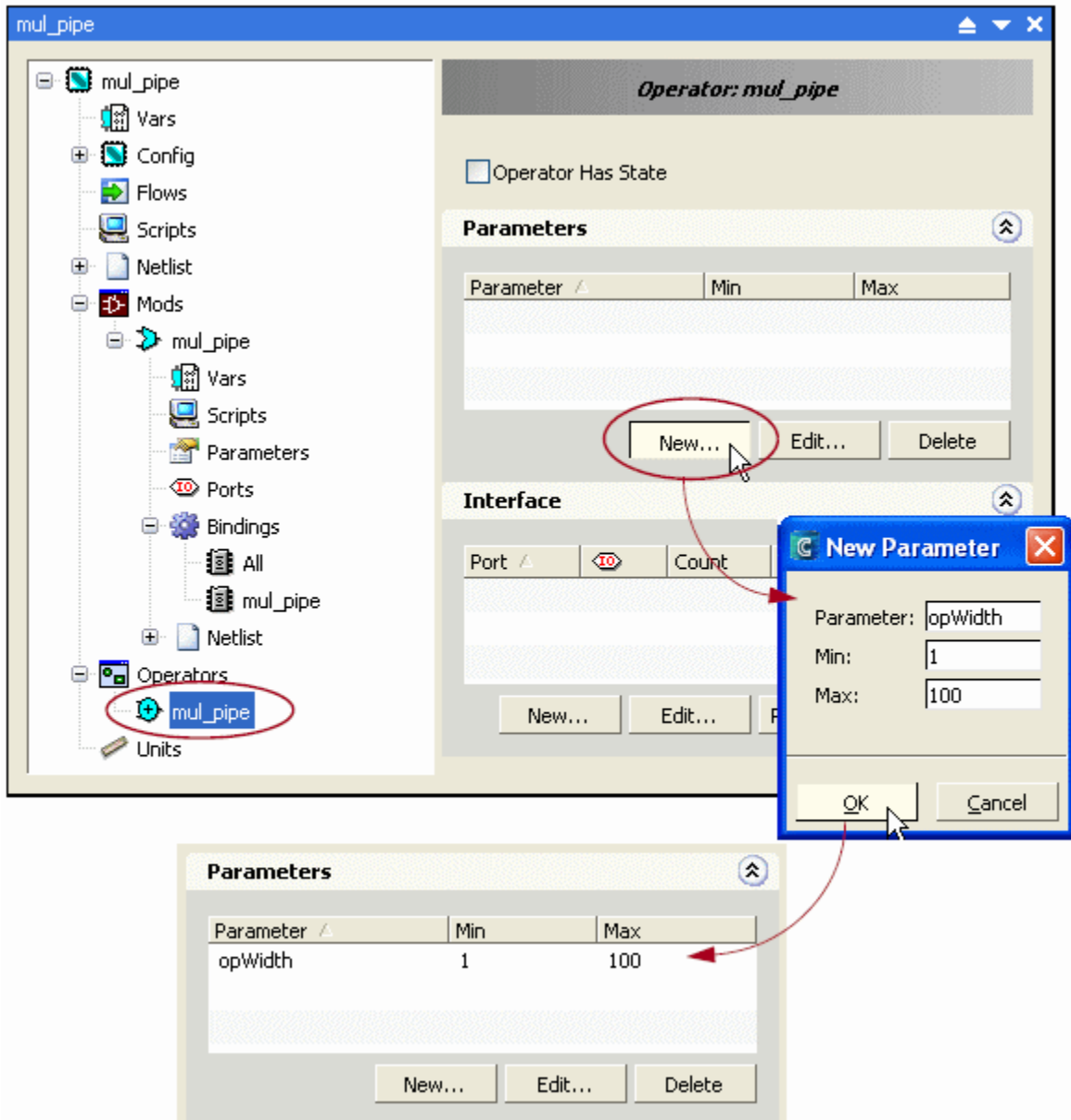
```

flow run /DesignCompiler/library add blank -libname mul_pipe -libtitle
mul_pipe -vendor LSI -technology lcbg11p -link_library lsi_lgbg11p_wc.db
-target_library lsi_lgbg11p_wc.db

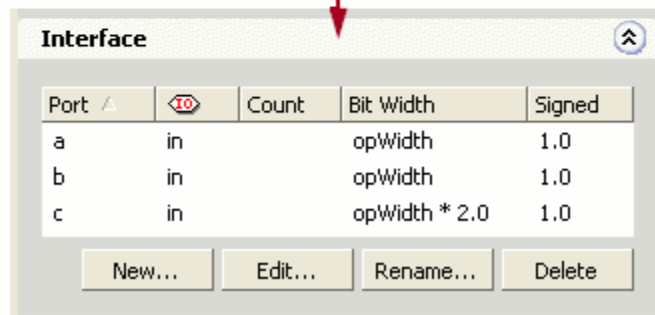
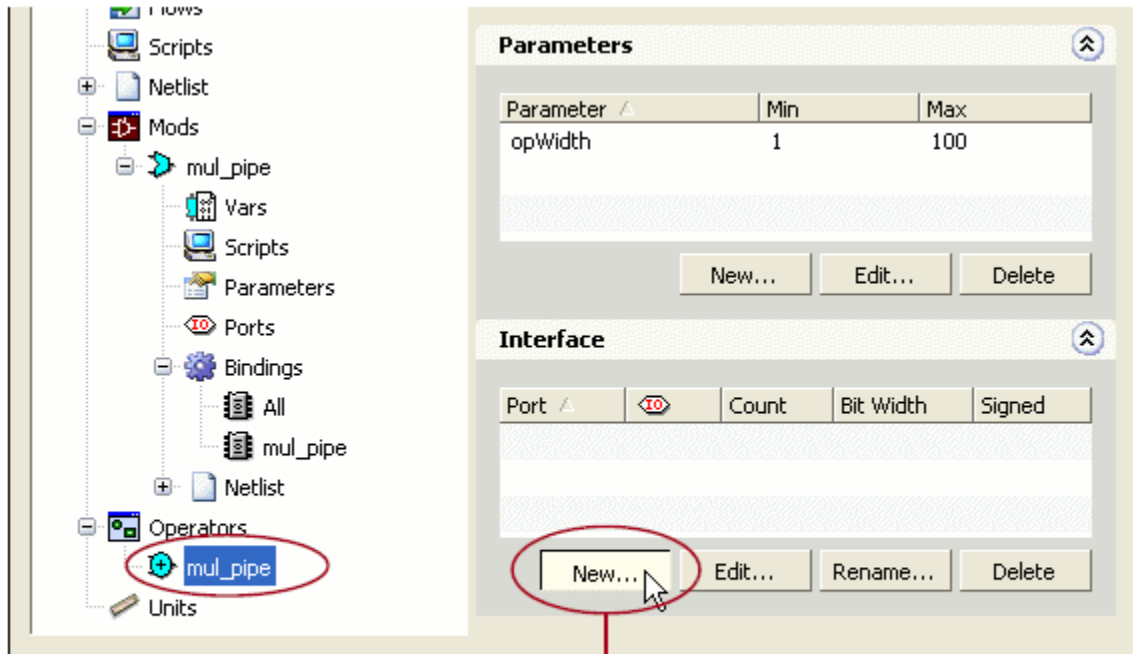
library import -module mul_pipe -vhdl -mod_type userop -libname mul_pipe
mul_pipe.vhd
    
```

After import the Module, bindings, and operator are created automatically, but the user must add the operator parameters and ports. Since the RTL port widths are parameterizable, we want to make the operator port widths parameterizable as well. Looking at the RTL we can see that

the input and output ports are all parameterized based on a single generic. So we need to create a parameter for the operator:

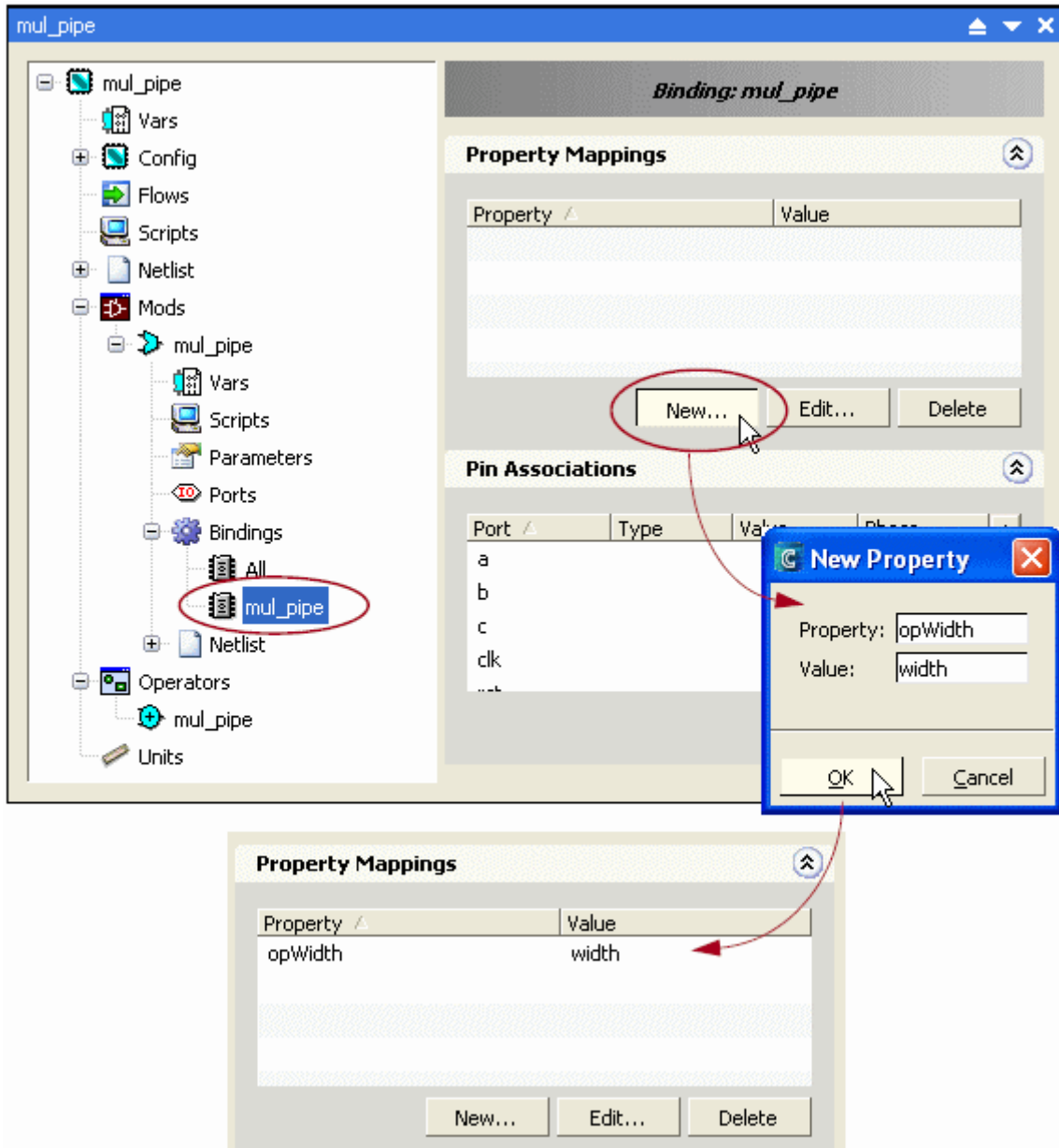


We then create the operator ports. The operator ports will correspond to the “data” ports on the RTL. In this example the RTL data ports are inputs “a” and “b” and output “c”.



After adding the operator ports, we can now add all of the pin associations on the operator binding as shown in the previous section.

Since the operator port widths are parameterized, we need to add a property mapping from the operator parameter (“opWidth”) to the module parameter (“width”):

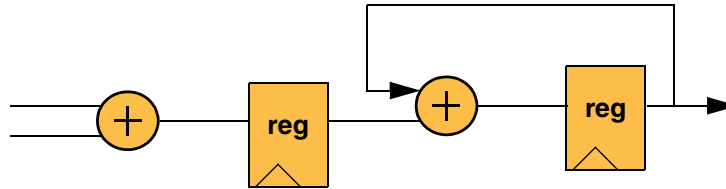


Creating Custom Operators with State

Operators with state are required when the operator has storage, such as multiply and accumulate, RAM with byte enable. Operators with state require a unique operator ID for each unique instance of the operator so that the operator is not shared. The operator ID is set as a parameter on the module and as a template parameter on the C++ function. Template functions are used to create unique instances of the C++ function.

These are required since a C++ function with state (e.g.) static variables is not unique unless its template parameter is unique.

Figure 4-10. Operator with State (MAC)



The RTL for the MAC shown above is:

```

ENTITY MAC IS
  GENERIC(width: natural range 1 to 100:= 2;
          ph_arst : natural := 1);
  PORT (
    clk      : IN STD_LOGIC ;
    rst      : IN STD_LOGIC ;
    a : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
    b : IN STD_LOGIC_VECTOR(width-1 DOWNTO 0) ;
    c : OUT STD_LOGIC_VECTOR(width*2-1 DOWNTO 0)
  ) ;
END MAC ;

ARCHITECTURE rtl OF MAC IS
  SIGNAL prod : STD_LOGIC_VECTOR(width*2-1 DOWNTO 0) ;
  SIGNAL acc  : STD_LOGIC_VECTOR(width*2-1 DOWNTO 0) ;
  BEGIN

  PROCESS (clk)
  BEGIN
    IF active(rst,ph_arst)then
      prod <= (others => '0');
      acc <= (others => '0');
    ELSIF(clk'EVENT AND clk = '1') THEN
      prod <= a * b;
      acc <= acc + prod;
    END IF ;
  END PROCESS;
  c <= acc;
END rtl;

```

Creating the library and importing the netlist for custom operators with state:

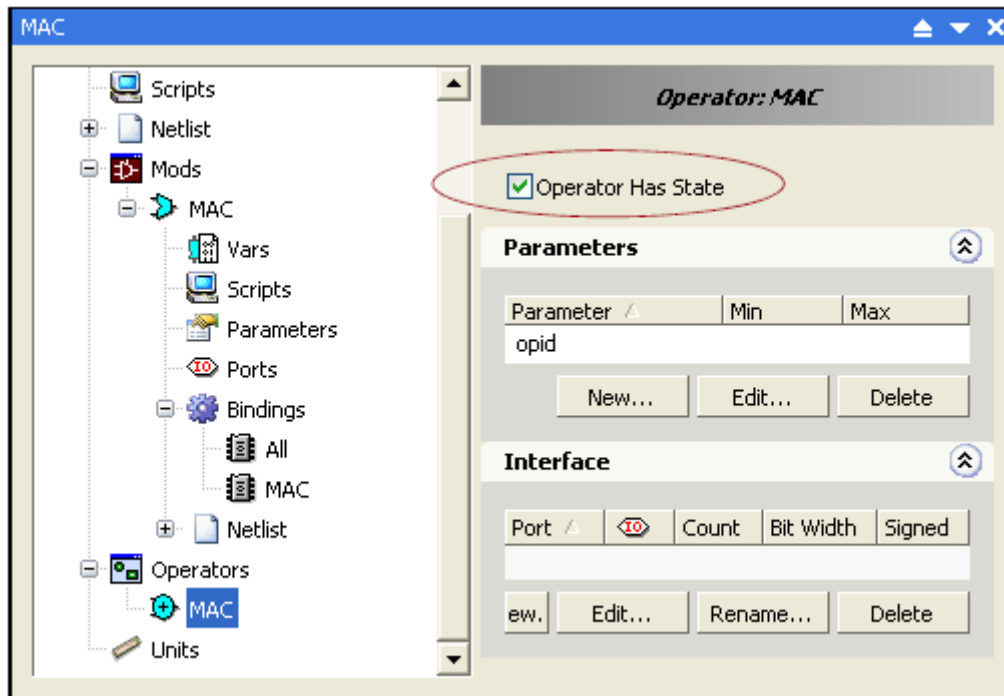
```

#Create the Library
flow run /DesignCompiler/library add blank \
  -libname MAC -libtitle MAC -vendor LSI -technology lcbg11p \
  -link_library lsi_lgbg11p_wc.db -target_library lsi_lgbg11p_wc.db

#Import the Netlist
library import -module MAC -vhdl -mod_type userop_withstate \
  -libname MAC MAC.vhd

```

Once the import has finished the module and operator must be edited in the same fashion describe for custom operators without state. In other words the operator parameters and ports must be added and the module binding pin associations and property mappings must be added. When the module and operator are created during library import a parameter called “opid” is automatically added to the module and the operator. The “Operator has State” bit is also set automatically.



Once the library is created a C++ template function must also be created that matches the functionality of the custom operator RTL. The custom C++ function must have a template parameter called “opid”. This must also be the first parameter listed if there are multiple template parameters.

```
typedef ac_int<18> inType;
typedef ac_int<36> outType;
#pragma map_to_operator "MAC"
template<int opid>
outType MAC(inType a, inType b){
    static outType acc;
    acc += a*b;
    return acc;
}
#pragma design top
void MAC_test(inType a, inType b, inType c, inType d, outType &e, outType &f){
    e = MAC<1>(a,b);
    f = MAC<2>(c,d);
}
```

As seen above, the `opid` template parameter can be used to create unique instances of the C++ function, and hence unique instances of the custom operator will be used when synthesizing the design.

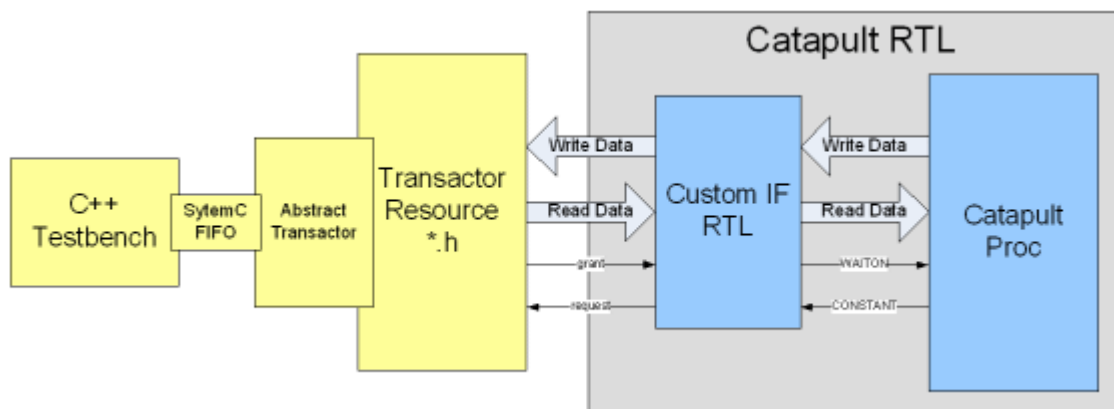
Verifying the Custom Operator RTL and Custom C++ Function

The custom operator RTL and corresponding C++ function must be verified against one another. They **MUST** be functionally equivalent for this flow to work properly. The Catapult SCVerify automated verification flow can be used to verify that the custom operator RTL and C++ function are functionally equivalent. To do this a C++ test bench is required to test the top-level design which instantiates the custom operator.

Using Custom Interfaces with SCVerify

Custom interfaces will not automatically work with the SCVerify automated verification flow. A SystemC transactor must be created by the user which allows the custom RTL module to be connected to the C++ verification environment. The transactor consists of two parts:

1. The abstract transactor class that pulls C++ typed values from the input FIFOs and performs type conversion from the C++ type into the bit-vector representation. There can be many transactors in the design - one for each formal argument in the original C++ function interface
2. The transactor resource which has the physical connection ports that are hooked up by the SCVerify generated wrappers.



Catapult provides transactors for inputs, outputs and bi-directional arguments. Catapult also provides implementations of transactor resources for each of the standard interface synthesis components such as `mgc_in_wire_en` and `mgc_out_stdreg_wait`. Because this flow requires manual creation of the SystemC transactor resources, the built-in transactor resource files can be used as a starting point for creating custom transactor resources. These can be found in

`$MGC_HOME/pkgs/siflibs`. There are also a number of custom interface examples in the Catapult toolkits that can be used as a starting point for creating other custom interfaces.

