Verification Continuum™

Synopsys
Synplify Pro for Microchip Command Reference Manual

June 2020

SYNOPSYS®

Synopsys Confidential Information

Copyright Notice and Proprietary Information

© 2020 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Free and Open-Source Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at
http://www.synopsys.com/Company/Pages/Trademarks.aspx.
All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 East Middlefield Road
Mountain View, CA 94043
www.synopsys.com

June 2020

© 2020 Synopsys, Inc.

Contents

Chapter 1: Overview of the Synthesis Commands

About Tcl Commands 14

About the GUI Commands 16

Chapter 2: Tcl Synthesis Commands

add_file 23
add_folder 27
analyst 28
cdpl_queue 31
check_fdc_query 32
command_history 36
constraint_file 37
create_fdc_template 38
design 40
dump_metrics 50
encryptIP 53
encryptP1735 56
IEEE 1735 Encryption Use Models 59
get_env 66
get_option 66
hdl_define 67
hdl_param 68
help 70
history 71
impl 72
job 74
log_filter 75
log_report 77
message_override 78
open_design 80
open_file 81
partdata 82

Synopsys Confidential Information
© 2020 Synopsys, Inc.

program_terminate 83
program_version 84
proj ect 85
project_data 93
project_file 94
project_folder 96
query_available_metrics 97
query_metric 100
query_metric_details 102
recording 104
report_clocks 105
report_messages 106
report_message_summary 108
run_tcl 109
select 110
sdc2fdc 111
set_opt ion 113
status_report 136
sub_impl 140
synplify_pro 141
Tcl Command Categories 144

Chapter 3: Tcl Find, Expand, and Collection Commands

find 147
Tcl Find Syntax 148
Tcl Find Syntax Examples 151

find -filter 154

expand 161

Collection Commands 164
c_diff 165
c_info 166
c_intersect 166
c_list 167
c_print 168
c_symdiff 169
c_union 170
define_collection 171
define_scope_collection 171
get_prop 171
set 172

Query Commands 174
all_clocks 177
all_fanin 177
all_fanout 179
all_inputs 181
all_outputs 182
all_registers 182
get_cells 184
get_clocks 186
get_flat_cells 188
get_flat_nets 190
get_flat_pins 192
get_nets 194
get_pins 195
get_ports 199
object_list 200
report_timing 201

Synopsys Standard Collection Commands 204
add_to_collection 204
append_to_collection 206
copy_collection 207
foreach_in_collection 208
get_object_name 210
index_collection 210
remove_from_collection 212
sizeof_collection 213

Chapter 4: Constraint Commands

SCOPE Constraints Editor 216

SCOPE Tabs 217
Clocks 218
Generated Clocks 223
Collections 225
Inputs/Outputs 228
Registers 231
Delay Paths 232
Attributes 235
I/O Standards 236
Compile Points 237
TCL View 240

Industry I/O Standards 242

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Industry I/O Standards 242
Delay Path Timing Exceptions 245
Multicycle Paths 245
False Paths 248
Specifying From, To, and Through Points 250
Timing Exceptions Object Types 250
From/To Points 250
Through Points 252
Product of Sums Interface 253
Clocks as From/To Points 256
Conflict Resolution for Timing Exceptions 258
Timing Constraints 262
create_clock 263
create_generated_clock 265
reset_path 269
set_clock_groups 271
set_clock_latency 277
set_clock_uncertainty 279
set_false_path 281
set_input_delay 284
set_max_delay 287
set_multicycle_path 290
set_output_delay 294
set_reg_input_delay 297
set_reg_output_delay 298
Naming Rule Syntax Commands 298
Design Constraints 301
define_compile_point 302
define_current_design 303
define_io_standard 304

Chapter 5: User Interface Commands

File Menu 306
New Command 307
Create Image Command 308
Build Project Command 310
Open Project Command 311
Edit Menu 311
Find Command (Text) 314

Find Command (In Project) 315
Find Command (HDL Analyst) 317
Find in Files Command 321
Replace Command 323
Goto Command 324

View Menu 325
Toolbar Command 328
View Sheets Command 329
View Log File Command 330

Project Menu 333
Add Source File Command 334
Remove Implementation 336
Change File Command 337
Set VHDL Library Command 337
Add Implementation Command 338
Archive Project Command 339
Un-Archive Project Command 340
Copy Project Command 343

Implementation Options Command 346
Device Panel 347
Options Panel 349
Constraints Panel 351
Implementation Results Panel 353
Timing Report Panel 355
High Reliability Panel 357
VHDL Panel 358
Verilog Panel 362
Compiler Directives and Design Parameters 369
Place and Route Panel 379

Run Menu 381
Run Tcl Script Command 384
Run Implementations Setup Command 385
Job Status Command 387
Identify Instrumentor Command 387
Launch Identify Debugger Command 389
Launch SYNCore Command 389
Configure and Launch VCS Simulator Command 389

Analysis Menu 400
Timing Report Generation Parameters 401

HDL Analyst Menu 412
HDL Analyst Menu->RTL and Technology View Submenus 412
HDL Analyst Menu: Hierarchical and Current Level Submenus 413
HDL Analyst Menu: Filtering and Flattening Commands 415
HDL Analyst Menu: Timing Commands 419
HDL Analyst Menu: Analysis Commands 419
HDL Analyst Menu: Selection Commands 423
HDL Analyst Menu: FSM Commands 423

Options Menu 424
Configure Parallel or Compile Point Process Command 425
Project View Options Command 429
Editor Options Command 436
Place and Route Environment Options Command 439
Configure 3rd Party Tools Options Command 440
Project Status Page Location 441
HDL Analyst Options Command 443
Standard HDL Analyst Options Command 444
Configure External Programs Command 453

Web Menu 454

Help Menu 455
Preferred License Selection Command 456
Tip of the Day Command 457

Chapter 6: GUI Popup Menu Commands

Popup Menus 460
Watch Window Popup Menu 460
Tcl Window Popup Menu 461
Text Editor Popup Menu 461
Log File Popup Menu 461
FSM Viewer Popup Menu 463

Project View Popup Menus 466
Project Management View Popup Folder Commands 470
Vendor Tool Invocation Popup Menu Command 472
File Options Popup Menu Command 473
Copy File Popup Menu Command 475
Change Implementation Popup Menu Commands 476
Show Compile Points Popup Menu Command 476
Project Options Popup Menu Command 477
Add P&R Implementation Popup Menu Command 478
Options for Place & Route Jobs Popup Menu Command 480

RTL and Technology Views Popup Menus 483

© 2020 Synopsys, Inc.
Synopsys Confidential Information

SYNOPSYS®
Silicon to Software™

CHAPTER 1

Overview of the Synthesis Commands

This document is part of a set that includes reference and procedural information for the Synopsys® FPGA tools. This document describes the commands available for the synthesis tools, which usually includes a graphical user interface (GUI) as well as command line access. Commands may vary with the capabilities of the synthesis tool.

The following sections provide an overview of the commands in the tool:

- About Tcl Commands, on page 14
- About the GUI Commands, on page 16

© 2020 Synopsys, Inc.

About Tcl Commands

Tcl (Tool Command Language) is a popular scripting language for controlling software applications. Synopsys has extended the Tcl command set with additional commands that you can use to run the Synopsys FPGA programs. These commands are not intended for use in controlling interactive debugging, but you can use them to run synthesis multiple times with alternate options to try different technologies, timing goals, or constraints on a design.

Tcl scripts are text files that have a .tcl file extension and contain a set of Tcl commands designed to complete a task or set of tasks. You can also run Tcl scripts through the Tcl window (see *Tcl Script Window*, on page 41).

The Synopsys FPGA Tcl commands are described here. For information on the standard Tcl commands, syntax, language, and conventions, refer to the Tcl online help (Help->TCL).

Tcl Conventions

Here is a list of conventions to respect when entering Tcl commands and/or creating Tcl scripts.

- Tcl is case sensitive.
- Comments begin with a hash mark or pound sign (#).
- Enclose all path names and filenames in double quotes (").
- Use a forward slash (/) as the separator between directory and path names (even on the Microsoft® Windows® operating system). For example:

```
designs/big_design/test.v
```

Tcl Commands

You can enter the Tcl (Tool Command Language) commands directly in the Tcl window, or include them in Tcl scripts that you can run in batch mode. For more information about Tcl commands, see *Tcl Synthesis Commands*, on page 21.

© 2020 Synopsys, Inc.

Tcl Scripts and Batch Mode

For procedures for creating Tcl scripts and using batch mode, see *Working with Tcl Scripts and Commands*, on page 512 in the *User Guide*:

- Running Batch Mode on a Project File, on page 506
- Running Batch Mode with a Tcl Script, on page 507
- Generating a Job Script, on page 513
- Creating a Tcl Synthesis Script, on page 515
- Using Tcl Variables to Try Different Clock Frequencies, on page 516
- Running Bottom-up Synthesis with a Script, on page 519

© 2020 Synopsys, Inc.
Synopsys Confidential Information

About the GUI Commands

The GUI commands are accessed from the software graphical interface. Most commands open dialog boxes where you can specify parameters for the command.