Transactor resource variables

In addition to creating the library, module, adding property mappings and pin associations, the user must add specific variables to the library module so that the SCVerify verification flow knows where the transactor resource is and how to hook it up. The following three variables are required:

- **scverify_trans_rsc_class**

Specifies the name of the transaction resource class

- **scverify_trans_rsc_tmpl**

Specifies the list of transactor resource class template parameters. These typically match the generics on the RTL module. The template parameter “streamcnt” is required for wire type interfaces. It must be the first parameter in the parameter list. This parameter tells the transactor resource file if the interface has been steamed. `Streamcnt == 1` indicates that there is no streaming. `streamcnt > 1` indicates the number of elements to be transferred.

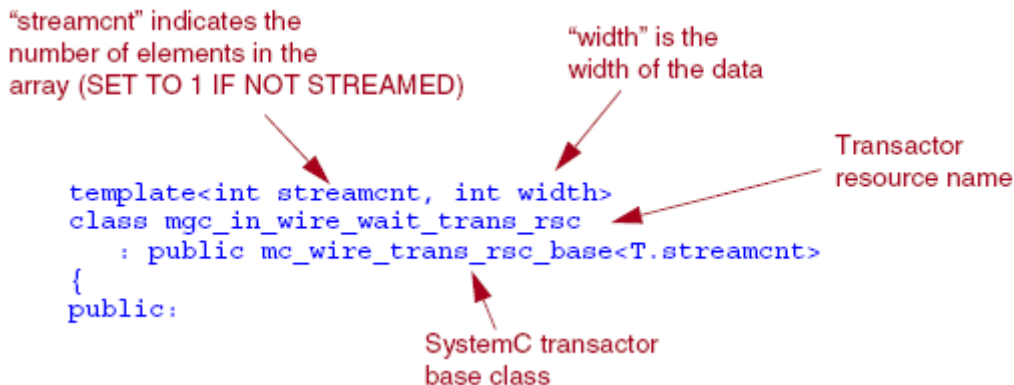
- **scverify_trans_rsc_hdr**

Specifies the name and location of the transactor resource header file

The built-in interfaces and transactor resource files can be used as examples for creating user defined interfaces and transactor resources. We can look at the `mgc_inwire_wait` component to understand the basic structure of the transactor resource file.

Figure 4-11 shows the `mgc_in_wire_wait` transactor resource class. The class is templated with the template parameters “streamcnt” and “width”. The transactor resource instantiates the base class transactor.

Figure 4-11. mgc_in_wire_wait Transactor Resource



Looking in the body of the transactor resource, in Figure 4-12, we see that the signals and ports must correspond to the ports on the RTL. Note that a clock must be defined in the transactor resource even if the RTL block does not contain a clock. The `mgc_in_wire_wait` transactor resource has sensitivity to the clock, `_lz` enable signal, and can control the `_vz` wait signal.

Figure 4-12. `mgc_in_wire_wait` Transactor Resource Body

```

sc_in<bool>          clk;
sc_inout<data_type> z;
sc_in<sc_dt::sc_logic> lz;
sc_out<sc_dt::sc_logic> vz;

SC_HAS_PROCESS(mgc_in_wire_wait_trans_rsc);
mgc_in_wire_wait_trans_rsc(const sc_module_name& name, bool phase)
    : base(name, phase)
    , clk("clk")
    , z("z")
    , lz("lz")
    , vz("vz")
{
    MC_METHOD(at_active_clk);
    this->sensitive << (phase ? this->clk.pos() : this->clk.neg());
    this->dont_initialize();

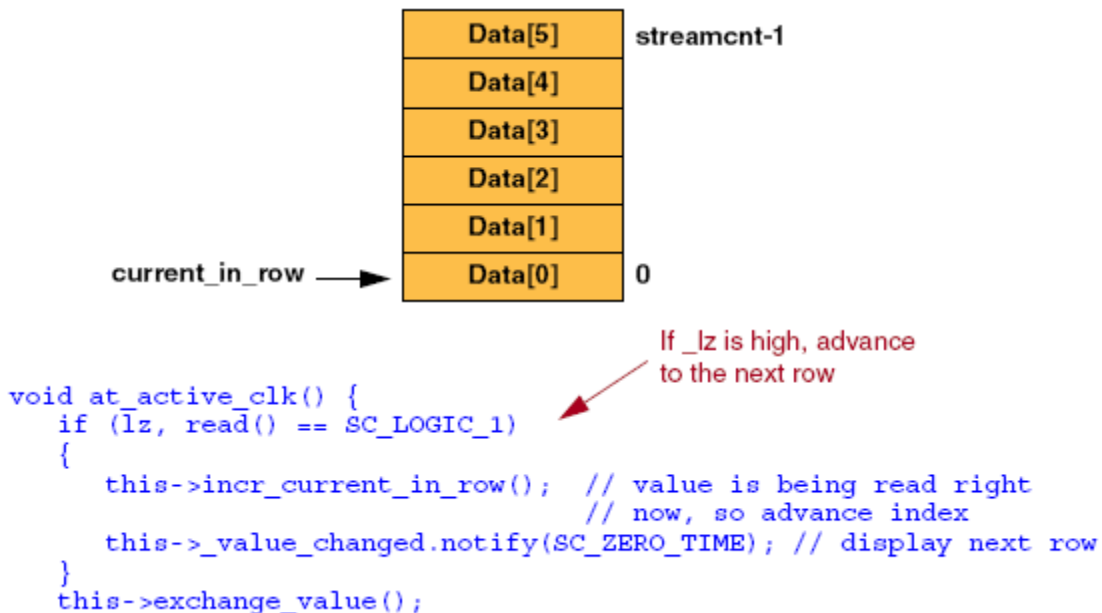
    SC_METHOD(update_z);
    this->sensitive << lz << this->_value_changed;
    this->dont_initialize();

    SC_METHOD(drive_v_signals);
    this->sensitive << (phase ? this->clk.pos() : this->clk.neg());
    this->sensitive << this->_wait_cycle_changed;
}

```

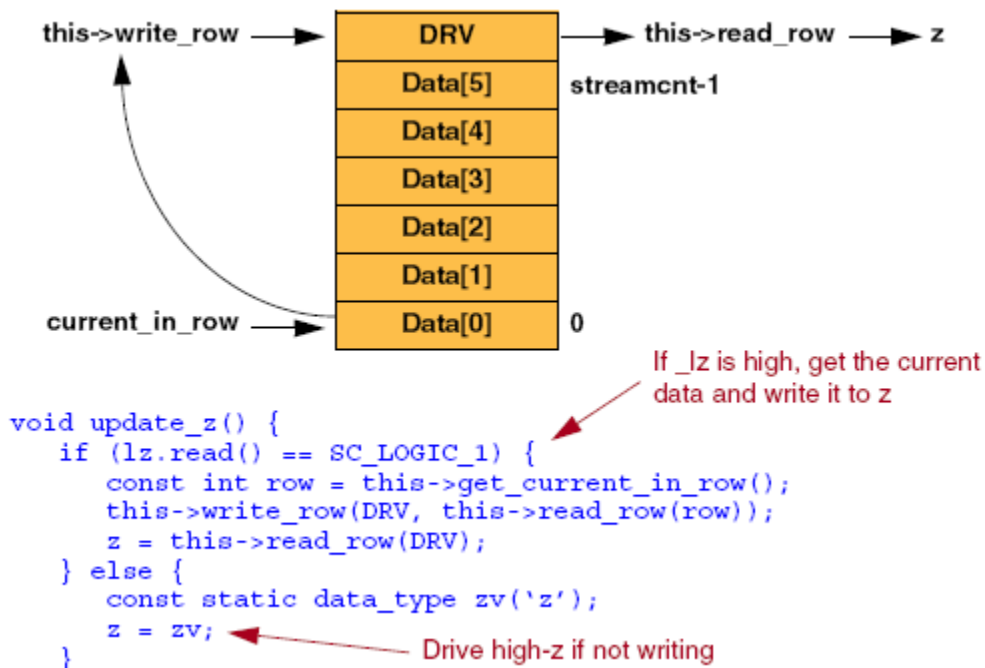
The testbench stimulus is pushed onto a FIFO for each interface every time the function is called. The number of rows of data in the FIFO is based on whether the interface has been streamed. For non-streamed interfaces (`streamcnt==1`) there is a single row. The `at_active_clock` method shown below shows that the row ptr into the FIFO is advanced when `_lz` (enable) is sampled high.

Figure 4-13. mgc_in_wire_wait Transactor Resource at_active_clk Method



Next the update_z method shown below illustrates how the transactor resource drives data to the RTL block. This method tests to see if read data is being requested (e.g. `_lz == 1`). When `_lz` is sampled high the current row of data in the FIFO is read and then written into temporary storage (DRV). This data is then driven out to the RTL block.

Figure 4-14. mgc_in_wire_wait Transactor Resource update_z Method



Lastly the `drive_v_signals` method is used to control the `_vz` (wait) signal. For the built-in interfaces `_vz` is usually set to high. However this is where the user can customize the behavior to delay the handshake.

Figure 4-15. `mgc_in_wire_wait` Transactor Resource `drive_v_signals` Method

```

void drive_v_signals() {
    if (this->_wait_cycles_cntr == 0) {
        vz = SC_LOGIC_1;
    } else {
        vz = SC_LOGIC_0;
        if (this->_wait_cycles_cntr != -1)
            this->_wait_cycles_cntr--;
    }
}

```

WAITON signal driven
based on counter

An example of creating the transactor resource files is shown below. It is based on the FSL interface example described earlier in this document. This interface example has the same behavior as the `mgc_in_wire_wait` component. The transactor resource header file was simply copied from `$MGC_HOME/pkgs/siflibs` and then edited.

Step 1 - Create the library component

Creating the FSL interface library component was covered in the earlier sections. The entity declaration is shown below:

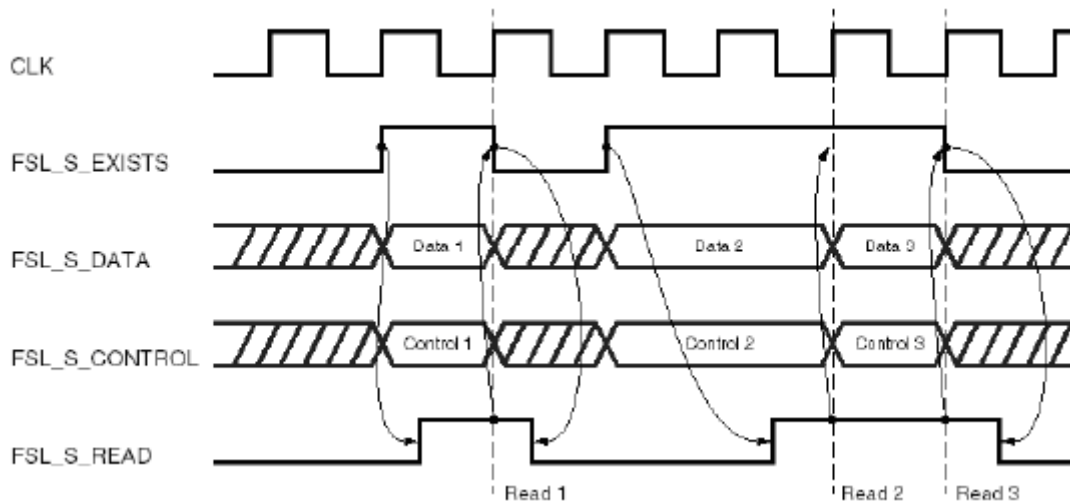
```

entity fsl_slave is
    generic (
        C_DWIDTH      : integer := 32
    );
    port (
        -- Slave FSL Signals
        FSL_S_Read      : out std_logic;
        FSL_S_Data      : in  std_logic_vector(0 to C_DWIDTH-1);
        FSL_S_Exists    : in  std_logic;
        --Catapult side signals
        data_out        : out std_logic_vector(C_DWIDTH- 1 downto 0);
        fsl_rdy         : out std_logic;
        fsl_rd          : in  std_logic
    );
end fsl_slave;

```

The FSL protocol has the following timing:

Figure 4-16. FSL Timing



Step2 - Copy the `mgc_in_wire_wait` transactor resource

The `mgc_in_wire_wait` transactor resource is in the `mgc_ioport_trans.h` file located in `$MGC_HOME/pkgs/siflibs`.

Step3 - Modify the transactor resource class

The `mgc_in_wire_wait` transactor resource file was copied and edited. First the transactor resource class name was changed and the template parameters were defined:

```
template<int streamcnt
        //EDITED BY USER
        //ADD REQUIRED TEMPLATE PARAMETERS
        , int C_DWIDTH>

class //EDITED BY USER
    //ADD TRANSACTOR RESOURCE NAME
    fsl_slave_trans_rsc
    : public mc_wire_trans_rsc_base<T,streamcnt>
    {
```

Step 4 - Modify the signals and constructor ports.

Next the transactor resource signals and constructor port names were changed to match the FSL RTL ports. Also a clock signal was added which is required:

```
//EDITED BY USER
//REPLACE WITH RTL PORT NAMES
sc_in<bool>          clk;
sc_in< sc_logic  >   arst;
sc_in< sc_logic  >   srst;
port_type           FSL_S_Data;
sc_in<sc_dt::sc_logic> FSL_S_Read;
sc_out<sc_dt::sc_logic> FSL_S_Exists;
```

```

SC_HAS_PROCESS(fsl_slave_trans_rsc);
fsl_slave_trans_rsc(const sc_module_name& name, bool phase)
: base(name, phase)
, clk("clk")
, FSL_S_Data(" FSL_S_Data")
, FSL_S_Read("FSL_S_Read")
, FSL_S_Exists("FSL_S_Exists")

```

Step 5 - Modify the update_z sensitivity

The update method was then modified to make it sensitive to the FSL_S_Read signal which is the equivalent to the mgc_in_wire_wait_lz signal:

```

//-----
//EDITED BY USER
//MAKE METHOD SENSATIVE TO ANY READ/WRITE STROBES FROM RTL
//CHANGE lz to RTL PORT NAME
SC_METHOD(update_z);
this->sensitive << FSL_S_Read << this->_value_changed;
this->dont_initialize();

```

Step6 - Modify the at_active_clk function

The at_active_clk function was changed to make it test FSL_S_Read:

```

void at_active_clk() {
//EDITED BY USER
//ADVANCE TO NEXT DATA WHEN READ CONDITION FROM RTL IS TRUE
//CHANGE lz to RTL PORT NAME
if (FSL_S_Read.read() == SC_LOGIC_1) {
    this->incr_current_in_row(); // value is being read right now,
                               // so advance index
}
}

```

Step7 - Modify the update_z function

The update_z function was edited to test the FSL_S_Read strobe to drive the testbench data:

```

void update_z() {
//EDITED BY USER
//WRITE READ DATA TO RTL PORT WHEN READ CONDITION IS TRUE
//CHANGE lz to RTL PORT NAME
if (FSL_S_Read.read() == SC_LOGIC_1) {
    const int row = this->get_current_in_row();
    this->write_row(DRV, this->read_row(row));
    if (this->is_combinational())
        this->initiate_driving_value_adjustments(row, row, COLS - 1, 0);
    //CHANGE z to RTL PORT NAME
    FSL_S_Data = this->read_row(DRV);
} else {
    const static data_type zv('Z');
    //CHANGE z to RTL PORT NAME
    FSL_S_Data = zv;
}
}

```

```
}
```

Step8- Modify the drv_v_signals function

The drive_v_signals function was edited to drive the FSL_S_Exists port when valid data is available:

```
void drive_v_signals()
{
    //EDITED BY USER
    //SET WAITON SIGNAL TRUE
    //CHANGE vz to RTL PORT NAME
    if (this->_wait_cycles_cntr == 0) {
        FSL_S_Exists = SC_LOGIC_1;
    } else {
        //CHANGE vz to RTL PORT NAME
        FSL_S_Exists = SC_LOGIC_0;
        if (this->_wait_cycles_cntr != -1) this->_wait_cycles_cntr--;
    }
}
```

Step9 - Edit the library to add the transactor resource variables

```
library add /LIBS/fsl_slave/MODS/fsl_slave/VARS/scverify_trans_rsc_class
--
-VALUE fsl_slave_trans_rsc
library add /LIBS/fsl_slave/MODS/fsl_slave/VARS/scverify_trans_rsc_tmpl -
-
-VALUE "streamcnt C_DWIDTH"
library add /LIBS/fsl_slave/MODS/fsl_slave/VARS/scverify_trans_rsc_hdr --
-VALUE {$LIBPATH/fsl_slave_trans_rsc.h}
```

Library Builder provides a shell command line interface that allows you to enter commands interactively and use scripts for batch processing. For more information, see the following topics:

- [“General Command Syntax”](#) on page 135.
- [“Using Tcl Commands in Scripts”](#) on page 143.
- [“Command Reference”](#) on page 145

General Command Syntax

The command line interface is based on the Tcl language and accepts all standard Tcl commands. Standard Tcl provides the foundation for the command syntax, including variable assignment, handling of lists and arrays, sorting, string manipulation, arithmetic operations, (if/case/foreach/while) statements, and procedures. [Table 5-1](#) describes the basic Tcl syntax rules.

Table 5-1. Basic Tcl Syntax

Syntax	Description
command arg1 ... argN	A command string consists of a command name followed by zero or more arguments. White space delimits each element.
<code>\$my_variable</code>	The dollar sign (\$) substitutes the value of a variable. In this example, the variable name is “my_variable”.
<code>[library get]</code>	Square brackets are used to execute a nested command. For example, if you want to pass the result of one command as an argument to another, use this syntax. In this example, the nested command is “ library get ”, which is used to obtain information about a currently loaded library.
<code>“some stuff”</code>	Double quotation marks group words as a single argument to a command. Dollar signs and square brackets are interpreted inside double quotation marks.
<code>{some stuff}</code>	Curly braces also group words into a single argument, but elements within the braces are not interpreted.

Table 5-1. Basic Tcl Syntax (cont.)

Syntax	Description
\	The backslash (\) is used to quote special characters. For example, \n generates a newline. The backslash also is used to “turn off” the special meanings of the dollar sign, quotation marks, square brackets, and curly braces.

Note



The following reference help files about the Tcl language are available in the Catapult software tree:

UNIX man pages: \$MGC_HOME/pkgs/tcl_msg/man

Windows help files: \$MGC_HOME/pkgs/tcl_msg/doc

Catapult commands typically have the form “<object> <operator>”, where <object> is the command name and <operator> is the action to be performed. The operator is first argument. Some examples are:

Table 5-2.

Object	Operators
library	add, get, set, rename, ...
flow	get, run

Some commands can operate on a set of objects. In these cases, the initial argument(s) specify the target object. For example:

Table 5-3.

Object	Operators
flow	get, run
flow package	provide, require, present, forget, option, names, versions, script, vcompare, vsatisfies
flow package option	add, get, remove, set

Documentation Conventions for Catapult Commands

For clarity when referring to Catapult commands by name, the documentation uses a composite name consisting of the object and operator components of the command string. For example the documentation might refer to the “[library add](#)” command or the “[flow package option add](#)” command. Similarly, the command reference pages use the composite form of the command names.

Command Reference Page Format

Each command reference page begins on a new page and is organized into the following sections:

- Command name and a short description of the what the command does.
- Syntax section that shows the proper usage of all of the command arguments and switches. For example:

```
library get ?<path>? ?<switches>? ?<args>?
                                Get info from the library database
<path>                          Hierarchical database path (Optional)
<switches>                      Valid switches: (Optional)
--                               End <switches> parsing
-recurse <string>               Everything under
-return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                                Return data format
    value                       just matching values
    path                        just matching paths
    pathvalue                   path and value combination for array set
    leaf                        just matching leaves
    leafvalue                   leaf and value combination for array set
    advanced                    hierarchical list structure
    none                        no return value
-checkpath <bool>              Error on path not found
-match <exact|glob>           Path match type
    exact                      exact paths only
    glob                       glob paths
-info <bool>                   Return object info
<args>                          Database subpath and value combinations
```

[Table 5-4](#) describes the meaning of the special characters used to express command line syntax.

Table 5-4. Documentation Conventions for Command Syntax

Symbol	Description
< >	Fields to be completed with your values.
?< >?	Optional argument.
	The “or” symbol. Indicates mutually exclusive arguments.

- Arguments section that describes each command argument. Note that there is a group of command switches that are common to many Catapult commands. Those switches are documented separately in the section [“Common Command Switches”](#) on page 141, rather than repeating the information throughout the command reference pages. A link to that section is provided from each command reference page that uses those switches.
- Description section that describes the purpose and usage of the command.
- Examples section that provides one or more examples to illustrate how the command is used.