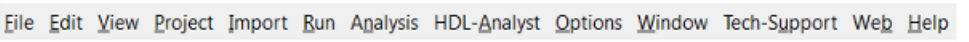
The GUI provides a few ways to access commands:

- Menus, on page 16
- Context-sensitive Popup Menus, on page 17
- Toolbars, on page 17
- Keyboard Shortcuts, on page 17
- Buttons and Options, on page 17
- Tcl Commands, on page 14

Most commands have GUI and command line versions, so you can use either method to specify commands.

Menus

The set of commands on the pull-down menus in the menu bar varies depending on the view, design status, task to perform, and selected object(s). For example, the File menu commands in the Project view differ slightly from those in the RTL view. Menu commands that are not available for the current context are dimmed out. The menu bar in the Project view is shown below:



The individual menus, their commands, and the associated dialog boxes are described in the following sections:

- File Menu, on page 306
- Edit Menu, on page 311
- View Menu, on page 325
- Project Menu, on page 333
- Run Menu, on page 381

- Analysis Menu, on page 400
- HDL Analyst Menu, on page 412
- Options Menu, on page 424
- Web Menu, on page 454
- Help Menu, on page 455

Context-sensitive Popup Menus

Popup menus, available by right-clicking, offer access to commonly used commands that are specific to the current context. See *Popup Menus*, on page 460, *Project View Popup Menus*, on page 466, and *RTL and Technology Views Popup Menus*, on page 483 for information on individual popup menus.

Toolbars

Toolbars contain icons associated with commonly used commands. For more information about toolbars, see *Toolbars*, on page 57.

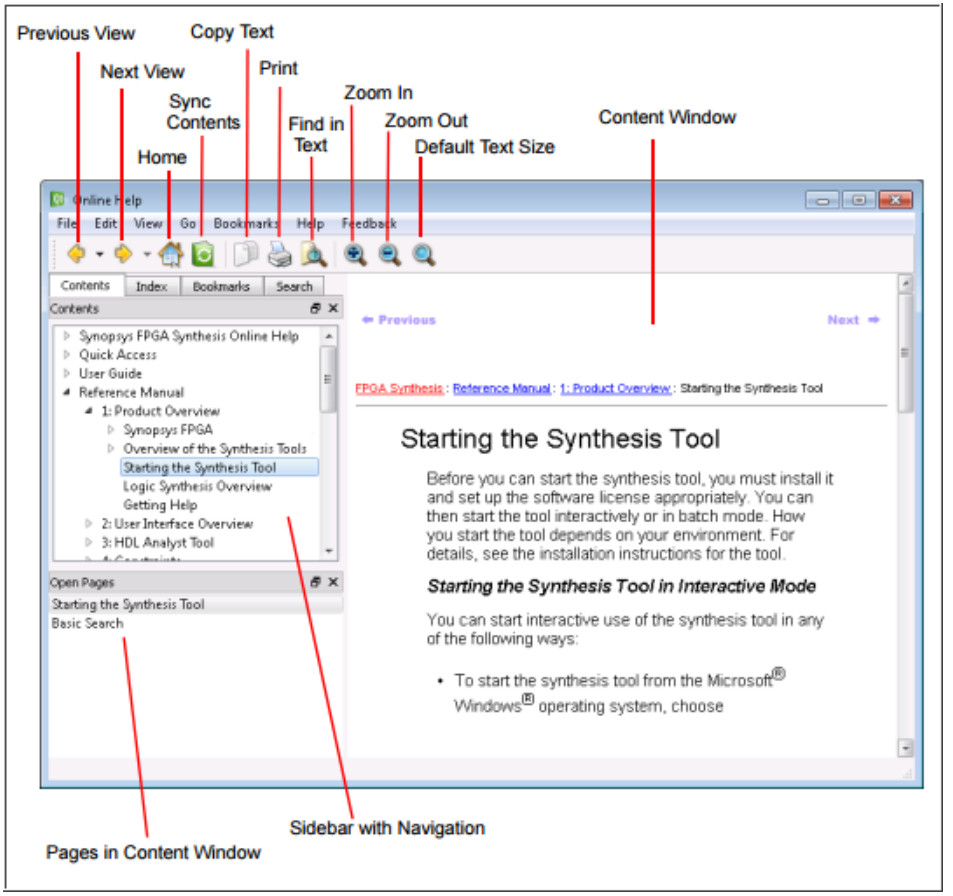
Keyboard Shortcuts

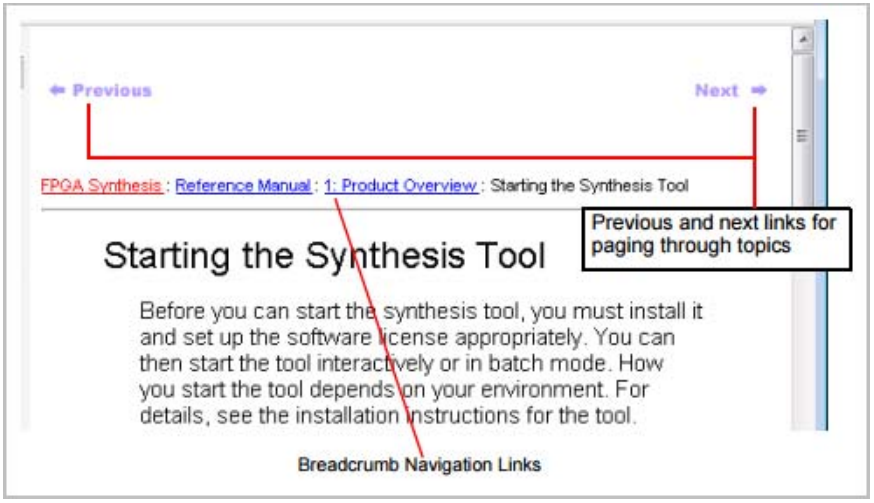
Keyboard shortcuts are available for commonly used commands. The shortcut appears next to the command in the menu. See *Keyboard Shortcuts*, on page 64 for details.

Buttons and Options

The Project view has buttons for quick access to commonly used commands and options. See *Buttons and Options*, on page 72 for details.

© 2020 Synopsys, Inc.
Synopsys Confidential Information


Previous View
Copy Text
Next View
Print
Sync Contents
Zoom In
Find in Text
Zoom Out
Content Window
Home
Default Text Size
Online Help
File Edit View Go Bookmarks Help Feedback
Contents Index Bookmarks Search
Contents
Synopsys FPGA Synthesis Online Help
Quick Access
User Guide
Reference Manual
1: Product Overview
Synopsys FPGA
Overview of the Synthesis Tools
Starting the Synthesis Tool
Logic Synthesis Overview
Getting Help
2: User Interface Overview
3: HDL Analyst Tool
Open Pages
Starting the Synthesis Tool
Basic Search
Previous
Next
FPGA Synthesis : Reference Manual : 1: Product Overview : Starting the Synthesis Tool
Starting the Synthesis Tool
Before you can start the synthesis tool, you must install it and set up the software license appropriately. You can then start the tool interactively or in batch mode. How you start the tool depends on your environment. For details, see the installation instructions for the tool.
Starting the Synthesis Tool in Interactive Mode
You can start interactive use of the synthesis tool in any of the following ways:
To start the synthesis tool from the Microsoft® Windows® operating system, choose
Pages in Content Window
Sidebar with Navigation



© 2020 Synopsys, Inc.

SYNOPSYS®
Silicon to Software™

CHAPTER 2

Tcl Synthesis Commands

This chapter describes supported Tcl commands. The synthesis commands are listed in alphabetical order.

add_file	add_folder
analyst	cdpl_queue
check_fdc_query	command_history
constraint_file	create_fdc_template
design	dump_metrics
encryptIP	encryptP1735
get_env	get_env
get_option	help
hdl_define	hdl_param
history	impl
job	log_filter
log_report	message_override
open_design	open_file
partdata	program_terminate
program_version	project
project_data	project_file

© 2020 Synopsys, Inc.

project_folder	query_available_metrics
query_metric	query_metric_details
recording	report_clocks
report_messages	report_message_summary
run_tcl	sdc2fdc
select	set_option
status_report	synplify_pro

See also:

- For specific categories of synthesis commands (for example, log file commands), see Tcl Command Categories, on page 144.
- For a description of the find, expand, and collection commands, see *Tcl Find, Expand, and Collection Commands*, on page 145.
- For the TCL timing and design constraints syntax and their descriptions in SCOPE, see *Constraint Commands*, on page 215.

add_file

The add_file command adds one or more files to a project.

Syntax

add_file [*-filetype*] *fileName* [*fileName* [...]]
add_file -verilog [**-lib** *fileName* [*fileName* [...]]] [**-folder** *folderName*]
add_file -vhdl [**-lib** *libName*[*libName*]] *fileName* [*fileName* [...]] [**-folder** *folderName*]
add_file -include *fileName* [*fileName* [...]]
add_file [*-filetype*] **-job_owner par** | **simulation** [*fileName* [...]]
add_file -structver [*fileName* [...]]
add_file -tooltag *tooltagName* **-toolargs** [*toolArguments*] *fileName*
add_file -vlog_std *standard fileName* [*fileName* [...]]

-filetype	Specifies the type of file being added to the project (files are placed in folders according to their file types; including this argument overrides automatic filename-extension placement). See *Filename Extensions* , on page 25 for a list of the recognized file types.
fileName	Specifies the name of the file being added to the project. Files are added to the individual project folders according to their filename extensions (View Project Files in Folders must be set in the Project View Options dialog box). You can add multiple files by separating individual filenames with a space, and you can specify different file types (extensions) within the same command.
-verilog or -vhdl	Adds HDL files with non-standard extensions to the Verilog or VHDL directory, so that they can be compiled with the project. For example, the following command adds the file alu.v.new to the project's verilog directory: `% add_file -verilog /designs/megachip/alu.v.new` If you do not specify -verilog, the file is added to the Other directory (new is not a recognized Verilog extension), and the file would not be compiled with the files in the Verilog directory.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

[-lib *libName*]	Specifies the library associated with Verilog or VHDL files. The default library is work. The -lib option sets the VHDL library to *libName*.
	Note: You can also specify multiple libraries for Verilog or VHDL files.
	Verilog Example:
	`add_file -verilog -lib top -vlog_std sysv "top.v"`
	VHDL Example:
	`add_file -vhdl -lib {mylib,work} "ff.vhd"`
	Both the logical and physical libraries must be specified in the Project file (if you only specify the logical library associated with the Verilog or VHDL files, the compiler treats the module as a black box).

[-folder *folderName*]	Creates logical folders with custom files in various hierarchy groupings within your Project view. For example:
	`add_file -verilog -folder memory "ram_1.v"`
	`add_file -verilog -folder memory "C:/examples/verilog/common_rtl/memory/ram_1.v"`

-include	Indicates that the specified file is to be added to the project as an include file (include files are added to the Include directory regardless of their extension). Include files are not passed to the compiler, but are assumed to be referenced from within the HDL source code. Adding an include file to a project, although not required, allows it to be accessed in the user interface where it can be viewed, edited, or cross-probed.
-job_owner	Allows you to determine how files are used; you can specify these options from the File Options dialog box. For example, you can automatically decide to pass files to the back-end place-and-route tool (Use for Place and Route) or use them for test benches containing HDL constructs for simulation (Use for Simulation only).
-structver *fileName*	Adds structural Verilog files as input for your design project. The software performs fast compilation of the structural Verilog files, providing runtime improvements for the design. For example: `add_file -structver` *fileName*`.vm` For more information, see *Using the Structural Verilog Flow* , on page 53.

-tooltag *tooltagName*	Creates a tool tag name for the application tool you want to invoke from within the Synopsys FPGA synthesis tool.
-toolargs tool	Specifies any argument options to use with the application tool you want to invoke from within the Synopsys FPGA synthesis tools. For example: `add_file -tooltag {EDK} -toolargs {$SynCode} "ram.v"`
-vlog_std *standard*	Overrides the global Verilog standard for an individual file. The accepted values for *standard* are v95 (Verilog 95), v2001 (Verilog 2001), and sysv (SystemVerilog). The file (*fileName*) is added to the Verilog folder in the project; the specified standard is listed after the filename in the project view and is enclosed in angle brackets (for example, commchip.v <sysv>). Note that when you add a SystemVerilog file (a file with an sv extension) to a project, the add_file entry in the project file includes the -vlog_std *standard* string. The default standard for new projects is SystemVerilog. For Verilog 2005 extensions, use sysv (SystemVerilog).

Filename Extensions

Files with the following extensions are automatically added to their corresponding project directories; files with any other extension are added to the Other directory. The *-filetype* argument overrides automatic filename extension placement.

Extension	-Filetype	Project Folder
.adc	-analysis_constraint	Analysis Design Constraint
.edf, .edn	-edif	EDIF
.fdc	-fpga_constraint/-constraint	Logic Constraints (FDC)
.sdc	-constraint	Logic Constraints (SDC)
.sv[1]	-verilog	Verilog
.tcl	-tcl	Tcl Script
.v	-verilog	Verilog

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Extension	-Filetype	Project Folder
.vhd, .vhdl	-vhdl	VHDL
.vm	-structver	Structural Verilog File
any	-include	Include

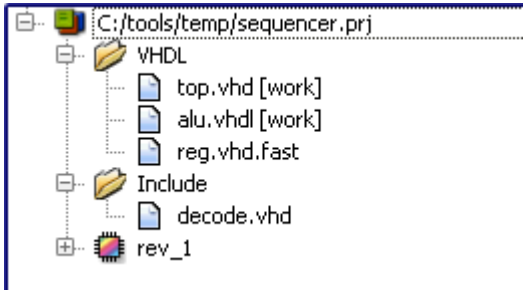
1. Use the sv format for SystemVerilog keyword support. Both Verilog and SystemVerilog formats are added to the Verilog folder.

Example: Add Files

Add a series of VHDL files to the VHDL directory and add an include file to the project:

```
% add_file /designs/sequencer/top.vhd
% add_file /designs/sequencer/alu.vhdl
% add_file -vhdl /designs/sequencer/reg.vhd.fast
% add_file -include /designs/std/decode.vhd
```

The corresponding directory structure in the Project view is shown in the following figure:



Example: File Options Designation

Designate some IP core wrappers as well as their associated instantiated component files that must be passed on to the place-and-route tool, since they are not written to the final netlist:

```
add_file -verilog -job_owner par "my_ip_core.v"
add_file -verilog -job_owner par "my_ip_core_enc.v"
```

add_folder

The add_folder command adds a custom folder to a project.

Syntax

add_folder *folderName*

Creates logical folders with files in various custom hierarchy groupings within your Project view. These custom folders can be specified with any name or hierarchy level.

```
add_folder verilog
```

```
add_folder verilog/common_rtl
```

```
add_folder verilog/common_rtl/prep
```

For more information about custom folders, see *Managing Project File Hierarchy*, on page 70 in the *User Guide*.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

analyst

Changes and manipulates the schematic views of the netlist. Note that the following HDL Analyst commands can be used without the analyst prefix as well.

analyst clone_view	analyst critical_path	analyst dissolve
analyst filter	analyst flatten	analyst get_selected
analyst group	analyst pop	analyst push
analyst select	analyst unfilter	analyst view

analyst clone_view

Opens a copy of the current view.

analyst clone_view *designID*

designID
The design ID to clone. If not specified, clones the current view.

analyst critical_path

Filters the view to show the instances that are part of the critical path, if available.

analyst critical_path

analyst dissolve

Removes targeted hierarchies from the design. Contents of the hierarchy are put into the level that originally contained the hierarchy.

analyst dissolve *collection*

collection
Collection of instances to dissolve.

analyst filter

Filters the view by selected instances and ports.

analyst filter

analyst flatten

Flattens the current view.

analyst flatten

analyst get_selected

Returns the names of currently selected objects.

analyst get_selected [-inst] [-net] [-port] [-pin]

-inst
Returns the names of selected instances. If no -*type* option is set, the names of all selected objects are returned.

-net
Returns the names of selected nets. If no -type option is set, the names of all selected objects are returned.

-port
Returns the names of selected ports. If no -*type* option is set, the names of all selected objects are returned.

-pin
Returns the names of selected pins. If no -*type* option is set, the names of all selected objects are returned.

analyst group

Creates a graphical group of instances.

analyst group [*collection*] [**-name** *groupName*]

collection
Instances to group. All instances must be on the same level of the hierarchy.

-name *groupName*
The name of the graphical group to be created.

analyst pop

Pops up the hierarchy.

analyst pop

© 2020 Synopsys, Inc.
Synopsys Confidential Information

analyst push

Pushes down the hierarchy.

analyst push *hierarchyName*

hierarchyName
The name of the group or instance to push the hierarchy down into.

analyst select

Selects specified objects.

analyst select [*collection*] [**-append**] [**-clear**] [**-instances**] [**-primitives**]

collection
The ID of the collection to select.

-append
Appends objects to the selection list.

-clear
Clears the selection list.

-instances
Selects all instances in the current view.

-primitives
Selects all primitives (leaf) instances in the current view.

analyst unfilter

Unfilters the view.

analyst unfilter

analyst view

Opens a schematic view.

analyst view *designID*

designID
The design ID to view.

cdpl_queue

Use the cdpl_queue command to set a CDPL queue, get the configuration setting of the CDPL queue, or clear the CDPL queue settings. These settings are valid only for the current session, and not saved in the .ini file or any other script. Create a Tcl file to save the settings and use it for subsequent sessions. This command is available only as a Tcl syntax.

Syntax

cdpl_queue -set [*queueType*][*queueConfig*] **| -get** [*queueType*] **| -clear**

The following table describes the cdpl_queue command options:

Option	Description
-set [**queueType**] [**queueConfig**]	Sets a CDPL queue configuration. • queueType is lowmem, highmem, default or vendor. The queue type vendor is for all place and route jobs. • queueConfig is the queue host file. See CDPL help documentation on how to set up a queue host file.
-get [**queueType**]	Returns the configuration setting of the queueType specified. If queueType is not given, the command returns all the queue types set in the current session.
-clear	Clears all the queue settings.