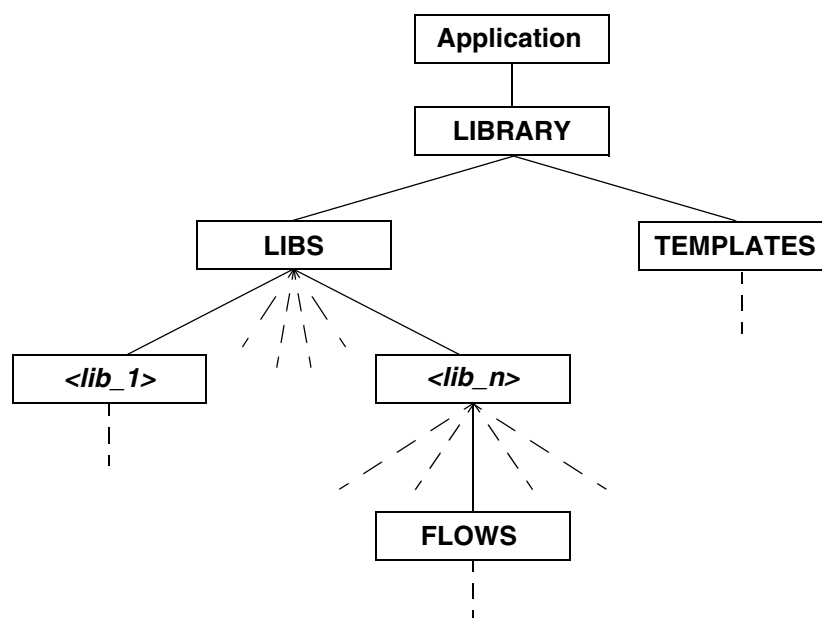
- Related Commands section that provides links to other command reference pages related to the current command.

Command Interface to the SIF Database

Every Catapult session has a SIF (synthesis internal format) database that stores the state of the application and the loaded libraries. Data values are stored as *key* and *value* pairs, and the keys are organized hierarchically. Data values are accessed by specifying the database path to their corresponding keys. In general, key names and hierarchical node names are in all uppercase letters. One notable exception is the key name *name* that appears in many places throughout the database.

The diagram in Figure 5-1 is a simplified representation of the SIF database hierarchy. It shows only the primary nodes of interest with respect to the set of Catapult C Library Builder commands that interface with the database. Those commands are listed in Table 5-5 on page 139.

Figure 5-1. Hierarchy of Objects in the SIF Database



The SIF database structure is not static. When a new Catapult session is started, the database is populated with the minimum set of key/value pairs (default settings) required by the system. Catapult dynamically creates, removes and modifies key/value pairs throughout the session.

Path and Sub-Path Argument Syntax Rules

The Catapult commands listed in Table 5-5 interact with the SIF database. These commands accept database path and sub-path arguments in order to access specific keys in the database.

Many of the switches described in the section “Common Command Switches” on page 141 can be used to control and refine how the path and sub-path arguments are evaluated by Catapult.

Table 5-5. Commands That Take Database Path Arguments

Command Name	Operators	Path Root
application	get	/ (Root of SIF)
flow	get	/FLOWS
flow package option	add, get, set, remove	/FLOWS
library	add, get, set, rename, remove	/LIBRARY

The root of the path argument is the node in the SIF hierarchy corresponding to the command name. In other words, the path argument for the “application” command is rooted at the “Application” node in the hierarchy. Similarly, the path for the “library” command is rooted at the “Application/LIBRARY” node.

Note



The set of key/value pairs in the database changes dynamically throughout the session as you work on the design. Use a “get” command to check for the existence of a key/value before attempting to modify or remove it.

Using Wildcard Characters in Path Arguments

Wildcard characters can be used in <path> and <sub-path> arguments. Wildcard expansion is enabled by the “-match glob” switch (see also “-match” on page 142). The following wildcard reserved characters are supported:

- `*` : Asterisk matches one and only one level.

Example:

```
library get /LIBS/*/name -match glob
#
# STDOPS mgc_ioport mgc_hierarchy my_base_asic_lib
```

- `...` : Ellipsis matches all sub-levels.

Example:

```
library get /LIBS/mgc_lcbg11p_beh_dc/.../VERSION -match glob
                                                -return pathvalue
```

In this example, the ellipsis wildcard expands the search to include the entire sub-tree below the /LIBS/mgc_lcbg11p_beh_dc node. The return values are found at the following paths:

```
/LIBS/mgc_lcbg11p_beh_dc/VERSION 2006a.72
```

```
/LIBS/mgc_lcbg11p_beh_dc/FLOWS/DesignCompiler/VERSION {}
```

The ellipsis is not valid at the root level. For example “/.../INTERFACE” is invalid.

Using Sub-Path Arguments

Commands that take database path arguments also accept optional sub-path arguments. Sub-paths make it possible to specify multiple variations of the path argument in a single command. The general form of the command line is shown below.

```
command_name <sif_path> ?<switches>? <sub-path_1> ... <sub-path_n>
```

The <sif_path> argument is a partial database path that is common to all of the target objects. A set of unique paths are formed by appending each sub-path to the partial path. In the following example the sub-paths “name” and “VERSION” are appended to the partial path “/FLOWPKGS/ModelSim” to form two complete paths. The “-return pathvalue” switch is used in this example so that the resulting paths are displayed in the return value.

```
library get /LIBS/mgc_lcbg11p_beh_dc -return pathvalue VERSION name
# /LIBS/mgc_lcbg11p_beh_dc/name mgc_lcbg11p_beh_dc
  /LIBS/mgc_lcbg11p_beh_dc/VERSION 2006a.72
```

Error Messages Caused by Invalid Paths

If an invalid path or sub-path is specified, the command is not executed and an error message is displayed for each invalid path. For example, the following command specifies two sub-paths, but one is invalid. The command exits and returns a path error for the invalid one.

```
library get /LIBS/mgc_lcbg11p_beh_dc -return pathvalue VERSION X_name
# Error: library get: Unknown path '/LIBS/mgc_lcbg11p_beh_dc/X_name'
```

The error can be suppressed by including the “-checkpath false” switch in the command line. In this case, the valid file name is returned.

```
library get /LIBS/mgc_lcbg11p_beh_dc -return pathvalue -checkpath false
                                                                VERSION X_name
# /LIBS/mgc_lcbg11p_beh_dc/VERSION 2006a.72
```

You can use the tcl catch command to catch the return value and then parse the results to determine which parts of the command passed. For example:

```
if ([catch {library get /LIBS/mgc_lcbg11p_beh_dc -return pathvalue VERSION
X_name} msg]) {
    // parse $msg for errors
    ...
}
```

Common Command Switches

The command switches described in this section are common to most of the Catapult commands. The switches qualify how the database is searched and how the returned data is displayed. The same command switch may be set several times on the same line. If the same switch is used, the last switch on the line will take precedence.

The following switches are described in this section:

```
-return
-checkpath
-match
-info
-recurse
-- (switch)
-help
--help
```

-return

The `-return` switch filters and formats the data returned by the command.

```
-return <value|path|pathvalue|leaf|leafvalue|advanced|none>
```

Arguments:

- **value**
Return only the data values stored at the database paths that match the search paths. For example:

```
library get /LIBS/mgc_lcb*/MODS/*and/name -match glob -return value
# mgc_and mgc_nand
```

- **path**
Return only the database paths that match the search paths. For example:

```
library get /LIBS/mgc_lcb*/MODS/*and/name -match glob -return path
# /LIBS/mgc_lcbg11p_beh_dc/MODS/mgc_and/name
/LIBS/mgc_lcbg11p_beh_dc/MODS/mgc_nand/name
```

- **pathvalue**
Return the database paths and the data values. For example:

```
library get /LIBS/mgc_lcb*/MODS/*and/name -match glob
                                           -return pathvalue
# /LIBS/mgc_lcbg11p_beh_dc/MODS/mgc_and/name mgc_and
/LIBS/mgc_lcbg11p_beh_dc/MODS/mgc_nand/name mgc_nand
```

- **leaf**
Return only the leaf names of the database paths that match the search paths. For example:

```
library get /LIBS/mgc_lcb*/MODS/*and/name -match glob -return leaf
```

```
# name name
```

- **leafvalue**

Return the leaf names and the data values. For example:

```
library get /LIBS/mgc_lcb*/MODS/*and/name -match glob
                                           -return leafvalue
# name mgc_and name mgc_nand
```

- **advanced**

Return a hierarchical representation of the database paths and the data values. Each level of hierarchy is enclosed in braces, and sub-levels are nested. For example:

```
library get /LIBS/mgc_lcb*/MODS/*and/name -match glob
                                           -return advanced
# LIBS {mgc_lcbg11p_beh_dc {MODS {mgc_and {name mgc_and} mgc_nand
{name mgc_nand}}}}
```

- **none**

Return nothing. Use this argument to suppress transcribing of the return value.

-checkpath

The `-checkpath` switch enables/disables error checking of paths.

```
-checkpath <true|false>
```

If the `-checkpath` switch is set to `true`, then the command will issue an error if any of the paths in the command line, including paths generated by `globs`, do not match an existing path.

-match

The `-match` switch can use `glob` or `exact` matching rules.

```
-match <glob|exact>
```

The following TCL rules are used with the `-match` switch. If the `-match` switch is set to `glob`, then the TCL `glob` rules are used to resolve the path argument. If set to `exact`, then the TCL `exact` rules are used to resolve the path argument. Refer to [“Using Wildcard Characters in Path Arguments”](#) on page 139 for more information about the `glob` option.

-info

If the `-info` switch is `true`, then the type information on each non-value node will be included in the return value if the return value includes the specified value.

```
-info <true|false>
```

-recurse

If the `-recurse` switch is true, then Catapult will perform a recursive path search.

```
-recurse <true|false|+<depth>>
```

Valid values for enabling full recursion are case insensitive and include: ‘1’, “yes”, or “true” (‘y’ and ‘t’ are sufficient). To disable it, use any other string or no string at all.

You can also limit the depth of recursion by specifying the number of levels to search. The argument format is “+<n>”, where the plus character is required and <n> is any whole number.

```
-recurse +3
```

Note that “+1” and “1” do not mean the same thing. Without the plus character, “1” evaluates to “true”.

-- (switch)

The `--` switch disables switch parsing for the rest of the line. You can set this switch so that you can access data objects that have names that conflict with reserved words (i.e. switches) and reserved characters.

-help

The `-help` switch displays help information about the commands.

--help

The `--help` switch displays recursive help for hierarchical commands.

Using Tcl Commands in Scripts

You can create a basic Tcl command script by first performing the desired command steps interactively in the GUI, and then use **File > Save Session Commands...** to capture the commands in a Tcl file. After you create the Tcl script, you can use one of the following methods to *source* the script from within a Library Builder session:

- [Interactive Command Line](#)
- [GUI](#)
- [Command Line Invocation Argument](#)
- [Tcl Startup Script](#)

Interactive Command Line

Type the following syntax to *source* your Tcl script:

```
source <my_tcl_script>
```

or type the following command to execute your Tcl script:

```
dofile {<path>/run1.tcl}
```

Note



Pathnames that Contain Spaces

Microsoft Windows operating systems allow pathnames to contain spaces. Therefore it is a good practice to enclose pathnames in curly braces ({ }) on Windows.

The `dofile` command is similar to the `source` command in that they both execute the Tcl commands contained in a specified file. An additional feature of the `dofile` command is that it also sends a message to the standard output device each time a Tcl command is executed. This is a useful feature that you can use to help debug the Tcl script.

All UNIX/Linux commands and many DOS commands are accessible from the Catapult command line shell as long as the command can be found in your search path (PATH environment variable). Another helpful Tcl feature is history tracking. Type the command “history” to view your previous commands. Any previous command can be re-executed typing “!*<history_number>*” or “!*<beginning_of_cmd>*”. You simply type “!!” to re-execute the last command.

GUI

On the menu bar select **File > Run Script**. Use the file navigator to locate, select and execute your script. Your script file runs in the Catapult Command/Transcript window.

Command Line Invocation Argument

When invoking Library Builder from a shell window, use the “-file” argument to source your script:

```
% catapult -product library_builder -shell -file <my_tcl_script>
```


Tcl Startup Script

Create a Tcl startup script named either “catapult.tcl” or “.catapult.tcl” and place it in one of the Catapult search directories identified below. During invocation, Library Builder reads and executes commands in the first startup file it finds. The search algorithm is as follows:

UNIX/Linux	Windows
1 \$HOME/catapult.tcl	1 %USERPROFILE%\catapult.tcl
2 \$HOME/.catapult.tcl	2 %USERPROFILE%\catapult.tcl
	3 %HOME%\catapult.tcl
	4 %HOME%\catapult.tcl
	5 c:\catapult.tcl
	6 c:\catapult.tcl

This is a good way to automatically define frequently used Tcl procedures. And because this file is in your home directory, you can update your Catapult software tree without overwriting this file.

Note



Adding project or design related commands (such as “flow package require”) to the startup script will cause errors because the script is parsed on invocation before any project or designs are loaded.

Command Reference

Table 5-6 summarizes the commands available in Library Builder and links to the associated command reference page.

Table 5-6. Alphabetical Command Summary

Command	Description
application get	Gets information from the Library Builder databases.
application exit	Closes the Library Builder session and returns an exit code.
application report	Generates a report of the Catapult licenses in use by Library Builder.
catapult -product library_builder	The shell-level command that invokes the Catapult C Library Builder tool.
dofile	Execute the specified Tcl script and print a message as each command in the script is executed.
flow get	Queries flow database information.
flow package names	Returns the names of all flow packages that are available to Catapult.

Table 5-6. Alphabetical Command Summary (cont.)

Command	Description
flow package option add	Creates a new option for a flow package.
flow package option get	Returns the value of the specified flow package option.
flow package option remove	Deletes the specified flow package option.
flow package option set	Sets the value of the specified option.
flow package provide	Registers a flow package name and compatible version(s).
flow package require	Loads the specified flow package.
flow package script	Returns the file system path to the flow package file.
flow package vcompare	Compares two version numbers and determines which one is newer.
flow package versions	Returns a list of all available versions of the specified flow package.
flow package vsatisfies	Compares two version strings and determines whether they are compatible.
flow run	Executes a flow procedure.
help command	Gets help on command syntax and usage.
help message	Gets help on system message codes.
library add	Creates new objects in the library database.
library characterize	Characterize a library or component.
library get	Get data from the library database.
library import	Import components into a library from HDL files.
library load	Load the specified library file(s).
library rename	Renames objects in the library database.
library remove	Remove objects from the library database.
library report	Generates a report about characterization data in the specified libraries or library element.
library save	Save one or more libraries to the file system.
library save_commands (Deprecated)	Saves the characterization command data of the specified library.
library set	Configure objects in the library database.
logfile	Controls the logfile options for the Library Builder session.
options defaults	Resets all flow options to default settings

Table 5-6. Alphabetical Command Summary (cont.)

Command	Description
<code>options exists</code>	Checks whether an option exists in the database.
<code>options get</code>	Returns the value of the specified option.
<code>options load</code>	Loads saved option settings.
<code>options save</code>	Save the in-memory options to the Catapult registry or an alternate file.
<code>options set</code>	Sets the value of an option in the database.
<code>quit</code>	Exits Library Builder.
<code>set_working_dir</code>	Set the working directory to the specified pathname.
<code>utility farm add</code>	Adds hosts to the library farm host database.
<code>utility farm get</code>	Returns information about a specified host or the host database in the Library Farm.
<code>utility farm release</code>	Releases an allocated host.
<code>utility farm remove</code>	Deletes hosts from the Library Farm host database.
<code>utility farm reserve</code>	Reserves a host from the host database.
<code>utility farm reset</code>	Resets hosts in the Library Farm host database to an idle state.
<code>utility farm set</code>	Configures the Library Farm host database and allocated hosts.

application get

Gets information from the Library Builder databases.

Syntax

```
application get ?<path>? ?<switches>? ?<args>?

<path>                Hierarchical database path (Optional)
<switches>            Valid switches: (Optional)
  --                  End <switches> parsing
  -recurse <string>   Everything under
  -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                       Return data format
    value             just matching values
    path              just matching paths
    pathvalue         path and value combination for array set
    leaf              just matching leaves
    leafvalue         leaf and value combination for array set
    advanced          hierarchical list structure
    none              no return value
  -checkpath <bool>   Error on path not found
  -match <exact|glob> Path match type
    exact             exact paths only
    glob              glob paths
  -info <bool>        Return object info
<args>                Database subpaths (Optional)
```

Arguments

- **<path>s**
Path into the Catapult SIF database. For detailed information about specifying database paths, refer to [“Command Interface to the SIF Database”](#) on page 138.
- **<switches>**
Use these switches control how the database is searched and how the returned data is displayed. These switches are common to all commands that take database path arguments. For detailed descriptions of the switches, refer to the section, [“Common Command Switches”](#) on page 141.
- **<args> s**
Sub-path(s) into the Catapult SIF database. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to [“Using Sub-Path Arguments”](#) on page 140.

Description

This command is used to get information from the library database or to get system information.

Examples

Example 1:

The following example queries the system database for the company name. The switch “-return pathvalue” causes the command to return both the database path that was queried and the value stored at that path.

```
application get /SYSTEM/COMPANY_NAME_VERBOSE -return pathvalue
```

The return value is:

```
# /SYSTEM/COMPANY_NAME_VERBOSE {Calypto Design Systems Corporation}
```

Example 2:

This example queries for all of the path nodes under the SYSTEM node. The “-match glob” switch is required in order to evaluate and expand the ‘*’ wildcard. The “-return” switch in this case is set to “path” so that only the database paths are returned and not the values stored there.

```
application get /SYSTEM/* -match glob -return path
```

The return value is:

```
# /SYSTEM/PRODUCTS /SYSTEM/COMPANY_NAME /SYSTEM/COMPANY_NAME_VERBOSE  
/SYSTEM/COMPANY_LOCATION /SYSTEM/DET_VERSION /SYSTEM/SHORT_VERSION  
/SYSTEM/BUILD_NUMBER /SYSTEM/RELEASE_DATE /SYSTEM/RELEASE_VERSION  
/SYSTEM/RELEASE_TYPE /SYSTEM/RELEASE_COPYRIGHT  
/SYSTEM/APPLICATION_SYN_PROJECT_EXT /SYSTEM/APPLICATION_LB_PROJECT_EXT  
/SYSTEM/EMAIL_SUPPORT /SYSTEM/EMAIL_LICENSE /SYSTEM/WEBPAGE  
/SYSTEM/DATE_CURRENT /SYSTEM/DATE_START /SYSTEM/PW_NAME /SYSTEM/PW_PASSWD  
/SYSTEM/PW_UID /SYSTEM/PW_GID /SYSTEM/PW_GECOS /SYSTEM/PW_DIR  
/SYSTEM/PW_SHELL /SYSTEM/UNAME_NODENAME /SYSTEM/UNAME_SYSNAME  
/SYSTEM/UNAME_VERSION /SYSTEM/UNAME_RELEASE /SYSTEM/UNAME_MACHINE  
/SYSTEM/SYS_PID /SYSTEM/ENV_HOME /SYSTEM/ENV_MGC_HOME  
/SYSTEM/ENV_MODEL_TECH /SYSTEM/ENV_STACK_SIZE  
/SYSTEM/ENV_LIMIT_LICENSE_PID /SYSTEM/ENV_VCO /SYSTEM/ENV_APPVAR  
/SYSTEM/ENV_APPHOME /SYSTEM/ENV_APP_INI /SYSTEM/ENV_KEEP_TMPS  
/SYSTEM/INI_DEBUG /SYSTEM/INI_LANGUAGE /SYSTEM/INI_CACHE_DIR  
/SYSTEM/INI_TCLSH_CMD /SYSTEM/ENV_TMPDIR /SYSTEM/APPLICATION_NAME  
/SYSTEM/APPLICATION_SHORT /SYSTEM/APPLICATION_TITLE  
/SYSTEM/APPLICATION_TITLE_SHORT /SYSTEM/APPLICATION_SUBTYPE  
/SYSTEM/APPLICATION_PROJECT_EXT /SYSTEM/APPLICATION_SPLASH  
/SYSTEM/APPLICATION_EXECUTABLE /SYSTEM/BANNER
```

Related Commands

[application exit](#)
[application report](#)

application exit

Closes the Library Builder session and returns an exit code.