Examples

To use this command, set up the following string in your project file:

```
cdpl_queue -set default $PWD/hosts_default
```

For more information, see the CDPL (Common Distributed Processing Library) documentation.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

check_fdc_query

Runs the constraint checker for constraints using the get_* and/or all_* query commands specified in the timing constraint file for the project.

Syntax

check_fdc_query [**-full_check**]

Arguments and Options

-full_check

Runs the full constraint checker before checking the query commands. The default is to run the check_fdc_query command without this option.

When the -full_check option is *not* specified, the command only runs the constraint syntax checker, which reduces runtime significantly, since most objects being searched are found in pre-mapping and do not require full mapping to be run. However, this option does not find bit-blasted registers and objects using the advanced -filter @property =~ commands, where the property is created or applied during mapping because it requires optimizations such as register replication.

For example, if a 4-bit RAM output is targeted with the get_cell command, the differences in the results are shown below:

Command	Run Stage	Results
Default (without -full_check)	Pre-mapping	ram_out [3:0]
With -full_check	Mapping	ram_out [3] ram_out [2] ram_out [1] ram_out [0]

Description

The check_fdc_query command reads the .fdc constraint file of the current project file. It runs the constraint checker for the following object query commands that are used with FDC constraints:

all_* Commands	get_* Commands
all_clocks	get_cells
all_fanin	get_clocks
all_fanout	get_nets
all_inputs	get_pins
all_outputs	get_ports
all_registers	

The report provides feedback on how these query commands are applied and ensures that the commands are used properly with constraints in the constraint file.

Collections created with define_scope_collection, find, and expand are not covered by this Tcl command. You can check these SCOPE collections in the HDL Analyst and the SCOPE interface. The report does not cover the define_io_standard constraint either.

Example

Invoke check_fdc_query from the Tcl command line for the project. You can also invoke it from a shell window.

The command writes out the results of the object query commands to the *projectName*_cck_fdc.rpt file that opens in the GUI. You may need to run the constraint checker (Run->Constraint Check) to find additional issues with constraints.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

The following example shows the results of running the constraint checker in the *projectName*_cck_fdc.rpt file.

```
FDC query commands results
**************************
################################
# 1019 : set_multicycle_path 2 -from [get_cells -hier {*[4]}]
# line 175 in  :
C:/check_fdc_query/all_clocks/test1_basic/top_translated.fdc
Results of query command: get_cells -hier {*[4]}
   (none)
################################
# 1027 : set_multicycle_path 3 -to [all_clocks]
# line 196 in  :
C:/check_fdc_query/all_clocks/test1_basic/top_translated.fdc
Results of query command: all_clocks
   clka
   clkb
   dcm|CLK0_BUF_clock_CLKIN1
   dcm|clk0_i_clock_CLKIN1
   dcm|CLK0_BUF_1_clock_CLKIN1
```

The syntax checker reports the object query commands and any issues it found and writes them to the *projectName*_scck.rpt file.

```
# Synopsys Constraint Checker (syntax only), version map610dev,
Build 1085R
# Copyright (C) 1994-2016, Synopsys, Inc.
# Written on Tue Apr 30 15:39:07 2013
##### DESIGN INFO
#########################################################
Top View:                "top"
Constraint File(s):
"C:\check_fdc_query\all_clocks\test1_basic\top_translated.fdc"

"C:\builds\syn201309_063R\lib\fdc_query.fdc"

# Run constraint checker to find more issues with constraints.
#########################################################################
#######

No issues found in constraint syntax.
```

```
Clock Summary
**************
Start
Requested     Requested     Clock                          Clock

Clock    Frequency Period Type Group
-------------------------------------------------------------------
clka     100.0 MHz 10.000 declare default_clkgroup
clkb     50.0 MHz  20.000 declared default_clkgroup
dcm|CLK0_BUF _clock_CLKIN1
         200.0 MHz 5.000 derived (from clka) default_clkgroup
dcm|CLK0_BUF_1 _clock_CLKIN1
         50.0 MHz  20.000 derived (from clka) default_clkgroup
===================================================================
```

See Also

- *Constraint Checking*, on page 138
- *Constraint Checking Report*, on page 173

© 2020 Synopsys, Inc.
Synopsys Confidential Information

command_history

Displays a list of the Tcl commands executed during the current session.

Syntax

command_history [**-save** *filename*]

Arguments and Options

-save
Writes the list of Tcl commands to the specified *filename*.

Description

The command_history command displays a list of the Tcl commands executed during the current session. Including the -save option, saves the commands to the specified file to create Tcl scripts.

Examples

```
command_history -save C:/DesignsII/tut/proto/myTclScript.tcl
```

See Also

- *recording*, on page 104

constraint_file

The constraint_file command manipulates the constraint files used by the active implementation.

Syntax

constraint_file
-enable *constraintFileName*
-disable *constraintFileName*
-list
-all
-clear

The following table describes the command arguments.

Option	Description
-enable	Selects the specified constraint file to use for the active implementation.
-disable	Excludes the specified constraint file from being used for the active implementation
-list	Lists the constraint files used by the active implementation
-all	Selects (includes) all the project constraint files for the active implementation.
-clear	Clears (excludes) all the constraint files for the active implementation

Examples

List all constraint files added to a project, then disable one of these files for the next synthesis run.

```
% constraint_file -list
attributes.fdc clocks1.fdc clocks2.fdc eight_bit_uc.fdc
% constraint_file -disable eight_bit_uc.fdc
```

Disable all constraint files previously enabled for the project, then enable only one of them for the next synthesis run.

```
% constraint_file -clear
% constraint_file -enable clocks2.fdc
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

create_fdc_template

Lets you create an initial constraint file (.fdc) for your specific design.

Syntax

create_fdc_template [**-period** *float*] [**-in_delay** *float*] [**-out_delay** *float*]

The following table describes the create_fdc_template command options.

Option	Description
-period *float*	Specifies the default values for port clocks.
-in_delay *float*	Specifies the default values for the input delay ports.
-out_delay *float*	Specifies the default values for the output delay ports.

Examples

Each port clock includes a set_clock_groups header with details shown below, which can help you determine whether clocks have been optimized away or if there are any derived clocks.

```
################################################################################
### Individual "set_clock_groups" commands for all "clka" derived clocks
### appear at the end of this file. Enabling a given command will make the
### given clock asynchronous to all other clocks. If a given clock (below)
### does not appear in the final Performance Summary (in the *.srr
### file after synthesis), the clock may have been optimized away due to
### Gated/Generated Clock Conversion.
### See the "CLOCK OPTIMIZATION REPORT" in the *.srr file.
### Below is a list of any clocks derived from "clka":
###  clka DERIVED CLOCKS:
###       dcm|CLK0_BUF_1_derived_clock_CLKIN Clock Object: {t:dcm_inst.CLK_BUF1.O}
###       dcm|CLK0_BUF_derived_clock_CLKIN1    Clock Object: {t:dcm_inst.CLK_BUF0.O}
################################################################################


set_clock_groups -disable -asynchronous -name {clka_group}
   -group {clka} -comment {Source clock clka group}

set_clock_groups -disable -asynchronous
   -name {dcm|CLK0_BUF_1_derived_clock_CLKIN1_group}
   -group [get_clocks
   -of_objects [get_pins {t:dcm_inst.CLK_BUF1.O}]]
   -comment {Derived clock dcm|CLK0_BUF_1_derived_clock_CLKIN1
      from source clock clka}
```

```
set_clock_groups -disable -asynchronous
   -name {dcm|CLK0_BUF_derived_clock_CLKIN1_group}
   -group [get_clocks
   -of_objects [get_pins {t:dcm_inst.CLK_BUF0.O}]]
   -comment {Derived clock dcm|CLK0_BUF_derived_clock_CLKIN1
      from source clock clka}

set_clock_groups -disable -asynchronous -name {clkb_group}
   -group {clkb} -comment {Source clock clkb group}
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

design

Returns netlist data representing information about the design. Commands are available in both batch and GUI mode. Note that the following HDL Analyst find commands can be used without the design prefix as well.

design c_diff	design c_filter	design c_info
design c_intersect	design c_list	design c_print
design c_symdiff	design c_union	design close
design expand	design find	design get_prop
design get	design list	design open
design set	design top_level	

design c_diff

Returns a new find collection containing the differences between two existing find collections.

Syntax

design c_diff *collection1 collection2*

collection1
The first collection to compare.

collecton2
The second collection to compare.

design c_filter

Filters a find collection based on set properties.

Syntax

design c_filter *collection pattern* [**-inst**] [**-net**] [**-port**] [**-pin**] [**-view**]

collection
The collection ID to filter.

pattern
Statement used to filter.

-inst
Returns matching instances. If no -type option (**-inst**, **-net**, **-port**, or **-pin**) is set, all types will be returned.

-net
Returns matching nets. If no -type option (**-inst**, **-net**, **-port**, or **-pin**) is set, all types will be returned.

-port
Returns matching ports. If no -type option (**-inst**, **-net**, **-port**, or **-pin**) is set, all types will be returned.

-pin
Returns matching pins. If no -type option (**-inst**, **-net**, **-port**, or **-pin**) is set, all types will be returned.

-view
Returns matching views. If no -type option (**-inst**, **-net**, **-port**, or **-pin**) is set, all types will be returned.

design c_info

Returns information about the contents of a find collection.

Syntax

design c_info [*collection*] [**-array** *value*]

collection
Find collection information to display.

-array *value*
Specify an array to store collection information in.

design c_intersect

Defines common objects that are included in each of the collections being compared.

Syntax

design c_intersect *collectionList*

collectionList
List of collections separated by spaces.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

design c_list

Converts a collection to a Tcl list of objects.

Syntax

design c_list *collection*

collection
Collection to convert.

design c_print

Displays collections or properties in column format.

Syntax

design c_print *collection* [**-prop** *propertyName*] [**-file** *filename*] [**-append**]

collection
The collection to print as a table.

-prop
Writes a column in the table for properties of type *propname*.

-file
Writes the collection to *filename*.

-append
Appends to the file specified in **-file** rather than overwriting it.

design c_symdiff

Returns a new find collection containing the difference between two existing find collections.

Syntax

design c_symdiff *collection1 collection2*

collection1 The first collection.

collection2 The second collection.

design c_union

Combines multiple collections into a single collection.

Syntax

design c_union *collectionList*

collectionList
Space-separated list of collections.

design close

Closes the specified design ID. If no design ID is provided, this command closes the current active design.

Syntax

design close *designID*

designID

The design ID to close.

design expand

The design expand command identifies objects based on their connectivity, by expanding forward from a given starting point. Returns a collection.

Syntax

design expand [*-objectType*] [**-from** *object*] [**-thru** *object*] [**-to** *object*] [**-level** *integer*] [**-hier**] [**-leaf**] [**-seq**] [**-print**]

-objectType
Optionally specifies the type of object to be returned by the expansion. If you do not specify *objectType*, all objects are returned. The object type is one of the following:

-inst – returns all instances between the expansion points. This is the default.

-pin – returns all instance pins between the expansion points.

-net – returns all nets between the expansion points.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

-port – returns all top-level ports between the expansion points.

-from *object*

Specifies a list or collection of ports, instances, pins, or nets for expansion forward from all listed pins. Instances and input pins are automatically expanded to all output pins of the instances. Nets are expanded to all output pins connected to the net. If you do not specify this argument, backward propagation stops at a sequential element.

-thru *object*

Specifies a list or collection of instances, pins, or nets for expansion forward or backward from all listed output pins and input pins respectively. Instances are automatically expanded to all input/output pins of the instances. Nets are expanded to all input/output pins connected to the net. You can have multiple -thru lists for product of sum (POS) operations.

-to *object*

Specifies a list or collection of ports, instances, pins, or nets for expansion backward from all the pins listed. Instances and output pins are automatically expanded to all input pins of the instances. Nets are expanded to all input pins connected to the net. If you do not specify this argument, forward propagation stops at a sequential element.

-level *integer*

Limits the expansion to N logic levels of propagation. You cannot specify more than one -from, -thru, or -to point when using this option.

-hier

Modifies the range of any expansion to any level below the current view. The default for the current view is the top level and is defined with the define_current_design command as in the compile-point flow.

-leaf

Returns only non-hierarchical instances.

-seq

Modifies the range of any expansion to include only sequential elements. By default, the expand command returns all object types. If you want just sequential instances, make sure to define the *object_type* with the -inst argument, so that you limit the command to just instances.

-print

Evaluates the expand function and prints the first 20 results. If you use this command from HDL Analyst, results are printed to the Tcl window; for constraint-file commands, the results are printed to the log file at the

start of the Mapper section. For a full list of objects found, you must use c_print or c_list. Reported object names have prefixes that identify the object type. There are curly braces around each name to allow for spaces in the names. For example:

```
{i:reg1}
{i:reg2}
{i:\weird_name[foo$]}
{i:reg3}
<<found 233 objects. Displaying first 20 objects. Use
   c_print or c_list for all. >>
```

design find

Identifies design objects based on specified criteria.

Syntax

design find
[**-***objectType*] *pattern*
[**-seq**]
[**-inst** *instance*]
[**-net** *net*]
[**-port** *port*]
[**-pin** *pin*]
[**-view** *view*]
[**-depth** *viewNumber*]
[**-flat**]
[**-print**]
[**-filter** *expression*]
-in *value*
-below *value*
-nocase

-*objectType pattern*
Specifies the type of object to be found. Object types are view, inst, port, pin, or net. The *pattern* argument is required and specifies the search pattern to be matched. The pattern can include the * and ? wildcard characters.

-seq
Finds sequential (clocked) instances (the -inst object type is not required). This argument is equivalent to -filter @is_sequential.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

-hier

Extends the search downward through each level of the local hierarchy, instead of limiting the search to the current view. The default hierarchy separator for the search is the period (.).

-inst *instance*

Finds instances. If no -type option is set, find defaults to finding instances, nets, and ports.

-net *net*

Finds nets. If no -type option is set, find defaults to finding instances, nets, and ports.

-port *port*

Finds ports. If no -type option is set, find defaults to finding instances, nets, and ports.

-pin *pin*

Finds pins. If no -type option is set, find defaults to finding instances, nets, and ports.

-view *view*

Finds views. If no -type option is set, find defaults to finding instances, nets, and ports.

-depth *depth*

Sets the start depth for the search. *depth* may be a single hierarchy depth or a range. Using **-depth** with a range will cause **-hier** and **-flat** arguments to be ignored. Setting **-depth** to **0** will start the search at the top level.

-flat

Extends the search to all levels, but with -flat, the * wildcard character matches hierarchy separators as well as characters. This means that the following example finds instance a1_fft at the current level as well as the hierarchical instance a1.fft:

```
find -seq -flat a1*fft
```

-print

Prints the first 20 search results. For a full list of objects found, use c_print or c_list. If you use find from the shell, the results are printed to the Tcl window; if you find in the constraint file, the results are printed to the log file at the beginning of the Mapper section. Reported object names have prefixes that identify the object type and curly braces around each name to allow for spaces in the names as shown below:

```
{i:reg1}
{i:\weird_name[foo$]}
{i:reg2}
<<found 233 objects. Displaying first 20 objects. Use c_print
or c_list for all. >>
```

-filter *expression*
Further refines the results of find by filtering the results using the specified object property. For syntax details, refer to *find -filter*, on page 154.

-in *value*
Searches a collection to find a subset of the collection.

-below *value*
Sets the start point of the search to the specified instance path. Only search for objects below that point.

-nocase
Ignores the case when matching object names.

design get_prop

Returns a list of property values for an object or collection.

Syntax

design get_prop [*objectName* | *collection*] [**-prop** *value*] [**-all**] [**-array** *value*]

objectName | *collection*
The object or collection to use.

-prop
The property value to return.

-all
Prints all available properties.

-array
Specifies array where properties are stored. Only use with -all and the find collection must be limited to one object.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

design get

Returns the design ID for the current active design.

Syntax

design get

design list

Returns a list of available design IDs.

Syntax

design list

design open

View schematic of the design in its current state.

Syntax

design open [*netlist***]**

netlist

The netlist to view.

design set

Sets specified design ID as the active design.

Syntax

design set *designID*

designID

The design ID to set as active.

design top_level

Returns a Tcl list of top-level information in the following order: *lib topModule topView*.

Syntax

design top_level

For example:

```
design top_level
   work eight_bit_uc verilog
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

dump_metrics

Shows metrics and values available for the current implementation of a design. By default, only primary metrics are shown.

Syntax

dump_metrics [**-show_queries**] [**-all**]

-all
Shows detailed metrics as well as primary metrics for the design.

-show_queries
Shows available metrics in the form of a Tcl command that can be used to retrieve each metric.

The default output format is

table.[*object*|global]: *metric* = *value* [*units*] from *job* [// *description*]

Examples

```
% dump_metrics -all

*clock_conversion.global: icg_removed = 0  from premap
   //Number of ICG latches removed
*clock_conversion.global: icg_retained = 0  from premap
   //Number of ICG latches not removed
*clock_conversion.global: clean_clock_trees = 1  from fpga_mapper
   //Number of non-gated/non-generated clock trees
*clock_conversion.global: clean_clock_pins = 270  from fpga_mapper
   //Number of clock pins driven by non-gated/non-generated clock trees
*clock_conversion.global: gated_clock_trees = 0  from fpga_mapper
   //Number of gated/generated clock trees
*clock_conversion.global: gated_clock_pins = 0  from fpga_mapper
   //Number of clock pins driven by gated/generated clock trees
*clock_conversion.global: instances_converted = 0  from fpga_mapper
   //Number of sequential instances converted
*clock_conversion.global: instances_notconverted = 0  from fpga_mapper
   //Number of sequential instances left unconverted
*hdl_compile.global: modified_files = 28  from compiler
   //Total number of HDL input files compiled
*hdl_compile.global: modified_modules = 11  from compiler
   //Total number of modules compiled
*hdl_compile.global: total_modules = 11  from compiler
   //Total number of modules
*hdl_compile.global: total_files = 28  from compiler
   //Total number of HDL input files
*misc.global: Part = xc7vx485tffg1157-1  from fpga_mapper
*runtime.global: realtime = 3.154000 seconds from compiler
*runtime.global: cputime = 1.809612 seconds from compiler
```

© 2020 Synopsys, Inc.