Syntax

```
application exit ?<code>?
```

Arguments

- *<code>*

An integer value that specifies the status of the application upon exit. The set of valid values is platform dependent, but typically a '0' (zero) means a normal status and any other number represents an error status.

Description

This command immediately terminates the Library Builder session. Any unsaved work will be lost. The exit code, if specified, is passed to the operating system shell from with the application was invoked.

Example

The example below closes the Library Builder session and returns a status code of '2' to the operating system.

```
application exit 2
```

Related Commands

[application get](#)
[application report](#)

application report

Generates a report of the Catapult licenses in use by Library Builder.

Syntax

```
application report ?<options>?
```

```
<options>          Report options (Optional)
  -filename <string>  Write report to file
  -transcript <bool>  Send report to transcript
  -window <bool>      View report in a window
  -license <bool>     Include license in report
```

Arguments

- **-filename <string>**

Write the report to the specified file path. The path is relative to the current working directory for the Catapult session.

- **-transcript <bool>**

Send the report to the session transcript, which is also captured in the session log file. This is the default option when the application is running in “shell” mode.

- **-window <bool>**

Open the report in a document window in the Catapult session. This is the default option in the graphical user interface.

- **-license <bool>**

Include license information in the report.

Description

This command generates a license report and sends it to the specified output display or file.

Example

This example writes the report to file “LBreport.txt” in the current working directory.

```
application report -license true -filename LBreport.txt
```

Related Commands

[application get](#)
[application exit](#)

catapult -product library_builder

The shell-level command that invokes the Catapult C Library Builder tool.

Syntax

```
catapult -product library_builder ?<switches>? ?<project>?

<switches>          Valid switches: (Optional)
  -shell             Start the application in shell mode
  -product <library_builder|setup>
                    Start the specified product
                    Library Bulder [lb]
                    Setup Wizard [sw]
  -file <file>       Source the specified Tcl command file after invoking
  -logfile <file>    Create and open a log file at the specified pathname
  -version           Display the release version banner and exit
  -mglc_license_file <string>
                    Specify license file location
  -license <string> Specify license checkout preference (Multiple)
<project>          Project file to open (Optional)
```

Arguments

- **-shell**
Use shell command line user interface instead of the graphical user interface.
- **-product <library_builder | setup>**
The “library_builder” argument launches the Library Builder application. The more concise form “-library_builder” can be used instead of “-product library_builder.”

The “setup” argument launches the Catapult Setup Wizard that is used for configuring the Catapult software installation. For more information, refer to “[Using the Catapult Setup Wizard](#)” in the *Catapult C Synthesis Installation Guide*.
- **-file <Tcl_script_pathname>**
Source the specified Tcl command file after invoking.
- **-logfile <file>**
Create and open a log file at the specified path.
- **-version**
Display the release version banner and exit.
- **-mglc_license_file <string>**
Specify the path to a license file containing the Catapult C Library Builder licenses. This switch overrides the MGLS_LICENSE_FILE environment variable. Refer to the “[Licensing Mentor Graphics Software](#)” for information about the MGLS_LICENSE_FILE variable.
- **-license <string>**
This switch gives priority to the specified Catapult composite license when Catapult requests a license from the license server. If the specified license is not available, but a

different valid license is available, the alternate will be checked out and Catapult will display a warning message stating that an alternate license is being used.

Use multiple instances of the `-license` switch to specify a prioritized list of target licenses. The order of priority is the order in which the switches appear on the command line. The first valid license found is checked out.

If the `-license` switch is not used, the first valid composite found in the license file will be checked out. For a line-by-line description of a license file, refer to the [FLEXnet Licensing End User Guide](#).

From within a Catapult session, you can use the following command to see a list of the license features Catapult currently has checked out:

```
application report -license true
```

- **<project>**

Optionally specify a Catapult library file (`.lib`) to be loaded at invocation.

Description

This command invokes Catapult C Library Builder from a Shell command line. For Windows users, the option switches can be added to the “Target” field of the Windows Shortcut.

Examples

Example 1

This example opens the Library Builder session window and loads the specified library:

```
catapult -product library_builder /my_libs/ram_register-file.lib
```

Example 2

This example opens the Library Builder in shell mode and then “sources” the specified Tcl script.

```
catapult -product library_builder -shell
                                     -file /my_libs/do_ram_register_lib.tcl
```

dofile

Execute the specified Tcl script and print a message as each command in the script is executed.

Syntax

```
dofile ?options? <filename>
  -verbose  -- Transcript commands and return values
  -quiet    -- Do not transcript commands or return values
  -noerrors -- Continue processing commands even if error occurs
```

Arguments

- **<filename>**

This required argument specifies the pathname of a Tcl file to be executed. If only the leaf name is specified, Catapult looks in the current working directory for the file.

- **-verbose**

During execution of the Tcl file, display commands being executed and their return values in the transcript. Enabled by default.

- **-quiet**

Do not display commands or return values in the transcript.

- **-noerrors**

Continue processing commands even if an error occurs.

Description

The `dofile` command is an extension of the Tcl source command. In addition to executing a Tcl file, the `dofile` command sends a message to the standard output device as each command executes. This is very helpful when debugging a Tcl script.

Example

```
dofile run1.tcl
```

flow get

Queries flow database information.

Syntax

```

flow get ?<path>? ?<switches>? ?<args>?

<path>                Hierarchical database path (Optional)
<switches>            Valid switches: (Optional)
  --                  End <switches> parsing
  -recurse <string>   Everything under
  -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                        Return data format
    value             just matching values
    path              just matching paths
    pathvalue         path and value combination for array set
    leaf              just matching leaves
    leafvalue         leaf and value combination for array set
    advanced          hierarchical list structure
    none              no return value
  -checkpath <bool>   Error on path not found
  -match <exact|glob> Path match type
    exact             exact paths only
    glob              glob paths
  -info <bool>        Return object info
<args>                Database subpaths (Optional)

```

Arguments

- **<path>**
A hierarchical path into the SIF database. The root node of <path> is /FLOWS. For more information, refer to [“Command Interface to the SIF Database”](#) on page 138.
- **Common Switches**
Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section, [“Common Command Switches”](#) on page 141.
- **<args>**
One or more arguments specifying database sub-paths to flows. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to [“Using Sub-Path Arguments”](#) on page 140.

Description

The “flow get” command queries the Catapult database for information about currently loaded flow packages. Refer to the section [“Flow Customization”](#) in the *Catapult C Synthesis User’s and Reference Manual* for information about how flows operate in Catapult and how to create custom flow packages.

Examples

Example 1:

This example returns the SIF database paths for all of the flows currently loaded:

```
flow get /* -match glob -return pathvalue
```

The return value will be similar to the following:

```
# /Scanner {} /DesignCompiler {} /Precision {}
```

Example 2:

This example gets the version number and option settings of the Precision flow package. The <path> argument supplies a partial path that is common to both of the target data paths. The sub-path arguments “-VERSION” and “-OPTIONS” provided the remaining portions of the paths. The “-recurse” switch is enabled in order to traverse the OPTIONS sub-tree.

```
flow get /Precision -recurse true -return pathvalue -VERSION -OPTIONS
```

The return value shown below has been reformatted for clarity:

```
# /Precision/VERSION 2006a.101
/Precision/OPTIONS {
  FOLDERNAME {
    name FOLDERNAME
    type string
    DESCRIPTION {Output File Folder Name}
    DEFAULT Precision VALUE Precision}
}
```

Related Commands

flow package names	flow package require
flow package option add	flow package script
flow package option get	flow package vcompare
flow package option remove	flow package versions
flow package option set	flow package vsatisfies
flow package provide	flow run

flow package names

Returns the names of all flow packages that are available to Catapult.

Syntax

```
flow package names
```

Arguments

None

Description

The “flow package names” command returns a list of the flow packages in the flow index. When initializing a new project, Catapult scans all of the flow files (.flo) in the search path and constructs an index of the flow packages it finds. The flows are not loaded at this time, but the index enables Catapult to load them dynamically when they are required. Refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual* for information about how flows operate in Catapult and how to create custom flow packages.

Example

```
flow package names
```

Returns a list similar to the following:

```
# ArtisanMemories DesignCompiler GIDEL HDL_Tcl Make ModelSim NCSim Netlist  
OSCI PowerPlay PowerTheater Precision Report SCVerify SLEC Scanner  
Schedule Schematic XPower
```

Related Commands

flow get	flow package require
flow package option add	flow package script
flow package option get	flow package vcompare
flow package option remove	flow package versions
flow package option set	flow package vsatisfies
flow package provide	flow run

flow package option add

Creates a new option for a flow package.

Syntax

```
flow package option add <path> ?<switches>?
```

<path>	Hierarchical database path (Required)
<switches>	Valid switches: (Optional)
-default <value>	default value of option
-description <string>	description of option
-type <type>	option type

Arguments

- **<path>**

Hierarchical database path of the new option in the SIF database. The path is rooted at the FLOWS node in the database and has the form /<package_name>/<option_name>. Only the <option_name> portion is required when this command is called from within a flow package script. When called from the command line, the whole path is required, including the leading forward slash.

For more information about the SIF database, refer to [“Command Interface to the SIF Database”](#) on page 138.

- **-default <value>**

Initializes the new option to the specified default value. If the “-default” switch is omitted, no default value is assigned and the option is not initialized.

- **-description <string>**

Prompt string that appears next to the input field on the options dialog page. If the “-description” switch is omitted, the default description text is the value of the <path> argument.

- **-type <value>**

Defines the data type of the new option. Valid types are “string,” “bool,” “integer,” “double,” “inputfile,” “outputfile,” and “directory.”

Description

The “flow package option add” command creates a new option in the database for the specified flow package. The new option is also added to the Flows page of the options dialog (**Tools > Set Options... > Flows > package_name**). If this command is implemented in a flow script, it is only executed during the *flow-indexing* process, which is initiated by calling [flow package require](#). Refer to the section [“Flow Customization”](#) in the *Catapult C Synthesis User’s and Reference Manual* for information about how flows operate in Catapult and how to create custom flow packages.

Example

The example below creates and initializes a new option named `ReportFile` in the flow package `MyFlowPackage`. The option is initialized to `my_flow.rpt`, its default value. In the GUI, the new option appears on the options page for the specified flow package, and the description text appears next to the `ReportFile` option field.

```
flow package option add /MyFlowPackage/ReportFile -default "my_flow.rpt"
                    -description "Output Report Filename"
```

Related Commands

flow get	flow package require
flow package names	flow package script
flow package option get	flow package vcompare
flow package option remove	flow package versions
flow package option set	flow package vsatisfies
flow package provide	flow run

flow package option get

Returns the value of the specified flow package option.

Syntax

```
flow package option get <path> ?-nocomplain?

<path>           Hierarchical database path (Required)
<switches>       Valid switches: (Optional)
  -nocomplain     Do not error if package or option is not defined
```

Arguments

- **<path>**

A pseudo path into the SIF database that has the form `/<package_name>/<option_name>`. The pseudo path is a shorthand form of the true database path to flow option values: `/<package_name>/OPTIONS/<option_name>/VALUE`.

The path is rooted at the FLOWS node in the database. The leading forward slash is required.

- ***-nocomplain***

Disable error reporting. If the command encounters an error, no error message is reported and the command does not terminate.

Description

The “flow package option get” command returns the current value of the specified flow package option. This command is the preferred method of accessing flow option settings because it takes a simplified form of the `<path>` argument, as compared to the “flow get” command.

An error status is returned if the option value is not defined or if `<path>` is not valid.

Example

The example below returns the value of the option `FOLDERNAME` in package `Precision`. Error reporting is disabled by the `-nocomplain` switch. The return value is “Precision”.

```
flow package option get /Precision/FOLDERNAME -nocomplain
# Precision
```

Related Commands

flow get	flow package require
flow package names	flow package script
flow package option add	flow package vcompare
flow package option remove	flow package versions
flow package option set	flow package vsatisfies
flow package provide	flow run

flow package option remove

Deletes the specified flow package option.

Syntax

```
flow package option remove <path> ?-nocomplain?

<path>           Hierarchical database path (Required)
<switches>      Valid switches: (Optional)
    -nocomplain  Do not error if package or option is not defined
```

Arguments

- **<path>**

A pseudo path into the SIF database that has the form `/<package_name>/<option_name>`. The pseudo path is a shorthand form of the true database path to flow option values: `/<package_name>/OPTIONS/<option_name>/VALUE`.

The path is rooted at the FLOWS node in the database. The leading forward slash is required.

- **-nocomplain**

Disable error reporting. If the command returns an error, no error message is reported and the script does not terminate.

Description

The “flow package option remove” command deletes the option from the SIF database. However, it is not deleted from the **Flows** options dialog in the GUI.

This command can be used in flow package scripts only if it is in global context (not inside a procedure), and it is valid only in the *flow-indexing* safe interpreter (Refer to the section “[Safe Interpreters](#)” in the *Catapult C Synthesis User’s and Reference Manual*). This command can also be called from the command line.

Example

The example below deletes the option `ReportFile` in package `MyflowPackage`. Error reporting is disabled by the `-nocomplain` switch.

```
flow package option remove /MyflowPackage/ReportFile -nocomplain
```

Related Commands

<ul style="list-style-type: none"> flow get flow package names flow package option add flow package option get flow package option set flow package provide 	<ul style="list-style-type: none"> flow package require flow package script flow package vcompare flow package versions flow package vsatisfies flow run
---	--

flow package option set

Sets the value of the specified option.

Syntax

```
flow package option set <path> <value>
```

<path>	Hierarchical database path (Required)
<value>	value of option (Required)

Arguments

- **<path>**

A pseudo path into the SIF database that has the form `/<package_name>/<option_name>`. The pseudo path is a shorthand form of the true database path to flow option values: `/<package_name>/OPTIONS/<option_name>/VALUE`.

The path is rooted at the FLOWS node in the database. The leading forward slash is required.

- **<value>**

String value assigned to the flow package option.

Description

The “flow package option set” command assigns a string value to the specified flow package option. This command can be used in flow package scripts only if it is in global context (not inside a procedure), and it is valid only in the *flow-indexing* safe interpreter (Refer to the section “[Safe Interpreters](#)” in the *Catapult C Synthesis User’s and Reference Manual*). This command can also be called from the command line.

Example

The example below assigns the value `project.rpt` to option `ReportFile` in package `MyflowPackage`.

```
flow package option set /MyflowPackage/ReportFile "project.rpt"
```

Related Commands

flow get	flow package require
flow package names	flow package script
flow package option add	flow package vcompare
flow package option get	flow package versions
flow package option remove	flow package vsatisfies
flow package provide	flow run

flow package provide

Registers a flow package name and compatible version(s).

Syntax

```
flow package provide <path> <package> ?<switches>?

<path>           path (Required)
<package>       package name (Required)
<switches>      Valid switches: (Optional)
  -require <package ?version ?exact??>
                require additional flow package
  -description <string>  flow package description
  -hidden <bool>         set hidden flag
```

Arguments

- **<path>**
 The file system path to a flow package file (*.flo). This argument is only required if the “flow package provide” command is called from the Catapult command line. Typically the command is called from within a flow package file. In that context this argument superfluous.
- **<package>**
 The name of the flow package. This is the name by which the package will be indexed and referenced in the Catapult system database.
- **-require <package ?version ?exact??>**
 Require an additional flow package. If the current flow package depends on another flow package, the option will “require” the other package if it is not already loaded. Optionally specify the version label for the dependent flow package. Refer to “[flow package require](#)” on page 165.

 For more information about flow package versioning, refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual*.
- **-description <string>**
 A string describing the flow package. This string is stored in the SIF database.
- **-hidden <bool>**
 Enable this switch to hide the flow in the Catapult GUI. Default value is ‘0’ (false).

Description

The “flow package provide” command registers a package name and version(s) in the Catapult system database. Each time a new project is created, Catapult locates all flow package files in the *Flows Search Path*, then generates an index of flow packages based on the “flow package provide” command in each file. For more information about how flow packages are loaded, refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual*.

Example

The example below adds the package `My_Reports` to the flow package index, and declares that it is compatible with scripts written for the listed versions.

```
flow package provide My_Reports -description "Generates custom reports"  
                                v2.4 v2.5 v3.0
```

Related Commands

flow get	flow package require
flow package names	flow package script
flow package option add	flow package vcompare
flow package option get	flow package versions
flow package option remove	flow package vsatisfies
flow package option set	flow run

flow package require

Loads the specified flow package.

Syntax

```
flow package require ?<switches>? <package> ?<version>?

<switches>      Valid switches: (Optional)
  -exact         versions must exactly match
  -source        loads the package source into the current package
<package>       package name (Required)
<version>       Version (Optional)
```

Arguments

- **<package>**
 Name of a flow package that exists in the Catapult index. Use the [flow package names](#) command to get a list of all indexed flow packages.
- **-exact**
 The “-exact” switch must be used in conjunction with the <version> argument. It forces Catapult to use the exact <version> specified. If the switch is omitted, Catapult will load the latest compatible version of the package. [flow package names](#)
- **-source**
 When the [flow package require](#) command is called from inside of a flow package file, the “-source” switch causes the required package to be loaded into the current package. Thus, flow from the required package and the current package will run in the same Tcl interpreter.
- **<version>**
 Specifies the version of the flow package to load. Use the [flow package versions](#) command to get a list of available versions for the package. If no version is specified, the latest version will be loaded. For more information about flow package versioning, refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual*.

Description

The “flow package require” command makes the flows in the named package available in the session. Packages must be *required* before their flows can be run. If the specified package is already available, no change is made.

A flow package can access flows from a another flow package in two ways:

1. `flow package require <package_name>`
`flow run <package_name>/<flow_name>`
2. `flow package require <package_name> -source <flow_name>`

In the first method, the required package remains external to the current package and you must use the flow run command to invoke its flows. Furthermore, its flows are run in a separate Tcl interpreter. In the second method, the required package is inlined into the current package, and

flow package require

flows from both packages share the same Tcl interpreter. Be aware that sharing the same interpreter can lead to errors caused by conflicting names of global variables or processes.

Flow code is not loaded into the tool, but is read from the file system each time it is run. For more information about how flow packages are loaded, refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual*. To get a list of available flow packages, use the [flow package names](#) command.

Example

The example below loads flow package `My_Report`, version “v3.0”.

```
flow package require -exact My_Reports v3.0
```

Related Commands

flow get	flow package provide
flow package names	flow package script
flow package option add	flow package vcompare
flow package option get	flow package versions
flow package option remove	flow package vsatisfies
flow package option set	flow run

flow package script

Returns the file system path to the flow package file.

Syntax

```
flow package script <package> <version>

<package>      package name (Required)
<version>     package version (Required)
```

Arguments

- **<package>**

The name of a flow package that is currently in the package index. To get a list of all indexed packages use [flow package names](#).

- **<version>**

Specifies the version of the flow package to query. Use the [flow package versions](#) command to get a list of available versions for the package. For more information about flow package versioning, refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual*.

Description

The “flow package script” command returns the full file system path to the flow package file for the specified package name.

Example

```
flow package script Precision 2006a.101
```

Returns the following path:

```
# /Catapult/mgc_home/pkgs/sif/userware/En_na/flows/app_psr.flo
```

Related Commands

flow get	flow package provide
flow package names	flow package require
flow package option add	flow package vcompare
flow package option get	flow package versions
flow package option remove	flow package vsatisfies
flow package option set	flow run

flow package vcompare

Compares two version numbers and determines which one is newer.

Syntax

```
flow package vcompare ?<switches>? <version1> <version2>

<switches>      Valid switches: (Optional)
                 -nocomplain  Do not error if either version is invalid
<version1>      version for comparison (Required)
<version2>      version for comparison (Required)
```

Arguments

- **<version1> and <version2>**
Flow package versions to be compared. Use the [flow package versions](#) command to get a list of available versions for the package.
- **-nocomplain**
Do not report an error if either <version> argument is invalid.

Description

The “flow package vcompare” command compares the ordinal value of <version1> to <version2> and returns a status as follows:

Table 5-7.

Status	Description
-1	<version1> is older (lower ordinal value)
0	<version1> and <version2> are equivalent
1	<version1> is newer (higher ordinal value)

For more information about flow package versioning, refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual*.

Example

The example below compares the version strings “2007a.12” and “2007a.15” and returns the status “-1” that indicates string 2 (2007a.15) is the more recent version.

```
flow package vcompare 2007a.12 2007a.15
# -1
```


Related Commands

flow get
flow package names
flow package option add
flow package option get
flow package option remove
flow package option set

flow package provide
flow package require
flow package script
flow package versions
flow package vsatisfies
flow run

flow package versions

Returns a list of all available versions of the specified flow package.