```
*runtime.global: realtime = 1.452000 seconds from premap
*runtime.global: cputime = 1.622410 seconds from premap
*runtime.global: realtime = 9.881000 seconds from fpga_mapper
*runtime.global: cputime = 9.828063 seconds from fpga_mapper
*timing.global: "Worst Slack" = -0.445800 ns from fpga_mapper
utilization.global: LUT1 = 31  from fpga_mapper
utilization.global: LUT2 = 64  from fpga_mapper
utilization.global: LUT3 = 45  from fpga_mapper
utilization.global: LUT4 = 84  from fpga_mapper
utilization.global: LUT5 = 57  from fpga_mapper
utilization.global: LUT6 = 160  from fpga_mapper
utilization.global: IBUF = 1  from fpga_mapper
utilization.global: IBUFG = 1  from fpga_mapper
utilization.global: IOBUF = 24  from fpga_mapper
*utilization.global: "I/O primitives" = 26  from fpga_mapper
utilization.global: BUFG = 1  from fpga_mapper
*utilization.global: "I/O Register bits" = 0  from fpga_mapper
*utilization.global: "Total Luts" = 411  from fpga_mapper
```

Note: The * denotes a primary metric.

```
% dump_metrics -show_queries

query_metric clock_conversion.icg_removed -jobname premap
query_metric clock_conversion.icg_retained -jobname premap
query_metric clock_conversion.clean_clock_trees -jobname fpga_mapper
query_metric clock_conversion.clean_clock_pins -jobname fpga_mapper
query_metric clock_conversion.gated_clock_trees -jobname fpga_mapper
query_metric clock_conversion.gated_clock_pins -jobname fpga_mapper
query_metric clock_conversion.instances_converted -jobname fpga_mapper
query_metric clock_conversion.instances_notconverted -jobname fpga_mapper
query_metric hdl_compile.modified_files -jobname compiler
query_metric hdl_compile.modified_modules -jobname compiler
query_metric hdl_compile.total_modules -jobname compiler
query_metric hdl_compile.total_files -jobname compiler
query_metric misc.Part -jobname fpga_mapper
query_metric runtime.realtime -jobname compiler
query_metric runtime.cputime -jobname compiler
query_metric runtime.realtime -jobname premap
query_metric runtime.cputime -jobname premap
query_metric runtime.realtime -jobname fpga_mapper
query_metric runtime.cputime -jobname fpga_mapper
query_metric {timing.Worst Slack} -jobname fpga_mapper
query_metric {utilization.I/O primitives} -jobname fpga_mapper
query_metric {utilization.I/O Register bits} -jobname fpga_mapper
query_metric {utilization.Total Luts} -jobname fpga_mapper
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

Naming Conventions for Metrics

The naming convention used for metrics consists of the following:

- Table – Represents a group of related metrics, such as, timing, runtime, or clock conversion.
- Metric Name – Descriptive string used to query metrics. This name usually consists of lower case letters with underscores between words.
- Units – Values associated with the metric, such as ns or percent are only shown if details are specified.
- Object – Some metrics are associated with an object, while others are global. Objects can be a clock net name, view name, or an instance path.
- Description – Brief description of the metric.

For example, clock conversion metrics can be specified as follows:

Table	Metric Name	Description
clock_conversion	clean_clock_trees	Number of non-gated/non-generated clock trees
clock_conversion	clean_clock_pins	Number of clock pins driven by non-gated/non-generated clock trees
clock_conversion	gated_clock_trees	Number of gated/generated clock trees
clock_conversion	instances_converted	Number of sequential instances converted
clock_conversion	instances_notconverted	Number of sequential instances left unconverted

See Also

To query metrics for a design, see the following commands:

- query_available_metrics, on page 97
- query_metric, on page 100
- query_metric_details, on page 102

encryptIP

The encryptIP script lets you encrypt data with the OpenIP scheme. Download the script from the Synopsys website and run it directly from Perl. The Perl command line syntax for running the script is as follows:

Perl encryptIP Script Syntax

encryptIP
-in | **input** *inputFile*
-out | **output** *outputFileName*
-c | **cipher "{des-cbc** | **3des-cbc** |**aes128-cbc** }"
-k | key *symmetricEncryptionKeyInTextFormat*
-kx | keyx *symmetricEncryptionKeyInHexadecimalFormat*
-bd | build_date *ddmmmyyyy*
-om | outputmethod "{plaintext | blackbox | persistent_key}"
-incv | includevendor *vendorKeyBlock*
-dkn | datakeyname *sessionKeyName*
-dko | datakeyowner *sessionKeyOwner*
-a | author *dataAuthor*
-v | verbose

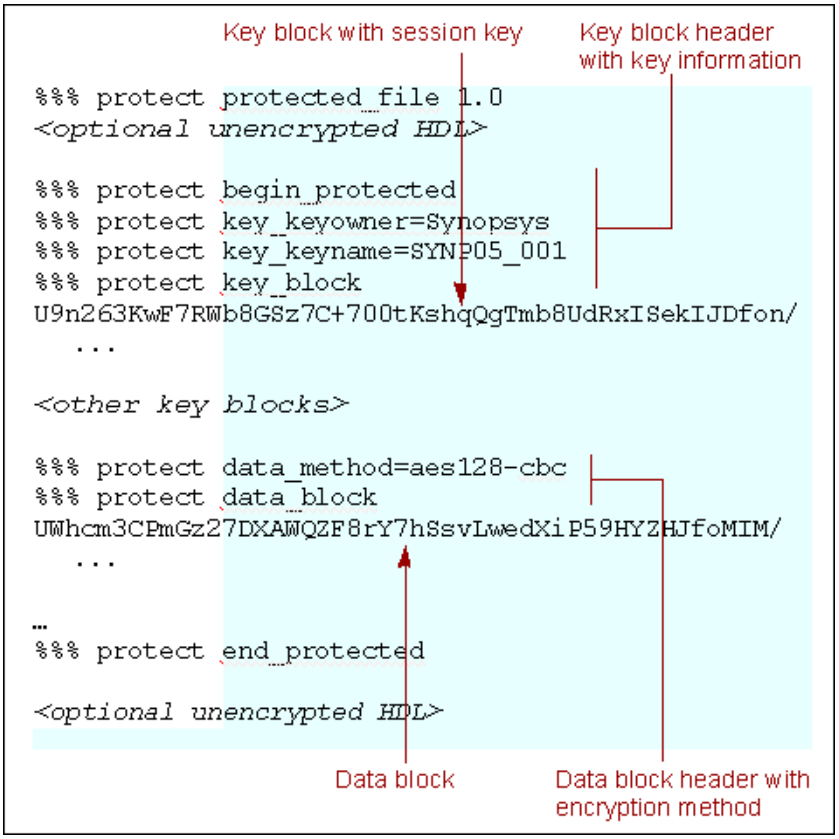
You must specify all required parameters.

-in | input	Names the input HDL file to be encrypted.
-out | output	Names the output file generated after encryption.
-c | cipher	Specifies the symmetric encryption cipher. The key length must match the algorithm being used, with each character using 8 bits. • des-cbc specifies the Data Encryption Standard (DES); uses a 64-bit key. • 3des-cbc specifies the Triple Data Encryption Standard (Triple DES); uses a 192-bit key. • aes128-cbc specifies the Advanced Encryption Standard (AES Rijndael); uses a 128-bit key. See *Encryption and Decryption*, on page 477 in the *FPGA Synthesis User Guide* for an overview.
-k | key	Specifies the symmetric data decryption key used to encode your HDL data block. The key is in text format, and can be any string (e.g. ABCDEFG).The exact length of the key depends on the data method you use.

-kx | keyx*	Optional parameter. Specifies the symmetric encryption key in hexadecimal format.
-bd | build_date	Specifies a date (ddmmmyyyy). The IP only works in Synopsys software released after the specified date. This option lets you force users to use newer Synopsys FPGA releases that contain more security features. Contact Synopsys if you need help in deciding what build date to use.
-om | outputmethod	Determines how the IP is treated in the output after synthesis: • plaintext specifies that the IP is unencrypted in the synthesis netlist. • blackbox specifies that the IP is treated as a black box, and only interface information is in the output. • persistent_key is the default setting. It re-encrypts the output after synthesis in accordance with the OpenIP standard. See *Specifying the Script Output Method for OpenIP Encryption* , on page 494 for more information.
-incv | includevendor	Optional parameter that specifies a key block for an EDA vendor, so that IP can be read by the vendor tools. C
-dkn | datakeyname	Specifies a string that denotes your session key, that was used to encrypt your IP.
-dko | datakeyowner	Optional parameter that names the owner of the session key. The value can be any string.
-a | author	Optional parameter that names the author of the session key. The value can be any string.
-v | verbose	Specifies that the script run in verbose mode.

Example of encryptIP (OpenIP) Script Output

The following is an example of the script output:



For brief descriptions of the pragmas used in the output of the encryptIP script, see *Pragmas Used by Encryption Scripts*, on page 58.

© 2020 Synopsys, Inc.

encryptP1735

Run the encryptP1735 script directly from Perl.

The script supports different models for encrypting HDL files and accessing the encrypted information (see *IEEE 1735 Encryption Use Models*, on page 59). The use model is determined by how blocks are marked for encryption in the HDL, combined with the information in the public keys file, which is described in *Public Keys File*, on page 57.

Perl encryptP1735 Script Syntax

encryptP1735
-l | **list** *listofFiles*
[**-pk** | **public_keys** *keyFileName*]
[**-sk** | **showkey**]
[**-verbose**]
[**-verilog**]
[**-vhdl**]
[**-log** *logFileName*]
[**-h** | **-help**]

The following table describes the command-line arguments.

-l | **list**	Specifies a list of the files to be encrypted; *listofFiles* is a list of the non-encrypted HDL input files with each filename entry on a separate line.
-pk | **public_keys**	Specifies the public keys repository file. This file contains public keys for various tools. If the encryption envelope contains a key block with a particular keyowner and keyname, the script searches the public keys file to find a corresponding public key to use during key-block generation. See *Public Keys File* , on page 57 for information about this file.
-sk | **showkey**	When used, the encryption script displays the session key in use. This is useful when random keys are used and you want to know which key is being used.
-verbose	Prints more detailed messages to the screen or log file.

-verilog	Specifies Verilog HDL file format when filename does not include a .v or .sv extension.
-vhdl	Specifies VHDL HDL file format when filename does not include a .vhd or .vhdl extension.
-log	Prints messages to the specified log file.

Public Keys File

The encryptP1735.pl encryption script requires public key information, which is specified in a designated file (-public_keys or -pk) option). This file includes public keys for each of the tools that are allowed access to the envelope with the encrypted data. The default keys file is called keys.txt and is located with the encryption script in the lib directory of the tool installation.

```
// Use verilog pragma syntax in this file

`pragma protect version=1
`pragma protect author="default"
`pragma protect author_info="default"

`pragma protect key_keyowner="Synopsys", key_keyname="SYNP15_1",
key_method="rsa"
`pragma protect key_public_key
<public_key_block>

// Add additional public keys below this line

// Add additional public keys above this line

`pragma protect data_keyowner="default-ip-author"
`pragma protect data_keyname="default-ip-key"
`pragma protect data_method="aes128-cbc"

// End of file
```

For the partial file with all pragmas use model, the following pragma attribute values must match the corresponding values in the key-block section of the encryption envelope:

```
`pragma protect key_keyowner="Synopsys", key_keyname="SYNP15_1",
key_method="rsa"
```

For information on the pragmas supported, see *Pragmas Used by Encryption Scripts*, on page 58.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Pragmas Used by Encryption Scripts

Both the encryptIP1735 and encryptIP (OpenIP) schemes use the pragmas described in the following tables. Note the following:

- The %%% protect directive must be placed at the exact beginning of a line.
- Exactly one white-space character must separate the %%% sequence from the command that follows.

The following table lists the pragmas used. In Verilog, the pragma must be preceded by the word pragma; this is not required in VHDL.

General Pragmas	
%%% protect protected_file 1.0	Line 1 of file with encrypted data
%%% protect begin_protected	Marks the beginning of data to be encrypted
%%% protect end_protected	Marks the end of data to be encrypted
%%% protect comment *comment*	Single-line plain-text comment
%%% protect begin_comment	Marks the beginning of plain-text comment block
%%% protect end_comment	Marks the end of plain-text comment block
Data Block Pragmas (IP Author Data Encryption Information)	
%%% protect author=*string*	Lists name of IP author
%%% protect version=1	Specifies encryption version; required only for IEEE 1735 Partial File with Standard Pragmas encryption use model
%%% protect data_method=des-cbc | 3des-cbc |aes128-cbc	Specifies the DES encryption method used: • des-cbc: Data Encryption Standard (DES) • 3des-cbc: Triple DES • aes128-cbc: Advanced Encryption Standard (AES)
%%% protect data_block	Immediately precedes the encrypted data block
Key Block Pragmas (IP Consumer Public Key Information)	
%%% protect key_keyowner=*string*	Lists the owner of the key

%%% protect key_keyname=*string*	Name recognized by the Synopsys software to select the key block
%%% protect key_method=*string*	Encryption algorithm (RSA currently supported)
w %%% protect key_block	Immediately precedes encrypted key block

IEEE 1735 Encryption Use Models

Encryption models determine the scope of what gets encrypted and who can access the files. The encryptP1735 script lets you use these use models to encrypt HDL files:

Encryption Model	Details
Full file	Full-File Use Model, on page 59
Partial file with minimal pragmas	Partial File with Minimal Pragmas Use Model, on page 60
Partial file with standard pragmas (Recommended)	Partial File with Standard Pragmas Use Model, on page 61
Partial file with IEEE pragmas	Partial File with IEEE Pragmas Use Model, on page 63

Full-File Use Model

Use this model to encrypt the entire file. The entire HDL file is included in the decryption envelope. This model uses the keys.txt file to define which consumers have access to the encrypted data.

HDL	• Contains no encryption pragmas (entire file is encrypted)
keys.txt	• Contains public key information for multiple downstream tools • Owners of all public keys listed have access to the entire file

This Verilog example encrypts the whole file (tb_encrypt.v), including the module named secret that it contains.

```
module secret (a, b, clk);
input a, clk;
output b;
reg b=0;
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

```
always @(posedge clk) begin
   b = a;
end
endmodule
```

Run the script to encrypt the file:

```
perl encryptP1735.pl -list mylist -log encryptP1735.log
```

This command runs the script on a file (mylist), which lists the single Verilog file tb_encrypt.v. The command uses the default keys.txt file from the lib directory, and creates the decryption envelope file tb_encrypt.vp. Messages from the run are written to the encryptP1735.log file.

Partial File with Minimal Pragmas Use Model

With this encryption model, pragma protect begin and pragma protect end pragmas are used to indicate the start and end points of encryption regions. This model is best suited for cases where every encryption region must be encrypted for every key in the key file. When using this model, you must encrypt the entire module. The encryptP1735.pl script checks all the begin and end pragmas and generates the decryption envelope for each tool specified in the keys file.

HDL	• Contains individual blocks marked for encryption (partial file) • Contains no public key information
keys.txt	• Contains public key information for multiple downstream tools • Owners of all public keys listed have access to all encrypted HDL blocks

To illustrate this use model, consider a single, Verilog file (tb_encrypt.v) to be encrypted with only begin and end pragmas. This file contains a single module named secret.

```
`pragma protect begin
module secret (a, b, clk);
input a, clk;
output b;
reg b=0;

always @(posedge clk) begin
   b = a;
end
endmodule
`pragma protect end
```

When you run the script with the following command, it uses the begin and end pragmas specified in the HDL file to encrypt the file:

```
perl encryptP1735.pl -list mylist -pk keys.txt
```

Here, the list file (mylist) names the Verilog file tb_encrypt.v. The command encrypts the data between the begin and end pragmas and creates the decryption envelope file tb_encrypt.vp for all tools listed in the key file. No log file (-log option) is specified, so messages are not written to a log file.

Partial File with Standard Pragmas Use Model

This is the recommended encryption use model. It is the most flexible because you can choose to encrypt individual blocks instead of the entire file and specify which tools can access each encrypted block on a per-block basis. When using this model, you must encrypt the entire module. This model requires a side file (keys.txt) that contains public key information for IP consumers.

HDL	• Marks individual blocks for encryption (partial file) • Includes public key encryption pragmas for each tool that is allowed access to an encrypted block, except the key itself (key_public_key) • Key information must match the information in the keys.txt file • Can allow different keys access to different blocks
keys.txt	• Contains public key information for multiple downstream tools • Must include all public key encryption pragmas, as well as the public key itself • Only those owners of public keys listed in the HDL before the encrypted block have access to that block; all public keys listed in keys.txt need not be used in the HDL

If there are conflicting pragmas defined in the HDL and the keys.txt file, the HDL pragma takes precedence over the corresponding pragma in the keys.txt file. For example, if data_method in the HDL is defined as des-cbc but the same pragma in the keys.txt file defines it as aes128-cbc, the HDL definition is used and copied to the decryption envelope:

```
data_method="des-cbc"
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

Verilog Example

This example encrypts a single Verilog file (tb_encrypt.v). The file contains a module named secret and includes all the encryption-related pragmas in the HDL, with the exception of key_public_key.

```
`pragma protect version=1
`pragma protect encoding=(enctype="base64")
`pragma protect author="author-a", author_info="author-a-details"
`pragma protect encrypt_agent="encryptP1735.pl",
encrypt_agent_info="Synplify encryption scripts"
`pragma protect key_keyowner="Synopsys",key_keyname="SYNP15_1",
key_method="rsa", key_block
`pragma protect
data_keyowner="ip-vendor-a",data_keyname="fpga-ip",
data_method="des-cbc"
`pragma protect begin

module secret (a, b, clk);
input a, clk;
output b;
reg b=0;
always @(posedge clk) begin
   b = a;
end
endmodule

`pragma protect end
```

The script is then run with the following command, where the list file (mylist) names a single file, tb_encrypt.v. The command uses the default keys.txt file from the *installLocation*/lib directory as the public keys file to create the decryption envelope file tb_encrypt.vp. No log file is specified, so messages from the run are not sent to a log file.

```
perl encryptP1735.pl -list mylist -pk keys.txt
```

VHDL Example

This example partially encrypts a VHDL file (tb_encrypt.vhd) where all encryption pragmas are specified in the file, except for key_public_key. The file contains a single entity/architecture pair named secret. For VHDL pragmas, just use the keyword protect.