Syntax

```
flow package versions <package>  
  
<package>           package name (Required)
```

Arguments

- **<package>**
Required argument that specifies the name of a flow package currently in the package index. To get a list of all indexed packages use [flow package names](#).

Description

The “flow package versions” command returns the list of all versions of the specified flow package. For more information about flow package versioning, refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual*.

Example

The example below returns the list of versions for the Precision flow package

```
flow package version Precision  
# 2006a1.18 2006a.101 2005c.151 2005c.128 2005b.110 2005c.99 2005b.91  
2005c.79 2005a.69 2005a.56 2004a.75 2004b.67 2004c.45
```

Related Commands

flow get	flow package provide
flow package names	flow package require
flow package option add	flow package script
flow package option get	flow package vcompare
flow package option remove	flow package vsatisfies
flow package option set	flow run

flow package vsatisfies

Compares two version strings and determines whether they are compatible.

Syntax

```
flow package vsatisfies ?<switches>? <version1> <version2>

<switches>      Valid switches: (Optional)
                 -exact          versions must be equivalent
<version1>      version for comparison (Required)
<version2>      version for comparison (Required)
```

Arguments

- **-exact**
 Specifies that both versions must be equivalent.
- **<version1> and <version2>**
 Flow package versions to be compared. Use the [flow package versions](#) command to get a list of available versions for the package. For more information about flow package versioning, refer to the section “[Flow Customization](#)” in the *Catapult C Synthesis User’s and Reference Manual*.

Description

The “flow package vsatisfies” command compares the ordinal value of <version1> to <version2>. If <version1> is lower, it is not compatible with <version2> and the return status is ‘0’. If <version1> is greater than, or equal to <version2>, the versions are compatible and the return status is ‘1’.

Example

This example compares two versions of the Precision flow package to demonstrate their ordinal relationship and compatibility. First, version1 is compared to version2. Then the order is reversed and version2 is compared to version1.

```
flow package vsatisfies 2005c.151 2005c.128
# 1
flow package vsatisfies 2005c.128 2005c.151
# 0
```

In the first case, version “2005c.151” is compatible with version “2005c.128”. The second case, however, shows that the reverse is not true.

Related Commands

flow get	flow package provide
flow package names	flow package require
flow package option add	flow package script
flow package option get	flow package vcompare
flow package option remove	flow package versions
flow package option set	flow run

flow run

Executes a flow procedure.

Syntax

```
flow run <flow>

<flow>    flow name (Required)
```

Arguments

- **<flow>**

Hierarchical database path to the flow procedure. The path is rooted at the flo node in the database and has the form /<package_name>/<flow_name>. The leading forward slash is required. The /<package_name> portion can be omitted if <flow_name> is in the same package with the “flow run” command.

Description

The “flow run” command launches the specified flow procedure. This command is mainly used for running the “library add” flows provided with the Library Builder. For details about how to use this command with those flows, refer to [“Creating a New Library”](#) on page 51.

Example

The following command runs the “library add” flow in the “DesignCompiler” flow package. The flow arguments specify the library template type “base” (Base ASIC Library) and its required initial values. (Line breaks have been added to improve readability)

```
flow run /DesignCompiler/library add base \  
-libname my_lib \  
-libtitle my_lib \  
-vendor Sample \  
-technology 180nm \  
-link_library sample_180nm.db \  
-target_library sample_180nm.db
```

Related Commands

flow get	flow package provide
flow package names	flow package require
flow package option add	flow package script
flow package option get	flow package vcompare
flow package option remove	flow package versions
flow package option set	flow package vsatisfies

help command

Gets help on command syntax and usage.

Syntax

```
help command ?<args>?  
  
<args>      <pattern> | <command> <args> ... (Optional)
```

Arguments

- **<args>**

This argument specifies a Library Builder command name for which help information is returned. The “<pattern> | <command>” field represents the first word in the command name and can be expressed as a literal string or a wildcard glob expression using the asterisk (*) character. The “<args> ...” field represents any additional words in the command name. For information about Tcl command syntax with respect to Catapult command names, refer to [“General Command Syntax”](#) on page 135.

Description

The `help` command returns help on the syntax and usage of Library Builder commands. If no command name is specified, a list of command names is returned.

Note



Note, help about command usage is also accessible by invoking the command with the “-help” switch. For example, “library -help” or “library add -help”. The “--help” (double dash) option displays recursive help for hierarchical commands.

Examples

Example 1:

This example returns only the list of command names because `<args>` is omitted.

```
help command  
  
# User commands:  
# application  
# flow  
# help  
# library  
# logfile  
# options  
# set_working_dir  
# utility  
# view
```

Example 2:

This example returns the syntax for all of the “view” command operators (schedule, schematic, file and source):

```
# Usage: view <type>          View or edit design files
```

Commands

help command

```
# <type>                                Operation for <object> (Required)
# -----
# file <filepath> ?<switches>?
#                                     Any file type
# <filepath>                           Relative or absolute file path (Required)
# <switches>                             Valid switches: (Optional)
#   -filetype <string>
#                                     file type
#   -branch <string>
#                                     name
# -----
# library                               Edit library
```

Related Commands

[help message](#)

help message

Gets help on system message codes.

Syntax

```
help message <id> ?<switches>?
```

<id>	message identifier (Required)
<switches>	Valid switches: (Optional)
-description	return message long description
-hasdescription	return true if message has long description
-minseverity	return message minimum severity
-severity	return message severity

Arguments

- *<id>*

Each system message has a unique identifier associated with it, which is displayed in the transcript window at the end of the short description. The identifiers are composed of two elements, a text label that indicates type of operation that was being performed when the message was generated, and an integer to uniquely identify each message associated with a label. Some examples are:

```
ASSERT-1  
CIN-6  
LOOP-4  
PRJ-1
```

- *-description* | *-hasdescription* | *-minseverity* | *-severity*

One of these four optional switches can be used to specify the type of information that is returned. If none is specified, “-description” is the default.

Description

The “help message” command returns help information about Catapult system message codes (IDs). Catapult displays a brief form of system messages (error, warning and info) in the transcript window. At the end of the brief message is a message ID that, in many cases, corresponds to a longer, more detailed description of the message. The “-hasdescription” switch tells you whether or not a long description is available. Use the “-description” switch to get the long description. (In the GUI you can simply double-click on the ID to view the long description.)

The “-severity” and “-minseverity” return the present *severity* setting and the minimum severity setting of the message. The severity level (“error”, “Warning” or “Information”) of a message is user-configurable. For more information about Catapult system message IDs and severity classifications, refer to [“Set Messages Options”](#) on page 27.

Examples

Example 1:

This example first shows the how the short form of the “LOOP-4” message would appear in the transcript window. Following that, the “help message” command is used to get the long description.

Short form of message:

```
# fir_filter.cpp(26): Loop
'/fir_filter/fir_filter_proc/fir_filter_main/SHIFT' is left rolled.
(LOOP-4)
```

Get the long form of the message:

```
help message LOOP-4 -description
# The loop listed in this message is not being unrolled. This could be
because the UNROLL directive is set to "no" for this loop. Or it could be
because the UNROLL directive is set to "yes" but the number of iterations
is not known.
```

Example 2:

This example calls the “help message” command twice to the get severity information about the “LOOP-4” message.

Current severity:

```
help message ASSERT-1 -severity
# error
```

Minimum severity:

```
help message ASSERT-1 -minseverity
# warning
```

Related Commands

[help command](#)

library add

Creates new objects in the library database.

Syntax

```
library add ?<path>? ?<switches>? ?<args>?

<path>                Hierarchical database path (Optional)
<switches>            Valid switches: (Optional)
--                    End <switches> parsing
-return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                        Return data format
    value              just matching values
    path               just matching paths
    pathvalue          path and value combination for array set
    leaf               just matching leaves
    leafvalue          leaf and value combination for array set
    advanced           hierarchical list structure
    none               no return value
-checkpath <bool>     Error on path not found
-match <exact|glob>  Path match type
    exact              exact paths only
    glob               glob paths
-info <bool>          Return object info
-no_a_rst <value>    Suppress asynchronous RAM reset
-no_s_rst <value>    Suppress synchronous RAM reset
<args>                Database subpaths (Optional)
```

Arguments

- **<path>**
A hierarchical path into the SIF database. The root node of <path> is /LIBRARY. For more information, refer to [“Command Interface to the SIF Database”](#) on page 138.
- **Common Switches**
Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how returned data displays. For more information, see [“Common Command Switches”](#) on page 141.
- **-no_a_rst**
Creates a property on the binding to suppress the asynchronous reset on a RAM interface. The property value can be set to one of the following values:
 - 1** — unconditionally suppresses the reset.
 - Name of a module parameter (width, en_active)** — suppresses the reset when the value of the specified parameter is 1.

The RTL model must be modified before simulation.

- **-no_s_rst**

Creates a property on the binding to suppress the synchronous reset on a RAM interface. The property value can be set to one of the following values:

1 — unconditionally suppresses the reset.

Name of a module parameter (width, en_active) — suppresses the reset when the value of the specified parameter is 1.

The RTL model must be modified before simulation.

- **<args>**

Sub-path(s) into the Catapult SIF database. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to [“Using Sub-Path Arguments”](#) on page 140.

Description

Library add creates new objects in the library database.

Example

Example 1:

The following example uses the “library add” command to create a new variable (my_var) in a library (my_lib), and assigns it the value “hello”. The “library get” commands are included to show how the database is changed by the add command.

```
library get /LIBS/my_lib/VARS/my_var/* -match glob -return pathvalue
# Error: library get: Unknown path '/LIBS/my_lib/VARS/my_var'

library add /LIBS/my_lib/VARS/my_var -VALUE "hello"

library get /LIBS/my_lib/VARS/my_var/* -match glob -return pathvalue
# /LIBS/my_lib/VARS/my_var/name my_var
/LIBS/my_lib/VARS/my_var/type void
/LIBS/my_lib/VARS/my_var/VALUE hello
/LIBS/my_lib/VARS/my_var/REQUIRED 1
/LIBS/my_lib/VARS/my_var/DEFINED_VALUE 1
```

Example 2:

The following example removes the reset ports/parameters from an existing RAM library object and sets the necessary property on the object to tell Catapult that no reset is needed:

```
set lib ram_sample-065nm-dualport_beh_dc
set lp "/LIBS/$lib/MODS"
library load ${lib}.lib

#remove the synchronous reset port
library remove $lp/RAM_separateRW/PORTS/s_rst --
#Remove parameter that defines active phase of the synchronous reset
library remove $lp/RAM_separateRW/PARAMETERS/s_reset_active --
#Remove generic related to port
library remove $lp/RAM_separateRW/NETLIST/GENERIC/VALUE -- -4
```

```
#Set property on the RAM to tell Catapult a sync reset is not needed
library add $lp/RAM_separateRW/BINDINGS/all/PROPERTY_MAPPING -- -no_s_rst
1

#remove the asynchronous reset port
library remove $lp/RAM_separateRW/PORTS/a_rst --
#Remove parameter that defines active phase of the asynchronous reset
library remove $lp/RAM_separateRW/PARAMETERS/a_reset_active --
#Remove generic related to port
library remove $lp/RAM_separateRW/NETLIST/GENERIC/VALUE -- -4
#Set property on the RAM to tell Catapult an async reset is not needed
library add $lp/RAM_separateRW/BINDINGS/all/PROPERTY_MAPPING -- -no_a_rst
1

library save /LIBS/ram_sample-065nm-dualport_beh_dc -filename ram_sample-
065nm-dualport_beh_dc.lib
```

Related Commands

library edit	library rename
library characterize	library remove
library get	library report
library import	library save
library load	library save_commands (Deprecated)
	library set

library characterize

Characterize a library or component.

Syntax

```
library characterize ?<args>?  
#      <args>                path(s) (Optional)
```

Arguments

- *<args>*

Hierarchical path(s) into the SIF database. A path can specify an entire library, a module (MOD) or a qualified module (QMOD). The root node of <path> is /LIBRARY. For more information, refer to “[Command Interface to the SIF Database](#)” on page 138.

Description

This command is used to characterize one or more qualified library modules. Specifying a module will characterize all qualified modules it contains. Similarly, specifying the entire library will characterize all modules in the library.

Examples

Example 1:

This example characterizes the entire library.

```
library characterize /LIBS/my_lib
```

Example 2:

This example characterizes the all qualified modules under the “mgc_not” module.

```
library characterize /LIBS/my_lib/MODS/mgc_not
```

Example 3:

This example characterizes the qualified module “mgc_not (4)” under the “mgc_not” module.

```
library characterize /LIBS/my_lib/MODS/mgc_add/QMODS/mgc_not (4)
```

Related Commands

library add	library rename
library edit	library remove
library get	library report
library import	library save
library load	library save_commands (Deprecated)
	library set

library edit

Open a library in the Library Editor window.

Syntax

```
library edit <lib_path>
```

Arguments

- **<path>**

This required argument specifies the database path to a library that has been loaded. However, only the library name is required since all libraries share the same path (/LIBS/<lib_name>). For more information about the SIF database, refer to [“Command Interface to the SIF Database”](#) on page 138.

Description

This command opens the Library Editor window and loads the specified library.

Examples

The following example shows the two equivalent forms for specifying the target library.

```
library edit /LIBS/my_lib
```

```
library edit my_lib
```

Related Commands

[library add](#)
[library characterize](#)
[library get](#)
[library import](#)
[library load](#)

[library rename](#)
[library remove](#)
[library report](#)
[library save](#)
[library save_commands \(Deprecated\)](#)
[library set](#)

library get

Get data from the library database.

Syntax

```
library get ?<path>? ?<switches>? ?<args>?

<path>                Hierarchical database path (Optional)
<switches>            Valid switches: (Optional)
  --                  End <switches> parsing
  -recurse <string>   Everything under
  -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                        Return data format
    value             just matching values
    path              just matching paths
    pathvalue         path and value combination for array set
    leaf              just matching leaves
    leafvalue         leaf and value combination for array set
    advanced          hierarchical list structure
    none              no return value
  -checkpath <bool>   Error on path not found
  -match <exact|glob> Path match type
    exact             exact paths only
    glob              glob paths
  -info <bool>        Return object info
<args>                Database subpaths (Optional)
```

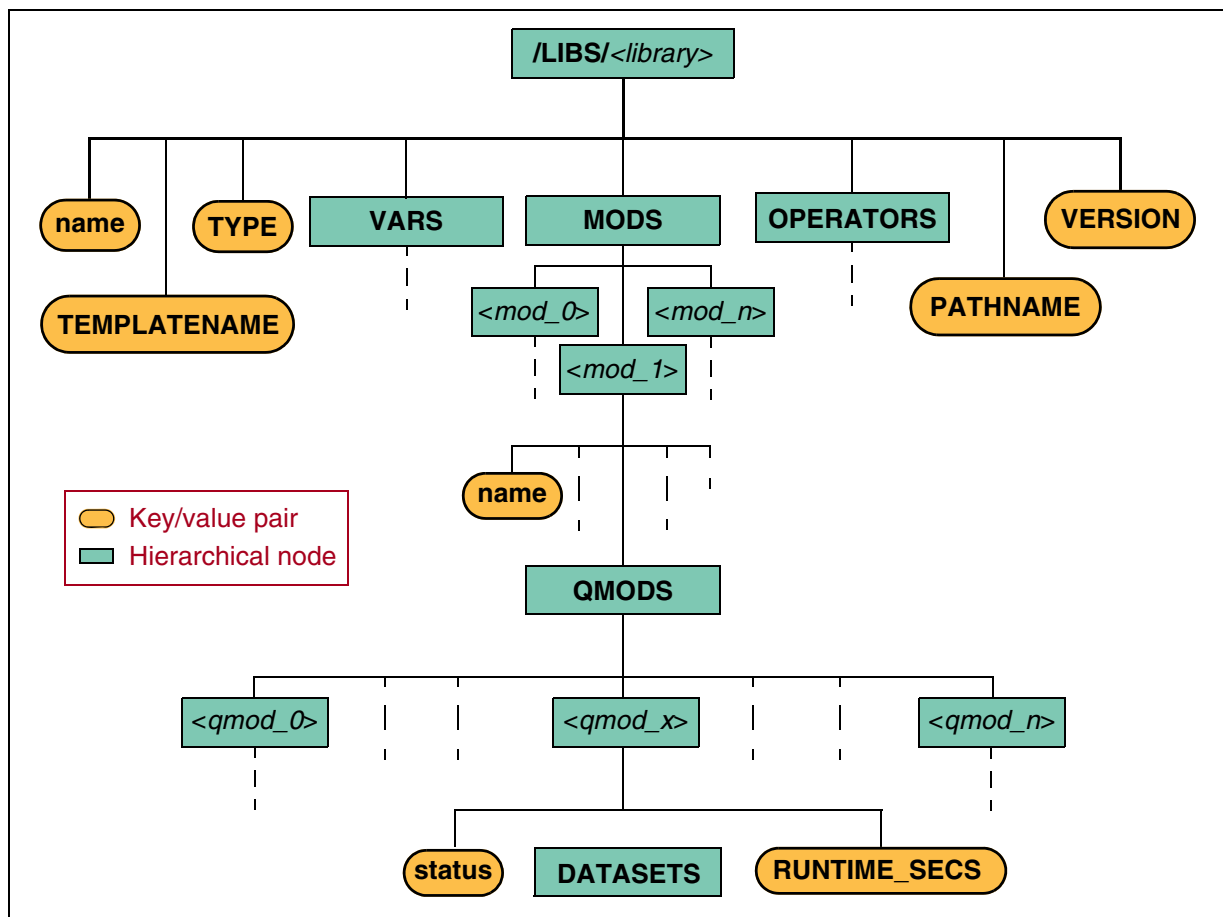
Arguments

- **<path>s**
A hierarchical path into the SIF database. The root node of <path> is /LIBRARY. For more information, refer to [“Command Interface to the SIF Database”](#) on page 138.
- **Common Switches**
Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section, [“Common Command Switches”](#) on page 141.
- **<args>**
Sub-path(s) into the Catapult SIF database. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to [“Using Sub-Path Arguments”](#) on page 140.

Description

This command queries the Catapult SIF database for information about the specified library. Data for each library is stored in a separate sub-tree in the database. All libraries in the database have the same general hierarchical structure. Figure 5-2 is a simplified representation of a library sub-tree. It shows only the nodes that may be of interest to the user.

Figure 5-2. Nodes of Interest in the Library Database Hierarchy



The actual set of nodes in any particular library is unique because of the dynamic nature of the SIF database. When a library is first created, the database is populated with the minimum set of nodes to represent the library. As you work on the library, Catapult adds/removes nodes accordingly.

Examples

Example 1:

The following example gets the names of all libraries in the database. The asterisk (*) wildcard in the <path> argument is used to match all nodes (libraries) under the LIBS node. The “-match glob” switch is required when wildcard expressions are used. The <path> resolves to the “name” key in each library. The “-return value” switch filters the query results to exclude all but the data values for the target key.

```

library get /LIBS/*/name -match glob -return value
# STDOPS mgc_ioport mgc_hierarchy my_asic_lib
  
```

Example 2:

This example get the characterization status for all modules whose names end with “or”. The <path> argument specifies the target module nodes, and the two sub-path arguments specify the

library get

target keys (“name” and “STATUS”) in each module. The “-return leafvalue” filters the results to include the leaf node of the path (database key in this case) and the value at that node.

```
library get /LIBS/my_asic_lib/MODS/*or -match glob -return leafvalue
-- -name -STATUS
# name mgc_or STATUS PENDING name mgc_nor STATUS PENDING name mgc_xor
STATUS PENDING name mgc_xnor STATUS PENDING
```

The command matched four modules, mgc_or, mgc_nor, mgc_xor and mgc_xnor. Each has “PENDING” status.

Example 3:

This example returns the “technology” and “vendor” settings for the library. The “-return pathvalue” switch shows you the fully resolved database path for each target node.

```
library get /LIBS/my_asic_lib/VARS -return pathvalue technology/VALUE
vendor/VALUE
# /LIBS/my_asic_lib/VARS/vendor/VALUE {LSI Logic}
/LIBS/my_asic_lib/VARS/technology/VALUE sample-130nm
```

Related Commands

[library add](#)
[library characterize](#)
[library edit](#)
[library import](#)
[library load](#)

[library rename](#)
[library remove](#)
[library report](#)
[library save](#)
[library save_commands \(Deprecated\)](#)
[library set](#)

library import

Import components into a library from HDL files.

Syntax

```
library import ?<switches>? ?<files>?
```

<switches> Valid switches: (Optional)

- module <string> Name of module to import
- operator <string> Name of operator to import
- vhdl Import VHDL netlist
- vhdl_libmap <name> <path>
VHDL library mapping (Required)
- property_map <propname> <propval>
Adds the property propname, propval to the All bindings for all imported mods (Required)
- port_default <portname> <default>
Adds the default value 'default' to the named port (Required)
- non_char_param <parameter>
Makes the named parameter an HDL generic only (not a characterization param)
- input_register <portname>
Adds the INPUT_REGISTER flag to the named port (must be an input port) (Required)
- char_range <param> <range_str>
Adds the named range to the CHAR_RANGE of the named parameter (Required)
- add_variable <varname,> <value>
Adds the named variable / value to the VARS the named parameter (Required)
- vhdl_option <string> Passes <string> as an option to the VHDL parser to specify the VHDL language version, either '-87' or '-93' (default). Example: -vhdl_option -87
- verilog Import Verilog netlist
- verilog_option <string>
Passes <string> as an option to the Verilog parser to specify the Verilog language version, either '-2001' or '-95' (default).
Example: -verilog_option -2001
- get_tops List top level modules
- libname <string> Name of library to import to
- mod_type <ram|rom|inport|outport|inoutport|userop|userop_withstate>
Module type

ram	ram
rom	rom
inport	inport
outport	outport
inoutport	inoutport
userop	userop
userop_withstate	userop_withstate

<files> files to import (Optional)

Description

The `library import` command creates a component library from an HDL/C++ input file. Some editing of the imported components may be necessary. The VHDL parser used by the `library import` command infers many pre-defined VHDL attributes to improve the mapping process and require less editing of the output library. For more information, see “[Pre-defined VHDL attributes](#)” on page 186.

Pre-defined VHDL attributes

This `library import` command can create fully defined library modules from properly written VHDL IP source files. This eliminates the need to interactively add properties to the new library after it is created.

The predefined VHDL attributes used by the `library import` command are described in the following sections:

- [BINDINGS Attribute](#)
- [PINASSOC Attribute](#)
- [PORT_DEFAULT Attribute](#)
- [PROP_MAP_Always and PROP_MAP_Area Attributes](#)
- [PROP_MAP_<property_name> Attributes](#)
- [CHAR_RANGE Attribute](#)
- [INPUT_REGISTER Attribute](#)
- [NON_CHAR_PARAM Attribute](#)
- [VAR_<variable_name> Attribute](#)

BINDINGS Attribute

This attribute marks the operator bindings used by the imported IP. The string is a comma-separated list of operator names and is applied to the ENTITY. The standard operator bindings are `read_port`, `write_port`, `read_ram`, `write_ram` and `read_rom`.

Examples:

```
attribute BINDINGS : string;  
  
attribute BINDINGS of i2s_input_wait_es : entity is "read_port";
```

PINASSOC Attribute

This attribute defines the PINASSOC types/values for the port bindings. It is applied to an ENTITY PORT. The string is a comma-separated list of specifications of the form:

```
"<oper_name>:<pinassoc_type>:<pinassoc_value>:<phase>"
```

where:

<oper_name>: Names one of the port bindings from the BINDINGS attribute. The asterisk character (“*”) can be used to create the PINASSOC for this port for all of the port bindings.

<pinassoc_type> : Can be one of the following values: SIGNAL, CONSTANT or WAITON.

<pinassoc_value>: Can be one of the following values: [CLOCK], [A_RST], [S_RST], [ENABLE], [EXTERNAL], [DIRECT] or [GLOBAL].

<phase>: Specifies whether the signal is active high (“1”) or active low (“0”). Can also be an expression using generics that resolves to 1 or 0.

Examples:

```
attribute PINASSOC : string;
attribute PINASSOC of SysClk : signal is "*:SIGNAL:[CLOCK]:";
attribute PINASSOC of a_rst : signal is "*:SIGNAL:[A_RST]:arst_active";
attribute PINASSOC of i2s_sclk : signal is "*:SIGNAL:[GLOBAL]:";
attribute PINASSOC of i2s_ws : signal is "*:SIGNAL:[GLOBAL]:";
attribute PINASSOC of sdin : signal is "*:SIGNAL:[EXTERNAL]:";
attribute PINASSOC of pdin : signal is "read_port:OPERATOR_PIN:D:";
attribute PINASSOC of pdin_req : signal is "*:CONSTANT:1:";
attribute PINASSOC of pdin_ack : signal is "*:WAITON::";
```

PORT_DEFAULT Attribute

This attribute defines the default value for Catapult to drive onto the port. The string is any expression. This attribute is applied to ENTITY PORTS.

Examples:

```
attribute PORT_DEFAULT : string;
attribute PORT_DEFAULT of a_rst : signal is "1 - arst_active";
attribute PORT_DEFAULT of pdin_req : signal is "0";
```

PROP_MAP_Always and PROP_MAP_Area Attributes

These attributes define the values for certain fixed PROPERTY_MAPPINGS that apply only to the "all" BINDING in the library. These attributes are applied to the ENTITY.

Examples:

```
attribute PROP_MAP_Always : string;
```

```
attribute PROP_MAP_Always of i2s_input_wait_es :  
    entity is "bits_per_frame >= 2^bits(width/2)";  
  
attribute PROP_MAP_Area : string;  
  
attribute PROP_MAP_Area of i2s_input_wait_es : entity is "interpolate(1)";
```

PROP_MAP_<property_name> Attributes

These attributes define the values for certain fixed PROPERTY_MAPPINGS, (such as “Delay”, “SeqDelay”, “InitDelay”, “Count”) that can be applied to the specified operator binding(s).

These attributes are applied to the ENTITY. The string is a comma-separated list of the form:

```
"<oper_name>:<expression>"
```

where:

<oper_name>: Names one of the operator bindings from the BINDINGS attribute. The asterisk character (“*”) can be used to create the PROPERTY_MAPPING for all of the operator bindings for the module.

<expression> : Can be any expression.

Examples:

```
attribute PROP_MAP_Delay : string;  
  
attribute PROP_MAP_Delay of i2s_input_wait_es :  
    entity is "*:interpolate(1)";  
  
attribute PROP_MAP_SeqDelay : string;  
  
attribute PROP_MAP_SeqDelay of i2s_input_wait_es :  
    entity is "read_port:0";  
  
attribute PROP_MAP_InitDelay : string;  
  
attribute PROP_MAP_InitDelay of i2s_input_wait_es :  
    entity is "read_port:1";
```

CHAR_RANGE Attribute

This attribute defines the characterization range for the generic to which it is applied.

Examples:

```
attribute CHAR_RANGE : string;  
  
attribute CHAR_RANGE of bits_per_frame : constant is "16,32";  
  
attribute CHAR_RANGE of width : constant is "40,64";  
  
attribute CHAR_RANGE of arst_active : constant is "0";
```

INPUT_REGISTER Attribute

This attribute defines whether the input port should have INPUT_REGISTER = true. It is applied to an ENTITY_PORT.

Examples:

```
attribute INPUT_REGISTER : boolean;

attribute INPUT_REGISTER of i2s_sclk : signal is true;
```

NON_CHAR_PARAM Attribute

This attribute indicates whether the generic to which it is applied should only be a generic and not a characterization parameter.

Example:

```
attribute NON_CHAR_PARAM : boolean;
```

VAR_<variable_name> Attribute

This attribute will cause a variable (<variable_name>) to be added to the library module with the value specified. This attribute applies to the ENTITY.

Examples:

```
attribute VAR_trans_rsc_class : string;

attribute VAR_trans_rsc_class of i2s_input_wait_es :
    entity is "i2s_input_write_es_trans_rsc_class";
```

Examples

Refer to [“Importing Netlists”](#) on page 101.

Related Commands

library add	library rename
library characterize	library remove
library edit	library report
library get	library save
library load	library save_commands (Deprecated)
	library set

library load

Load the specified library file(s).

Syntax

```
library load ?<switches>? ?<args>?
```

<switches>	Valid switches: (Optional)
-libname <string>	library name to load from file
-recover	attempt to restore characterization backup library
-quiet	supress output
<args>	library filename(s) (Optional)

Arguments

- **-libname <string>**

The name of a library contained in the library file being loaded. If the library file contains more than one library, use this option to selectively load an individual library.

- **-recover**

When this option is used, Catapult:

- a. Loads the specified library file and gather all of the libraries within the file.
- b. For each library in the file, Catapult looks for a matching “*.wlib” file. A “*.wlib” file is a backup file of a library that was auto-saved by Catapult.
 - i. If a matching “*.wlib” file is found and its file modification time is newer than the library file, Catapult will attempt to load from the “*.wlib” file.
 - ii. If the “*.wlib” library is loaded successfully it will replace the one loaded from the library file.

The `-recover` option may generate any of the following warning messages:

The following message means the characterization working directory does not exists or the *.wlib file is not in the directory.

```
[LIB-89] Unable to locate backup library
```

The following two messages mean that the loaded library is newer (by file modification time) and that the backup library was not loaded.

```
[LIB-90] Ignoring backup library '%1!s!', file modification time is equal to the original library
```

```
[LIB-91] Ignoring backup library '%1!s!', file modification time is older than the original library
```

The following message means the backup library failed to load for some reason.

```
[LIB-92] Ignoring backup library '%1!s!', library appears to be corrupted
```

If a library was restored, the following comment will be shown.

```
[LIB-93] Restored backup library '%!s!'
%!s! will be the relative path to the *.wlib from the .lib .
```

- **-quiet**

This option suppresses all comments that would normally be sent to the transcript window by the “library load” command. By default, the command transcripts information about the libraries being loaded.

- **<args>**

The filename or pathname of the library file(s) to be loaded. If only a filename is specified, Catapult first searches the “Library Search Path” for the file, and then searches the current working directory. Refer to “[Set Component Library Options](#)” on page 31 for information about setting the “Library Search Path” value.

Description

Use this command to load existing libraries into Library Builder.

Example

This example loads two library files. The path to file “../libs/my_lib_1” is relative to the current directory. The other file, my_lib_2.lib, does not have a path qualifier because is either located in the current working directory or in a directory that is included in the tool’s Library Search Path.

```
library load ../libs/my_lib_1.lib my_lib_2.lib
# Reading component library '../my_lib_1.lib'... (LIB-49)
# Reading component library 'my_lib_2.lib'... (LIB-49)
```

Related Commands

[library add](#)
[library characterize](#)
[library edit](#)
[library get](#)
[library import](#)

[library rename](#)
[library remove](#)
[library report](#)
[library save](#)
[library save_commands \(Deprecated\)](#)
[library set](#)

library rename

Renames objects in the library database.

Syntax

```
library rename ?<path>? ?<switches>? ?<args>?

<path>                Hierarchical database path (Optional)
<switches>            Valid switches: (Optional)
  --                  End <switches> parsing
  -recurse <string>   Everything under
  -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                        Return data format
    value             just matching values
    path              just matching paths
    pathvalue         path and value combination for array set
    leaf              just matching leaves
    leafvalue         leaf and value combination for array set
    advanced          hierarchical list structure
    none              no return value
  -checkpath <bool>   Error on path not found
  -match <exact|glob> Path match type
    exact             exact paths only
    glob              glob paths
  -info <bool>        Return object info
<args>                Database subpaths (Optional)
```

Arguments

- **<path>**
A hierarchical path into the SIF database. The root node of <path> is /LIBRARY. For more information, refer to [“Command Interface to the SIF Database”](#) on page 138.
- **Common Switches**
Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section, [“Common Command Switches”](#) on page 141.
- **<args>**
Sub-path(s) into the Catapult SIF database. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to [“Using Sub-Path Arguments”](#) on page 140.

Description

Use this command to rename objects in the library database. Only the following types of objects can be renamed: libraries, modules, operators and ports. Attempting to rename other types of objects will generate an error.

Example

This example changes the name of a module.

```
library rename /LIBS/my_lib/MODS/my_mux_test /LIBS/my_lib/MODS/my_mux
```

Related Commands

[library add](#)
[library characterize](#)
[library edit](#)
[library get](#)
[library import](#)

[library load](#)
[library remove](#)
[library report](#)
[library save](#)
[library save_commands \(Deprecated\)](#)
[library set](#)

library remove

Remove objects from the library database.

Syntax

```
library remove ?<path>? ?<switches>? ?<args>?

<path>                Hierarchical database path (Optional)
<switches>            Valid switches: (Optional)
  --                  End <switches> parsing
  -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                      Return data format
    value              just matching values
    path               just matching paths
    pathvalue          path and value combination for array set
    leaf               just matching leaves
    leafvalue          leaf and value combination for array set
    advanced           hierarchical list structure
    none               no return value
  -checkpath <bool>   Error on path not found
  -match <exact|glob> Path match type
    exact              exact paths only
    glob               glob paths
  -info <bool>        Return object info
<args>                Database subpaths (Optional)
```

Arguments

- **<path>**

A hierarchical path into the SIF database. The root node of <path> is /LIBRARY. For more information, refer to [“Command Interface to the SIF Database”](#) on page 138.

- **Common Switches**

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section, [“Common Command Switches”](#) on page 141.

- **<args>**

Sub-path(s) into the Catapult SIF database. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to [“Using Sub-Path Arguments”](#) on page 140.

Description

Use this command to remove objects from the library database. It can remove an entire library or specific objects within a library.

Example

Example 1:

This example removes an entire library from the database.

```
library remove /LIBS/my_lib
```

Example 2:

This example removes all modules that have the “test_” prefix. The “-return pathvalue” switch is used to see what object were found.

```
library remove /LIBS/my_asic_lib/MODS/test_* -match glob -return pathvalue  
# /LIBS/my_asic_lib/MODS/test_add {} /LIBS/my_asic_lib/MODS/test_mux {}
```

Related Commands

[library add](#)
[library characterize](#)
[library edit](#)
[library get](#)
[library import](#)

[library load](#)
[library rename](#)
[library report](#)
[library save](#)
[library save_commands \(Deprecated\)](#)
[library set](#)

library report

Generates a report about characterization data in the specified libraries or library element.

Syntax

```
library report libpath ?...? ?options?
```

libpath ?...?	Hierarchical database path(s)
?options?	Valid options
-compare <string>	Compare against reference library
-status <bool>	Generate library status report
-summary <bool>	Generate library summary report
-filename <string>	Saves report to filename
-fullname	Adds reference library name to component name

Arguments

- ***libpath ?...?***

Hierarchical path(s) into the SIF database. A path to the root node of a library or specific modules in a library. The wildcard character “*” can be used to create glob expressions. The root node of <path> is /LIBRARY. For more information, refer to “[Command Interface to the SIF Database](#)” on page 138.

- ***-compare <string>***

Generates a comparison report that compares `libpath` against a reference library. The <string> argument specifies the database path of the reference library. It must resolve to the top-level node of the reference library and not individual modules within it.

- ***-status <bool>***

When set to “true,” the command reports only characterization pass/fail status and run times for the modules in `libpath`.

- ***-summary <bool>***

When set to “true,” the command reports all characterization data and statistics for the modules in `libpath`.

- ***-filename <string>***

Write the report to the specified file. If Library Builder is running in GUI mode, the report file will also be displayed in a DesignPad window.

- ***-fullname***

Prefix the library name to the component names in the report.

Description

Use this command get the characterization status for one or more libraries, or for individual components within the libraries. Use the “-compare” option to compare the characterization information with that of a reference library. The report is automatically displayed in a DesignPad window in the Library Builder GUI.

Example

Example 1:

This example generates a summary report of all components in the `my_lib` library.

```
library report /LIBS/my_lib -summary true
```

Example 2:

This example compares the characterization data of the `mgc_add` module in the `my_lib` library against the same module in the reference library `my_ref_lib`.

```
library report /LIBS/my_lib/MODS/mgc_and -compare /LIBS/my_ref_lib
```

Related Commands

- [library add](#)
- [library characterize](#)
- [library edit](#)
- [library get](#)
- [library import](#)

- [library load](#)
- [library rename](#)
- [library remove](#)
- [library save](#)
- [library save_commands \(Deprecated\)](#)
- [library set](#)

library save

Save one or more libraries to the file system.

Syntax

```
library save ?<paths>? ?<switches>?

<paths>           database paths (Optional)
<switches>       Valid switches: (Optional)
  -filename <file>  save libraries to file
  -quiet           suppress output
```

Arguments

- **<paths>**
Hierarchical path(s) into the SIF database. The root node of <path> is /LIBRARY. For more information, refer to [“Command Interface to the SIF Database”](#) on page 138.
- **-filename <file>**
The name of the file in which the library (or libraries) will be saved. If the specified filename does not exist, it will be created. If the filename does exist, the file will be overwritten.

Description

This command allows you to save library edits and characterization data to disk. Use the “-filename” switch to save the library under a new filename or combining multiple libraries in a single file. If the “-filename” switch is omitted, the library is saved to <library_name>.lib. If multiple library paths are specified and the “-filename” switch is omitted, each library is saved in a separate file.

Example

Example 1:

This example saves the specified libraries to their default filenames.

```
library save /LIBS/my_asic_lib /LIBS/my_test_asic_lib
#
# Writing component library 'my_asic_lib' (license:
LIBRARY_BUILDER=catapultbasic)... (LIB-51)
# Component library written to 'my_asic_lib.lib' (LIB-52)
# Writing component library 'my_test_asic_lib' (license:
C=catapultbasic, LIBRARY_BUILDER=catapultbasic)... (LIB-51)
# Component library written to 'my_test_asic_lib.lib' (LIB-52)
```

Example 2:

This example saves two libraries in a single file.

```
library save /LIBS/my_asic_lib /LIBS/my_test_asic_lib
                                                    -filename my_comb_libs.lib
#
# Writing component library 'my_asic_lib' (license:
LIBRARY_BUILDER=catapultbasic)... (LIB-51)
```

```
# Writing component library 'my_test_asic_lib' (license:  
C=catapultlibasic, LIBRARY_BUILDER=catapultlibasic)... (LIB-51)  
# Component libraries written to 'my_comb_libs.lib' (LIB-53)
```

Related Commands

[library add](#)
[library characterize](#)
[library edit](#)
[library get](#)
[library import](#)

[library load](#)
[library rename](#)
[library remove](#)
[library report](#)
[library save_commands \(Deprecated\)](#)
[library set](#)

library save_commands (Deprecated)

Saves the characterization command data of the specified library.

Syntax

```
library save_commands <path> ... ?<options>?
-filename -- Filename to write commands
```

Arguments

- **<path>s**
A hierarchical path into the SIF database. The root node of <path> is /LIBRARY. For more information, refer to “[Command Interface to the SIF Database](#)” on page 138.
- **-filename**
The name of the file in which the library (or libraries) will be saved.

Description

This command is deprecated and should be avoided. The `library save_commands` writes out the “characterization data” values in the form of “`library add`” commands. The file can be run to restore the initial setup for a new library and the data point area/delay values.

Example

```
library save_commands /LIBS/my_asic_lib
-filename "/Catapult/Libs/my_asic_lib.tcl"
```

The above example saves the characterization data in the Tcl file `my_asic_lib.tcl`. The contents of the output file is similar to the following:

```
library add /LIBS/my_asic_lib -TEMPLATENAME mgc_tmpl_beh_dc -DATASETS
{100% 75% 50% 0%}
library add /LIBS/my_asic_lib -WORKING_DIR
{/Catapult/LibraryBuilder/my_asic_lib.char}
library add /LIBS/my_asic_lib/VARS/ui_libtitle -VALUE my_asic_lib
library add /LIBS/my_asic_lib/VARS/vendor -VALUE {LSI Logic}
library add /LIBS/my_asic_lib/VARS/technology -VALUE sample-130nm
...
```

Related Commands

[library add](#)
[library characterize](#)
[library edit](#)
[library get](#)
[library import](#)

[library load](#)
[library rename](#)
[library remove](#)
[library report](#)
[library save](#)
[library set](#)

library set

Configure objects in the library database.