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

```
entity secret is
   port (clk : in std_logic;
         a : in std_logic;
         b : out std_logic);
end entity;

`protect version=1
`protect author="author-a", author_info="author-a-details"
`protect encrypt_agent="encryptP1735.pl",
encrypt_agent_info="Synplify encryption scripts"
`protect encoding=(enctype="base64")
`protect key_keyowner="Synopsys", key_keyname="SYNP15_1",
key_method="rsa", key_block
`protect data_keyowner="ip-vendor-a", data_keyname="fpga-ip",
data_method="des-cbc"
`protect begin

architecture rtl of secret is
signal b_reg: std_logic;
begin
   process (clk) is
   begin
      if rising_edge(clk) then
         b_reg <= a;
      end if;
   end process;
   b <= b_reg;
end architecture;

`protect end
```

Encrypt the file with the following command, where the list file (mylist) names a single VHDL file, tb_encrypt.vhd. The command uses the default keys.txt file from the directory *installLocation*/lib as the public keys file to create the decryption envelope file tb_encrypt.vhdp. Messages are not captured in a log file.

```
perl encryptP1735.pl -list mylist -pk keys.txt
```

Partial File with IEEE Pragmas Use Model

Like the partial file with standard pragmas model, this use model is flexible, but it does not require a side file with the key block information. This makes it the most portable model, because all the information, including the key block information for each IP consumer, is included in the source code. When using this model, you must encrypt the entire module.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

HDL	• Marks individual blocks for encryption (partial file) • Includes public key encryption pragmas for each tool that is allowed access to an encrypted block, including the key itself (key_public_key) • Can allow different keys access to different blocks
keys.txt	• Not required

```
module top (qa, qb, a, b, clk);
input a, b, clk;
output qa, qb;
enc_and iand (qa, a, b, clk);
enc_or ior (qb, a, b, clk);
endmodule

`pragma protect version=1
`pragma protect author="author-a", author_info="author-a-details"

`pragma protect key_keyowner="Synplicity", key_keyname="SYNP15_1",
key_method="rsa"

`pragma protect key_public_key
<public_key_block>

`pragma protect key_keyowner = "XYZ"
`pragma protect key_method = "rsa"
`pragma protect key_keyname = "XYZ8_001"

`pragma protect key_public_key
<public_key_block>

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

module enc_and (q, a, b, clk);
input a, b, clk;
output q;
reg q=0;
always @(posedge clk) begin
   q = a & b;
end
endmodule

`pragma protect end
```

```
`pragma protect version=1
`pragma protect author="author-a", author_info="author-a-details"
`pragma protect key_keyowner="Synplicity", key_keyname="SYNP15_1",
key_method="rsa"
`pragma protect key_public_key
<public_key_block>

`pragma protect key_keyowner = "XYZ"
`pragma protect key_method = "rsa"
`pragma protect key_keyname = "XYZ8_001"
`pragma protect key_public_key
<public_key_block>

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

module enc_or (q, a, b, clk);
input a, b, clk;
output q;
reg q=0;
always @(posedge clk) begin
   q = a | b;
end
endmodule

`pragma protect end
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

get_env

The get_env command reports the value of a predefined system variable.

Syntax

get_env *systemVariable*

Use this command to view system variable values. The following example shows you how to use the get_env command to see the value of the previously created MY_PROJECT environment variable. The MY_PROJECT variable contains the path to an HDL file directory, so get_env reports this path.

```
get_env MY_PROJECT
d:\project\hdl_files
```

In the project file or a Tcl script, you can define a Tcl variable that contains the environment variable. In this example, my_project_dir contains the MY_PROJECT variable, which points to an HDL file directory.

```
set my_project_dir [get_env MY_PROJECT]
```

Then, use the $*systemVariable* syntax to access the variable value. This is useful for specifying paths in your scripts, as in the following example which adds the file myfile1.v to the project.

```
add_file $my_project_dir/myfile1.v
```

get_option

The get_option command reports the settings of predefined project and device options. The options are the same as those for set_option. See *set_option*, on page 113 for details.

Syntax

get_option *-optionName*

hdl_define

For Verilog designs, this command specifies values for Verilog text macros. You can specify text macro values that you would normally enter using the Verilog `define statement in a Verilog file included at the top of the synthesis project. The parameter value is valid for the current implementation only.

This command is equivalent to the set_option -hdl_define command.

Syntax

hdl_define
-set "*directive*=*value* [*directive*=*value* **...]"**
-clear
-list

Examples

```
hdl_define -set "SIZE=32"
```

This statement specifies the value 32 for the SIZE directive; the following statement is written to the project file:

```
set_option -hdl_define -set "SIZE=32"
```

To define multiple directive values using hdl_define, enclose the directives in quotes and use a space delimiter. For example:

```
hdl_define -set "SIZE=32 WIDTH=8"
```

The software writes the following statement to the prj file:

```
set_option -hdl_define -set "size=32 width=8"
```

See Also

Compiler Directives and Design Parameters, on page 369 for information on specifying compiler directives in the GUI.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

hdl_param

The hdl_param command shows or sets HDL parameter overrides. For the GUI equivalent of this command, select Project->Implementation Options->Verilog/VHDL.

Syntax

hdl_param
-add {*paramName***}**
-list | **-set** *paramName* **{***paramValue***}**
-clear
-overrides

The following table describes the command arguments.

Option	Description
-add	Adds a parameter override to the project.
-list	Shows parameters for the top-level module only and lists values for parameters if there is a parameter override.
-set	Sets a parameter override and its value for the active implementation. Only the parameter value is enclosed within curly braces.
-clear	Clears all parameter overrides of the active implementation.
-overrides	Lists all the parameter override values used in this project.

Examples

In batch mode, to set generic values using the set_option command in a project file, specify the hdl_param generic with quotes and enclose it within {}. For example:

```
set_option -hdl_param -set ram_file {"init.mem"}
set_option -hdl_param -set simulation {"false"}
```

Suppose the following parameter is set for the top-level module.

```
set_option -hdl_param -set {"width=8"}
```

Add a parameter override and its value, then list the parameter override.

```
hdl_param -add {"size=32"}
hdl_param -list "size=32"
```

You can specify hdl_param generics with different types, such as, an integer, std_logic_vector, or string value for VHDL. Here are some examples that show how to define these generics:

- With an integer value

  ```
  set_option -hdl_param -set DATA_WIDTH 4
  ```

- With a std_logic_vector value

  ```
  set_option -hdl_param -set MY_SLV {"0011"}
  ```

- Using a string value

  ```
  set_option -hdl_param -set initialization_file {"table2"}
  ```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

help

The help command displays the usage syntax and description for the specified command in the Tcl window.

Syntax

help *commandName* | *wildcardTerm*

Examples

```
help set_option

usage:set_option -<optionName> <optionValue> -- set option
         on active implementation

      get_option -<optionName> -- return option value on
         active implementation

help c_*

c_diff
c_filter
c_info
c_intersect
c_list
c_member
c_print
c_symdiff
c_union
```

© 2020 Synopsys, Inc.

history

Returns a numbered list of executed Tcl commands.

Syntax

history [**event** *number* | **clear** | **info** [*number*] | **keep** [*number*] | **nextid** | **redo** [*number*]]

event *number*
Returns command *number* from the history list.

clear
Clears the history list.

info [*number*]
Returns the last *number* of commands. If no *number* is included, returns all.

keep [*number*]
Sets the number of commands to save in history. Also returns the current setting.

nextid
Returns the index number that the next command will be assigned to in the history list.

redo [*number*]
Executes the *number* command. If no *number* is given, executes the latest command.

Examples

```
history event 12
```

```
history redo 4
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

impl

The impl command adds, removes, or modifies an implementation.

Syntax

impl
-add [*implName*] [*model*]
-name *implName*
-remove *implName*
-active [*implName*]
-list
-type *implType*
-result_file
-dir

The following table describes the command arguments.

Option	Description
-add	Adds a new device implementation. If: • *implName* is not specified, creates a unique implementation name by incrementing the name of the active implementation. • you want to add a new implementation copied from implementation *model*.
-name	Changes the name of the active implementation.
-remove	Removes the specified implementation.
-active	Reports the active implementation. If you specify an implementation name, changes the specified name to the active implementation.
-list	Lists all the implementations used in this project.
-type	Specifies the type of implementation to add. For example, the: • -type fpga option creates an FPGA implementation. • -type identify option creates an Identify implementation.
-result_file	Displays the implementation results file.
-dir	Displays the implementation directory.

Examples

The following command sequence lists all implementations, reports the active implementation, and then activates a different implementation.

```
% impl -list
design_worst design_typical design_best

% impl -active
design_best

%impl -active design_typical

% impl -active
design_typical

% impl -add rev_1_identify mixed -type identify
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

job

The job command, for place and route job support, creates, removes, identifies, runs, cancels, and sets/gets options for named P&