Syntax

```
library set ?<path>? ?<switches>? ?<args>?

<path>                Hierarchical database path (Optional)
<switches>           Valid switches: (Optional)
  --                 End <switches> parsing
  -return <value|path|pathvalue|leaf|leafvalue|advanced|none>
                    Return data format
    value            just matching values
    path             just matching paths
    pathvalue        path and value combination for array set
    leaf             just matching leaves
    leafvalue        leaf and value combination for array set
    advanced         hierarchical list structure
    none             no return value
  -checkpath <bool>  Error on path not found
  -match <exact|glob> Path match type
    exact            exact paths only
    glob             glob paths
  -info <bool>       Return object info
<args>                Database subpath and value combinations (Optional)
```

Arguments

- **<path>**

A hierarchical path into the SIF database. The root node of <path> is /LIBRARY. For more information, refer to [“Command Interface to the SIF Database”](#) on page 138.

- **Common Switches**

Command switches listed in the “Syntax” section above that are not described on this command reference page are common to many of the Catapult commands. Many of these switches control how the database is searched and how the returned data is displayed. For information about those switches, refer to the section, [“Common Command Switches”](#) on page 141.

- **<args>**

Sub-path(s) to data objects the Catapult SIF database, and data values to be assigned. Sub-paths are relative to the <path> argument. For detailed information about specifying database sub-paths, refer to [“Using Sub-Path Arguments”](#) on page 140.

For each sub-path specified, the next argument is assumed to be the data value for the object at the sub-path. If the sub-path is a wildcard expression, the data value is assigned to all matching data objects.

Description

This command modifies the value setting(s) in the specified library data object(s). This command cannot add or delete data object. For information about the general structure of the library database, refer to the description of the [library get](#) command on page page 182.

Examples

This example changes the value of the “technology” variable to “sample-130nm”:

```
library set /LIBS/my_asic_lib/VARS/technology/VALUE sample-130nm
# /LIBS/my_asic_lib/VARS/technology/VALUE sample-130nm
```

This example assigns the value “init_val” to all objects under the “VARS” node that start with “TEST_”. The “-match glob” switch is required when wildcard expressions are used.

```
library set /LIBS/my_asic_lib/VARS/TEST_*/VALUE -match glob init_val
# /LIBS/my_asic_lib/VARS/TEST_a/VALUE init_val
/LIBS/my_asic_lib/VARS/TEST_b/VALUE init_val
/LIBS/my_asic_lib/VARS/TEST_c/VALUE init_val
```

This example combines examples 1 and 2 above into single command by using sub-path and value pairs in the <args> field.

```
library set /LIBS/my_asic_lib/VARS -match glob -- TEST_*/VALUE init_val
                                                technology/VALUE sample_130nm
# /LIBS/my_asic_lib/VARS/technology/VALUE sample_130nm
/LIBS/my_asic_lib/VARS/TEST_3/VALUE init_val
/LIBS/my_asic_lib/VARS/TEST_a/VALUE init_val
/LIBS/my_asic_lib/VARS/TEST_b/VALUE init_val
/LIBS/my_asic_lib/VARS/TEST_c/VALUE init_val
```

Related Commands

[library add](#)
[library characterize](#)
[library edit](#)
[library get](#)
[library import](#)

[library load](#)
[library rename](#)
[library remove](#)
[library report](#)
[library save](#)
[library save_commands \(Deprecated\)](#)

logfile

Controls the logfile options for the Library Builder session.

Syntax

```
open <name [-project]>  
close [-project]  
move [-enable] [-disable] [<filename>]  
name  
save_commands <filename>
```

Arguments

- ***open <name [-project]>***
If no name is specified in a file, the temporary directory will be opened.
- ***-project***
Opens/Closes a project logfile (the default is a session logfile).
- ***close [-project]***
Stops logging commands.
- ***move***
Moves the session logfile.
- ***-enable***
Causes the session logfile to be moved to the project directory when created.
- ***-disable***
Causes a session logfile not to be moved.
- ***<filename>***
Name of the file or directory to which the session logfile should be moved.
- ***name***
Return current logfile name.
- ***save_commands <filename>***
Saves commands from the active session logfile to the filename specified.

Description

Use this command to control various logfile options in the Library Builder session.

Example

```
logfile -move /Catapult_2/lcbg11p.lib
```

options defaults

Resets all flow options to default settings

Syntax

```
options defaults ?<switches>?
```

```
<switches>  
-clean
```

```
Valid switches: (Optional)  
restore to predefined options
```

Arguments

- *-clean*

Removes all flow options from the project. Flows must be reloaded in order to re-establish the options in the project.

Description

The “options defaults” command resets the default option values for all flows loaded in the session. This command does not update the flow option values saved in the [Catapult C Library Builder Registry](#). If old option settings were saved previously, you may want to save the options again after resetting them.

Because flow options are saved in the registry, they can become out of synch with the flow if a new version of the flow is loaded and that new version has changed its default options. Use the “-clean” switch to purge all obsolete and conflicting options, the reload the flow to re-synchronize the flow options, and finally save the new options to the registry.

Example

The following example reverts all loaded flow settings to the factory defaults:

```
options defaults
```

The following example synchronizes an older version of flow settings with a new version:

```
options defaults -clean
```

Related Commands

[options exists](#)
[options get](#)
[options load](#)

[options save](#)
[options set](#)

options exists

Checks whether an option exists in the database.

Syntax

```
options exists <name>

<name>      option name (Required)
<switches> Valid switches: (Optional)
  -default  has a default value
```

Argument

- **<name>**

The name of an option in the project database. This argument has the form:

```
<section>/<option>
```

The `<section>` portion is a section name found in the Catapult Library Builder Options window or the Catapult initialization file (see “[Setting Library Builder Options](#)” on page 24 and “[The Catapult Initialization File](#)” on page 48). The `<option>` portion is the target option name in that section.

In the case of hierarchical sections (“Flows”), this argument has the form:

```
<section>/<sub-section>/<option>
```

- ***-default***

Reports whether or not a factory default value is defined for the specified option. The current value of the option is not evaluated, only the existence of its factory default.

Description

The “options exists” command reports whether or not an option (or its default value) exists in the database. The command returns either ‘1’ (true) or ‘0’ (false).

Example

This example checks for the existence of the “Path” option in the “Precision” flow. The return value is true.

```
options exist /Flows/Precision/Path
# 1
```

Related Commands

[options defaults](#)
[options get](#)
[options load](#)

[options save](#)
[options set](#)

options get

Returns the value of the specified option.

Syntax

```
options get <name> ?<switches>?
```

<name>	option name (Required)
<switches>	Valid switches: (Optional)
-all	all options
-default	default value
-hidden ?<bool>?	hidden options

Arguments

- **<name>**

The name of an option in the project database. This argument has the form:

```
<section>/<option>
```

The <section> portion is a section name found in the Catapult Library Builder Options window or the Catapult initialization file (see “[Setting Library Builder Options](#)” on page 24 and “[The Catapult Initialization File](#)” on page 48). The <option> portion is the target option name in that section.

In the case of hierarchical sections (“ProjectInit” and “Flows”), this argument has the form:

```
<section>/<sub-section>/<option>
```

- **-all**

Performs a recursive search and returns a list of all option names in the specified section and its sub-sections. Option values are not returned.

- **-default**

Returns the factory default value of the specified option.

- **-hidden ?<bool>?**

This switch includes hidden option names in the return value. Option values are not returned. The <bool> argument is true by default. Set <bool> to false to explicitly exclude hidden options from the return value.

Description

The “options get” command returns the value of an option in the in-memory database. If the <name> argument is omitted, the command returns a list of valid section names. If only a section name is specified, a list of valid option names is returned.

Examples

Example 1

In this example the <name> argument is omitted and the command returns the list of valid section names that can be used.

```
options get

# General Message ComponentLibs CatapultC Farm TextEditor Flows
```

Example 2:

This example provides only a section name (“Farm”) in order to get a list of valid option names in that section.

```
options get Farm

# EnableFarm HostDatabasePath MaxParallelTasks
SuspendHostOnInvocationFailure SuspendHostOnFailure RshCommand
```

Example 3:

This example illustrates the use of the “-default” switch. The first command returns the current value of the “/Farm/HostDatabasePath” option. The second command returns the factory default value.

```
options get Farm/HostDatabasePath
# /lib_builder/my_host_db_path

options get Farm/HostDatabasePath -default
# ./hostdb
```

Example 4:

The three commands in this example demonstrate the difference between the “-all” switch and the “-hidden” switch. The first command is the default case in which neither switch is used. It simply returns the visible set of sub-section names under the “Flows” section.

```
options get Flows
# FlowSearchPath DesignCompiler Precision
```

In the second command the “-hidden” switch exposes all of the sub-section names.

```
options get Flows -hidden
# FlowSearchPath DesignCompiler Precision Enable-DesignCompiler Enable-
Precision
```

Finally, the “-all” switch in the third command returns every sub-section as well as every option name under each sub-section. Each value in the list is a complete hierarchical path.

```
options get Flows -all
# FlowSearchPath DesignCompiler DesignCompiler/SearchPath
DesignCompiler/Path DesignCompiler/Flags DesignCompiler/RunBatch
DesignCompiler/ShellType DesignCompiler/ShellExe
DesignCompiler/FOLDERNAME Precision Precision/Path Precision/Flags
Precision/addio Precision/retiming Precision/run_pnr
Precision/Frontend2004 Precision/Exe Precision/FOLDERNAME Enable-
DesignCompiler Enable-Precision
```

Related Commands

[options defaults](#)
[options exists](#)
[options load](#)

[options save](#)
[options set](#)

options load

Loads saved option settings.

Syntax

```
options load ?<switches>?
```

<switches>	Valid switches: (Optional)
-file <string>	name of file to load
-registry <string>	registry location to load
-quiet	suppress messages

Arguments

- **-file <string>**

A pathname to an options file. Use this switch to load option settings from the specified file instead of the default location. If only the leaf name is specified, Catapult looks for the file in the current working directory.

- **-registry <string>**

(For Windows operating systems only) A Windows registry path where the Catapult option settings are stored. The standard location for Catapult options is as follows, where <version> is the Catapult version number.

```
{HKEY_CURRENT_USER\Software\Mentor Graphics\Catapult Synthesis\<version>}
```

- **-quiet**

Suppress informational messages returned by the command. Does not suppress error or warning messages.

Description

The “options load” command loads initialization options from the default location unless the “-file” or “-registry” switch is used to specify an alternate location. The default location is determined during invocation of the Catapult session. Catapult searches for an initialization file in the following search order:

1. A `catapult.ini` file in the current working directory
2. A `catapult.ini` file in the user’s HOME directory
3. The Catapult registry (see [“Catapult C Library Builder Registry”](#) on page 48)

The first one found becomes the default location for the duration of the catapult session, or until a different location is configured by using “options save” command with the “-default” switch.

Examples

Example 1:

This example simply loads option settings from the default location, which is the Catapult registry in this case.

```
options load
# Loading options from '$HOME/.catapult/2007a.ixl.reg'.
```

Example 2:

This example loads option settings from the file named “opt_settings.ini” in the current working directory.

```
options load -file opt_settings.ini
```

Example 3:

This example is the same as above, but specifies the full path to the file.

```
options load -file /jdoe/opt_settings.ini
```

Related Commands

[options defaults](#)
[options exists](#)
[options get](#)

[options save](#)
[options set](#)

options save

Save the in-memory options to the Catapult registry or an alternate file.

Syntax

```
options save ?<switches>?
```

```
<switches>          Valid switches: (Optional)
  -default <load message> <save message>
                    configures the default load and save location
  -file <string>     name of file to save
  -registry <string> name of registry key
  -section <string>  starting section name
  -hidden ?<bool>?  option is hidden
  -quiet            suppress messages
```

Arguments

- ***-default <load message> <save message>***

Configures default settings for the “[options load](#)” and “options save” commands. It sets the default location where Library Builder option settings are stored, and sets the message strings that are displayed by the “options load” and “options save” commands. The location setting is specified by either the “-file” or “-registry” switches, one of which must be used in conjunction with this switch.

When this switch is used, the “options save” command does not save options settings. It only configures the default settings. The new settings remain in effect for the duration of the current Library Builder session. New Library Builder sessions are always initialized to the system default settings.

- ***-file <string>***

A pathname to a file in which the option settings will be saved. Use this switch to save option settings to a file other than the default location. If only the leaf name of the file is specified, then the file is saved to the current working directory. If the file exists, it is overwritten. Otherwise, the file is created.

Use the filename “catapult.ini” in either the project directory or in the \$HOME directory in order to have Catapult automatically load the options at during invocation. Refer to “[Restoring Options](#)” on page 48 for more information.

- ***-registry <string>***

(For Windows operating systems only) A Windows registry path where the Catapult option settings will be stored. A hierarchy of keys and values is created below the specified location in the registry. The standard location for Catapult options is as follows, where <version> is the Catapult version number.

```
{HKEY_CURRENT_USER\Software\Mentor Graphics\Catapult Synthesis\<version>}
```

- **-section <string>**

Saves only those options stored in the specified section of the file or registry. Section names correspond to the names in the Catapult C Synthesis Options window. This switch must be used in conjunction with the “-file” or “-registry” switch. When saving to existing file, the entire contents of the file are overwritten and only the specified section is saved. Saving to the Windows registry overwrites only the specified section.
- **-hidden ?<bool>?**

This switch includes hidden options in the save operation. The <bool> argument is true by default. Set <bool> to false to explicitly exclude hidden options.
- **-quiet**

Suppress informational messages returned by the command. Does not suppress error or warning messages.

Description

This command saves the in-memory option settings to default location, unless the “-file” or “-registry” switch is used to specify an alternate location. The system default location is determined during invocation of the Catapult session. Refer to the section “[Restoring Options](#)” on page 48 for more information.

When “-file” or “-registry” is used in conjunction with the “-default” switch, the specified path becomes the default location used by the “options load” and “options save” commands. In addition, the default return string can be changed by using the “-default” switch. Refer to the “-default” switch above for more details.

Note



Enable the “General/SaveSettings” option to have Library Builder automatically save the options database when exiting.

Examples

Example 1:

This example saves the options to a Catapult initialization file in the user’s HOME directory.

```
options save -file /user/johnd/catapult.ini
```

Example 2:

This example uses the “-default” switch to configure the default location and messages for the “options load” and “options save” commands.

```
options save -default "Loading options from file: opt_settings.ini"  
"Saving options to file: opt_settings.ini" -file "/jdoe/opt_settings.ini"
```

As a result, the “options load” and “options save” commands now read/write the options in the file “/jdoe/opt_settings.ini” by default. And the commands return the specified message strings as demonstrated in the following command calls:

```
options load
```

Commands

options save

```
# Loading options from file: opt_settings.ini  
  
options save  
# Saving options to file: opt_settings.ini
```

Related Commands

[options defaults](#)
[options exists](#)
[options get](#)

[options load](#)
[options set](#)

options set

Sets the value of an option in the database.

Syntax

```
options set <name> <value> ?<switches>?
```

```
<name>          option name (Required)  
<value>        option value (Required)
```

Arguments

- **<name>**

The name of an option in the project database. This argument has the form:

```
<section>/<option>
```

or

```
<section> <option>
```

The `<section>` portion is a section name found in the Catapult Library Builder Options window or the Catapult initialization file (see [“Setting Library Builder Options”](#) on page 24 and [“The Catapult Initialization File”](#) on page 48). The `<option>` portion is the target option name in that section.

In the case of hierarchical sections (“Flows”), this argument has the form:

```
<section>/<sub-section>/<option>
```

or

```
<section>/<sub-section> <option>
```

Use the [“options get”](#) command to get a list of option names in the database.

- **<value>**

The new value(s) to be assigned to the option. Lists and strings containing spaces must be enclosed in double quotes or braces.

Description

The `options set` command modifies the value of an option in the database. If the command succeeds, the new value is returned as a comment. Otherwise, an error message is returned. The modified values are valid for the duration of the Library Builder session. To preserve the values for future sessions, use the [“options save”](#) command. For information about setting options from the Library Builder GUI, refer to [“Setting Library Builder Options”](#) on page 24.

Example 1

The following example shows the different forms of the `<name>` argument.

```
options set General/NetlistFormat Verilog  
# Verilog
```

```
options set General NetlistFormat Verilog
# Verilog

options set Flows/Precision/FOLDERNAME my_precision_folder
# my_precision_folder

options set Flows/Precision FOLDERNAME my_precision_folder
# my_precision_folder
```

Example 2

The following example specifies a list of values. The first string in the list is enclosed in double quotes because it contains a space. The entire list is enclosed in braces.

```
options set ComponentLibs/TemplateSearchPath
           {{"$MGC_HOME/pkgs/siflibs/templates"} /catapult/my_templates}
# {"$MGC_HOME/pkgs/siflibs/templates"} /catapult/my_templates
```

Related Commands

[options defaults](#)
[options exists](#)
[options get](#)

[options load](#)
[options save](#)

quit

Exits Library Builder.

Syntax

```
quit [-force] [-code]
```

Arguments

- ***-force***
Optional switch that forces Library Builder to terminate the current process and exit immediately without saving changes. By default, Library Builder finishes the current process before quitting.
- ***-code***
Optional switch that specifies an application code that Library Builder returns to the shell on exit.

Description

Exits Library Builder.

Example

```
quit
```

set_working_dir

Specifies the working directory.

Syntax

```
set_working_dir <directory_pathname>
```

Arguments

- **directory_pathname**
Required option that specifies a pathname to an existing directory.

Description

Specifies the working directory that all library sub-directories and files are written to. By default, the working directory is the the directory where Library Builder is invoked from.

Example

```
set_working_dir E:/designs/mydesigns
```

Related Commands

[library load](#)

[library save_commands \(Deprecated\)](#)

utility farm add

Adds hosts to the Library Farm host database.

Syntax

```
utility farm add <switches>
```

<switches>	Valid switches (Required)
-name <string>	Name of host to add
-comment <string>	Additional host information
-group <string>	Host group identifier
-count <int>	Number of tasks that can run on the host

Arguments

- **-name <hostname>**
Required switch and string that specifies the name of a host to add.
- **-comment <addl_host_info>**
Optional switch and string that specifies comment information to append to the added host.
- **-group <host_group_name>**
Optional switch and string that specifies the name of a host group to associate the new host with. This option allows you to operate on a group of hosts.
- **-count <int>**
Optional switch and integer that specifies how many tasks to run on the new host. The number of tasks are incremented by the specified value. Default is 1.

Description

Use this command to add a host to the Library Farm host database.

Examples

The following example adds a host *foo* to the host group *group1* and sets it to run five tasks:

```
utility farm add -name foo -group group1 -count 4
```

Related Commands

[utility farm get](#)
[utility farm release](#)
[utility farm remove](#)

[utility farm reserve](#)
[utility farm reset](#)
[utility farm set](#)

utility farm get

Returns information about a specified host or the host database in the Library Farm.

Syntax

```
utility farm get ?<args>? ?<switch>? <args>
```

<args>	Valid args (Optional)
directory	Directory name
state >	State of host
database	Database info
<switch>	Valid switches (Optional)
-hostkey <string>	Hostkey
<args>	Valid args (Required)
hostname	Host name
hostinfo	Host information
comment	Comment information
group	Host group name

Arguments

- **directory**
Optional argument that returns the directory location of the host database.
- **state**
Optional argument that returns the state of the host database, either enabled or disabled.
- **database**
Optional argument that reports the host database information, including all the hosts.
- **-hostkey <hostkey>**
Optional switch and string that specifies the hostkey of a host to report on. The hostkey is automatically assigned to a host when it is allocated with the *-reserve* option.
- **hostname**
Required argument that returns the name of the host associated with as specified hostkey.
- **hostinfo**
Required argument that returns information on the host associated with the specified hostkey.
- **comment**
Required argument that returns comment information for the host associated with the specified hostkey.
- **group**
Required argument that returns the group name for the host associated with the specified hostkey.

Description

This command reports on the Library Farm host database. You can query the entire database or a host within the database. If no switches are used, *utility farm get* displays the host database information.

Examples

The following example shows the results of a host database query:

```
utility farm get database
# stage40a##task_1 {status idle hostname stage40a group {} comment
task_1}
```

The following example returns the name of a host associated with the *host2* hostkey:

```
utility farm get -hostkey host2 hostname
foo
```

Related Commands

[utility farm add](#)
[utility farm release](#)
[utility farm remove](#)

[utility farm reserve](#)
[utility farm reset](#)
[utility farm set](#)

utility farm release

Releases an allocated host.

Syntax

```
utility farm release <hostkey>
```

Arguments

- **hostkey**

Required string that specifies the hostkey for the host to release. Multiple space-separated hostkeys can be specified. A hostkey is automatically assigned and returned when a host is allocated with the *-reserve* switch.

Description

This command releases one or more hosts in the Library Farm host database from being allocated. A host is allocated with the *-reserve* option. If no host is specified all hosts are released.

Examples

The following example releases host1 and host2:

```
utility farm release host1 host2
```

Related Commands

[utility farm add](#)
[utility farm get](#)
[utility farm remove](#)

[utility farm reserve](#)
[utility farm reset](#)
[utility farm set](#)

utility farm remove

Deletes hosts from the Library Farm host database.

Syntax

```
utility farm remove ?<switches>?
```

<switches>	Valid switches (Optional)
-name <string>	host name
-comment <string>	Host comment information
-group <string>	Host group name

Arguments

- **-name <hostname>**
Optional switch and string that deletes the specified host.
- **-comment <addl_host_info>**
Optional switch and string that deletes a host associated with the specified comment information.
- **-group <host_group>**
Optional switch and string that deletes a specified host group.

Description

Use this command to remove one or more hosts from the Library Farm host database. If no hosts are specified, all hosts are deleted from the host database.

Examples

The following example deletes the host named *foo* from the host database:

```
utility farm remove -name foo
```

Related Commands

[utility farm add](#)
[utility farm get](#)
[utility farm release](#)

[utility farm reserve](#)
[utility farm reset](#)
[utility farm set](#)

utility farm reserve

Reserves a host from the host database.

Syntax

```
utility farm reserve
```

Arguments

None

Description

This command reserves a host in the Library Farm host database and returns a hostkey identifier for it. If no hosts are available, an error displays. Once a host is reserved, you can use the hostkey and the *utility farm* commands to assign tasks to the reserved host.

Examples

The following example reserves a host and returns the *host1* hostkey:

```
utility farm reserve  
host1
```

Related Commands

[utility farm add](#)
[utility farm get](#)
[utility farm remove](#)

[utility farm reset](#)
[utility farm set](#)
[utility farm release](#)

utility farm reset

Resets hosts in the Library Farm host database to an idle state.

Syntax

```
utility farm reset ?<switches>?
```

<switches>	Valid switches (Optional)
-name <string>	Host name
-comment <string>	Host comment information
-group <string>	Host group name

Arguments

- **-name <hostname>**
Optional switch and string that specifies the name of a host to reset.
- **-comment <addl_host_info>**
Optional switch and string that specifies comment information to identify the host to reset.
- **-group <host_group>**
Optional switch and string that specifies a host group to reset.

Description

Use this command to reset one or more hosts in the Library Farm host database. If no host is specified all hosts are reset.

Examples

The following example resets the host named *foo*:

```
utility farm reset -name foo
```

Related Commands

[utility farm add](#)
[utility farm get](#)
[utility farm release](#)

[utility farm remove](#)
[utility farm reserve](#)
[utility farm set](#)

utility farm set

Configures the Library Farm host database and allocated hosts.

Syntax

```
utility farm set ?<switches>? <args>
```

<switches>	Valid switches (Optional)
-directory <string>	host database directory
-state <enabled disabled>	Enables/disables the database
-hostkey <string>	Specifies a hostkey
-information <string>	information associated with a hostkey
-append_information <name> <description>	Appends information to a hostkey
<name>	Name value (Required)
<description>	Description value (Required)
-status <string>	Sets the status of a hostkey

Arguments

- **-directory <dir_path>**
Switch and string that sets a directory for the host database.
- **-state <enabled/disabled>**
Switch and argument pair that enables or disables the host database.
- **-hostkey <hostkey>**
Switch and string that specifies a hostkey of an allocated host to configure. The hostkey is automatically assigned to a host when it is allocated with the *-reserve* option. Use [utility farm get](#) with the hostkey to retrieve additional host information.
- **-append_information <name> <description>**
Switch and double string pair that appends information to the host associated with the specified hostkey. Information includes:
 - Name
 - Description
- **-information <hostinfo>**
Switch and string that adds information to the host associated with the specified a hostkey.
- **-status <string>**
Switch and string that sets the status of a specified hostkey.

Description

This command sets values the Library Farm host database. You can set values for the entire database or an allocated host within the database.

Examples

The following example sets a host database directory path to *F:dbs2*:

```
utility farm set -directory F:/dbs2
```

The following example sets the *hostname##task_N* hostkey to the idle status:

```
utility farm set -hostkey hostname##task_N -status idle
```

Related Commands

[utility farm add](#)
[utility farm get](#)
[utility farm release](#)

[utility farm remove](#)
[utility farm reserve](#)
[utility farm reset](#)

-- Switch, 143

— A —

application exit command, 150
 application get command, 148
 application report command, 151

— B —

Blank Library template
 Multi-point characterization, 76

— C —

Catapult
 compiler options, 35, 37
 Catapult C Library Builder
 messages, 27
 Windows Shortcut, 13
 Catapult C Synthesis, 31
 catapult.ini, 48
 Saving and restoring, 47
 Characterization
 Characterize a Library, 75
 Multi-point data sets, 76
 Resetting the data before another
 characterization, 83
 -checkpath Switch, 142
 Command input window, 15
 Command Line Shell, 144
 Commands
 application exit, 150
 application get, 148
 application report, 151
 catapult -library_builder, 152
 dofile, 154
 flow get, 155
 flow package names, 157
 flow package option add, 158
 flow package option get, 160
 flow package option remove, 161
 flow package option set, 162
 flow package provide, 163

flow package require, 165
 flow package script, 167
 flow package vcompare, 168
 flow package versions, 170
 flow package vsatisfies, 171
 flow run, 172
 general command syntax, 135
 help, 173
 library add, 177
 library characterize, 180
 library edit, 181
 library get, 182
 library import, 185
 library load, 190
 library remove, 194
 library rename, 192
 library report, 196
 library save, 198
 library save_commands, 200
 library set, 201
 logfile, 203
 options defaults, 204
 options exists, 205
 options get, 206
 options load, 208
 options save, 210
 options set, 213
 quit, 215
 set_working_dir, 216
 Summary, 145

Compiler
 setting options, 35
 view settings, 37

Component library
 default options, 31

— D —

default options, 31
 dofile command, 154

— E —

Error Messages
 configuration options, 27

— F —

Farm
 default options, 33
 Files
 .catapult.tcl, 145
 flow commands
 flow get, 155
 flow package names, 157
 flow package option add, 158
 flow package option get, 160
 flow package option remove, 161
 flow package option set, 162
 flow package provide, 163
 flow package require, 165
 flow package script, 167
 flow package vcompare, 168
 flow package versions, 170
 flow package vsatisfies, 171
 flow run, 172

— G —

General command syntax, 135

— H —

Help, 23
 help command, 173
 help message, 175
 --help switch, 143
 -help switch, 143
 history tracking, 144

— I —

Import libraries, 185
 -info Switch, 142
 Input compiler options, 35

— L —

Libraries
 Characterize, 75
 Queue for multi-day run, 81
 RAM templates, 51, 68
 Library

 Reviewing library characterization passes, 81

 Troubleshooting Failures, 88

library add command, 177

Library Builder

 catapult.ini file, 47

 Initialization file, 47, 48

 Licenses, 12

 Main Menu, 13

 Overview, 11

 registry, 48

 Saving and restoring, 48

 Saving and restoring options, 47

 set default options, 24

 set working directory, 51

 View Options, 14

Library Characterization

 command, 180

 Multi-point data sets, 76

 Passes, 81

 Reset Data before another
 Characterization, 83

library characterize command, 180

Library Components

 Multi-point characterization, 76

library edit command, 181

Library Farm, 90

 Enable, 90

 Introduction, 11

 Setting up hosts, 92

library get command, 182

library import command, 185

library load command, 190

library remove command, 194

library rename command, 192

library report command, 196

library save command, 198

library save_commands, 200

library set command, 201

License report, 151

— M —

-match Switch, 141, 142

Memory Templates, 51, 68

Messages

 Naming Conventions, 30

setting severity levels, [27](#)
 Multi-Point Characterization, [76](#)

— O —

Options

- Catapult C Synthesis defaults, [31](#)
- component library, [31](#)
- general settings, [24](#)
- input compiler, [35](#)
- library farm, [33](#)
- saving, [47](#)
- system messages, [27](#)
- text editor, [37](#)

options commands

- defaults, [204](#)
- exists, [205](#)
- get, [206](#)
- load, [208](#)
- save, [210](#)
- set, [213](#)

— P —

Path and Sub-path Commands, [139](#)
 Path and Sub-Path Search Rules, [138](#)
 Path and Sub-Path Switches, [141](#)

— Q —

Queue Libraries for Multi-day Run, [81](#)
 quit command, [215](#)

— R —

RAM

- Editing components information, [69](#)
- Editing formula information, [70](#)
- Editing input registers, [72](#)
- Editing RAM timing, [71](#)
- Editing the library name, [68](#)

RAM Libraries

- Templates, [51](#), [68](#)
- recurse Switch, [143](#)
- return Switch, [141](#)

— S —

Script

- automatically running a TCL script, [145](#)
- Tcl commands, [143](#)

set_working_dir command, [216](#)

Startup file
 .catapult.tcl file, [145](#)

System messages
 help, [175](#)

System Registry, [48](#)

— T —

Tcl

- language, [135](#)

Tcl script

- command line with path, [144](#)
- Interactive Command Line Shell, [143](#)
- LOG file, [143](#)
- run script, [144](#)

Template Libraries

- RAM, [51](#), [68](#)

Text editor

- default options, [37](#)

Transcript

- of characterization, [87](#)

Transcript messages, [16](#)

Transcript window, [15](#)

Troubleshooting, [88](#)

— U —

User interface

- command input window, [15](#)
- transcript window, [15](#)

— W —

Warning Messages

- configuration options, [27](#)

Working Directory, [51](#)

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Third-Party Information

This section provides information on third-party software that may be included in the Calypso[®] family of products, including any additional license terms.

- This software application may include GCC 4.2.2 third-party software. GCC 4.2.2 is distributed under the terms of the General Public License version 2 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<install_directory>/Mgc_home/shared/legal/gnu_gpl_2.0.pdf`. To obtain a copy of the source code to the files licensed under the GNU GPL v2, send a request to `request_sourcecode@calypso.com`. This offer shall only be available for three years from the date Calypso Design Systems first distributed GNU GPL v2 covered source code.
- This software application may include gdb version 7.0 third-party software. gdb version 7.0 is distributed under the terms of the GNU General Public License version 2.0 and version 3.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: `<your_Mentor_Graphics_documentation_directory>/legal/gnu_gpl_2.0.pdf` and `<your_Mentor_Graphics_documentation_directory>/legal/gnu_gpl_3.0.pdf`. Portions of this software may be subject to the GNU Free Documentation License version 1.1. You can view a copy of the GNU Free Documentation License version 1.1 at: `<your_Mentor_Graphics_documentation_directory>/legal/gnu_free_doc_1.1.pdf`. Portions of this software may be subject to the GNU Free Documentation License version 1.2. You can view a copy of the GNU Free Documentation License version 1.2 at: `<your_Mentor_Graphics_documentation_directory>/legal/gnu_free_doc_1.2.pdf`. Portions of this software may be subject to the GNU Library General Public License version 2.0. You can view a copy of the GNU Library General Public License version 2.0 at: `<your_Mentor_Graphics_documentation_directory>/legal/gnu_library_gpl_2.0.pdf`. To obtain a copy of the gdb version 7.0 source code, send a request to `request_sourcecode@calypso.com`. This offer shall only be available for three years from the date Calypso Design Systems first distributed gdb version 7.0. gdb version 7.0 may be subject to the following copyrights:

© 1987 Regents of the University of California.
All rights reserved.

Redistribution and use in source and binary forms are permitted provided that the above copyright notice and this paragraph are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by the University of California, Berkeley. The name of the University may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

© 1983, 1990 Regents of the University of California.
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. [rescinded 22 July 1999]
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS

INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

© 1993,1991,1990,1989,1988,1987 Carnegie Mellon University
All Rights Reserved.

Permission to use, copy, modify and distribute this software and its documentation is hereby granted, provided that both the copyright notice and this permission notice appear in all copies of the software, derivative works or modified versions, and any portions thereof, and that both notices appear in supporting documentation.

CARNEGIE MELLON ALLOWS FREE USE OF THIS SOFTWARE IN ITS "AS IS" CONDITION. CARNEGIE MELLON DISCLAIMS ANY LIABILITY OF ANY KIND FOR ANY DAMAGES WHATSOEVER RESULTING FROM THE USE OF THIS SOFTWARE.

Carnegie Mellon requests users of this software to return to

Software Distribution Coordinator or Software.Distribution@CS.CMU.EDU
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-3890

any improvements or extensions that they make and grant Carnegie Mellon the rights to redistribute these changes.

- This software application may include Tcl version 8.5.8 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. Tcl version 8.5.8 may be subject to the following copyrights:

© 1988, 1993, 1994 The Regents of the University of California All rights reserved.
All Rights Reserved

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This software application may include Tcl Readline third-party software that may be subject to the following copyright:

© 1998 - 2000, Johannes Zellner johannes@zellner.org All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following

disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Johannes Zellner nor the names of contributors to this software may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

- This software application may include Tk version 8.5.8 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. Tk version 8.5.8 may be subject to the following copyrights:

© David Koblas

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty.

© 1998 Hutchison Avenue Software Corporation

<http://www.hasc.com>

info@hasc.com

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided "AS IS." The Hutchison Avenue Software Corporation disclaims all warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to this code and accompanying documentation.

© 1985, 1986, 1987, 1989, 1991 by the Massachusetts Institute of Technology

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 1987 by Digital Equipment Corporation, Maynard, Massachusetts, and the Massachusetts Institute of Technology, Cambridge, Massachusetts

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Digital or MIT not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

DIGITAL DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL DIGITAL BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

- This software application may include TKtreectrl version 2.2.8 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied TKtreectrl version 2.2.8 may be subject to the following copyrights:

This software is copyrighted by Tim Baker and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

- This software application may include gmake version 3.81 third-party software. Gmake version 3.81 is distributed under the terms of the General Public License version 2.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: [<install_directory>/Mgc_home/shared/legal/gnu_gpl_2.0.pdf](install_directory/Mgc_home/shared/legal/gnu_gpl_2.0.pdf). Portions of this software may be subject to the GNU Free Documentation License version 1.2. You can view a copy of the GNU Free Documentation License version 1.2 at: [<install_directory>/Mgc_home/shared/legal/gnu_free_doc_1.2.pdf](install_directory/Mgc_home/shared/legal/gnu_free_doc_1.2.pdf). Portions of this software may be subject to the Library General Public License version 2.0. You can view a copy of the Library General Public License version 2.0 at: [<install_directory>/Mgc_home/shared/legal/gnu_library_gpl_2.0.pdf](install_directory/Mgc_home/shared/legal/gnu_library_gpl_2.0.pdf). To obtain a copy of the source code to gmake version 3.81, send a request to request_sourcecode@calypto.com. This offer shall only be available for three years from the date Calypto Design Systems first distributed gmake version 3.81.
- This software application may include SystemC third-party software.

©2001 Dr. John Maddock. All rights reserved.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Dr. John Maddock makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

©1994 Hewlett-Packard Company. All rights reserved.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

©1996 Silicon Graphics Computer Systems, Inc. All rights reserved.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

- This software may include Info-Zip Unzip third-party software.

Copyright (c) 1990-2005 Info-ZIP. All rights reserved.

For the purposes of this copyright and license, "Info-ZIP" is defined as the following set of individuals:

Mark Adler, John Bush, Karl Davis, Harald Denker, Jean-Michel Dubois, Jean-loup Gailly, Hunter Goatley, Ed Gordon, Ian Gorman, Chris Herborth, Dirk Haase, Greg Hartwig, Robert Heath, Jonathan Hudson, Paul Kienitz, David Kirschbaum, Johnny Lee, Onno van der Linden, Igor Mandrichenko, Steve P. Miller, Sergio Monesi, Keith Owens, George Petrov, Greg Roelofs, Kai Uwe Rommel, Steve Salisbury, Dave Smith, Steven M. Schweda, Christian Spieler, Cosmin Truta, Antoine Verheijen, Paul von Behren, Rich Wales, Mike White

This software is provided "as is," without warranty of any kind, express or implied. In no event shall Info-ZIP or its contributors be held liable for any direct, indirect, incidental, special or consequential damages arising out of the use of or inability to use this software.

- This software application may include CUDD third-party software that may be subject to the following copyrights:

© 1995-2004, Regents of the University of Colorado. All rights reserved.

© 1985 by Digital Equipment Corporation, Maynard, MA. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the University of Colorado nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The information in this software is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation.

no responsibility for the use or reliability of its software on equipment which is not supplied by Digital.

Redistribution and use in source and binary forms are permitted provided that the above copyright notice and this paragraph are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by Digital Equipment Corporation. The name of Digital Equipment Corporation may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Do not take internally. In case of accidental ingestion, contact your physician immediately.

- This software application may include getline third-party software that is distributed by Chris Thewalt and may be subject to the following copyrights

©1991, 1992, 1993 by Chris Thewalt (thewalt@ce.berkeley.edu)

- This software application may include Mersenne Twister third-party software. Mersenne Twister is distributed under the terms of the Mersenne Twister License Agreement. You can view the complete license at: install_directory/Mgc_home/shared/legal/mersenne_twister.pdf.

Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License.

- This software application may include SystemC version 2.2 third-party software. To obtain a copy of the SystemC source code, send a request to request_sourcecode@calypto.com. SystemC software is distributed under the SystemC Open Source License Agreement (Download, Use and Contribution License Agreement Version 3.0) and is distributed on an

"AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. You can view a copy of the license at: <install_directory>/Mgc_home/shared/legal/systemc_open_source_3.0.pdf. Portions of this software are subject to the Boost License v.1.0. You can view a copy of the license at: <install_directory>/Mgc_home/shared/legal/boost_1.0.pdf. SystemC version 2.2 may be subject to the following copyrights:

Copyright (c) 1994 Hewlett-Packard Company

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Copyright (c) 1996 Silicon Graphics Computer Systems, Inc.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Copyright 1991 by the Massachusetts Institute of Technology

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

Calling this script install-sh is preferred over install.sh, to prevent `make' implicit rules from creating a file called install from it when there is no Makefile.

This script is compatible with the BSD install script, but was written from scratch. It can only install one file at a time, a restriction shared with many OS's install programs.

Copyright (c) 1993 by David Keppel

Permission to use, copy, modify and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice and this notice appear in all copies. This software is provided as a proof-of-concept and for demonstration purposes; there is no representation about the suitability of this software for any purpose.

- This software application may include libxml2 version 2.6.31 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. libxml2 version 2.6.31 may be subject to the following copyrights:

Copyright (C) 1998-2003 Daniel Veillard. All Rights Reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE DANIEL VEILLARD BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of Daniel Veillard shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from him.

Copyright (C) 2000 Bjorn Reese and Daniel Veillard.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

© 1991 by the Massachusetts Institute of Technology

Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 2000 Gary Pennington and Daniel Veillard.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

© 1998, 2000, 2001 Bjorn Reese and Daniel Stenberg.

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

© 2001 Bjorn Reese <breese@users.sourceforge.net>

Permission to use, copy, modify, and distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THIS SOFTWARE IS PROVIDED ``AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHORS AND CONTRIBUTORS ACCEPT NO RESPONSIBILITY IN ANY CONCEIVABLE MANNER.

- This software application may include TLM version 2.0 third-party software. TLM version 2.0 is distributed under the terms of the **SystemC Open Source License Agreement v3.0** and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: <install_directory>/Mgc_home/shared/legal/systemc_open_source_3.0.pdf. To obtain a copy of the TLM version 2.0 source code, send a request to request_sourcecode@calypto.com.
- This software application may include tbcload version 1.7 third-party software, which is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied.

- This software application may include portions of Boost Spirit version 1.8.5 third-party software. Boost Spirit version 1.8.5 is distributed under the terms of the Boost Software License version 1.0 and is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the license for the specific language governing rights and limitations under the license. You can view a copy of the license at: [<install_directory>/Mgc_home/shared/legal/boost_1.0.pdf](install_directory/Mgc_home/shared/legal/boost_1.0.pdf). Boost Spirit version 1.8.5 may be subject to the following copyrights:

© 1996, 1997 Silicon Graphics Computer Systems, Inc.

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Silicon Graphics makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

© 1994 Hewlett-Packard Company

Permission to use, copy, modify, distribute and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. Hewlett-Packard Company makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty .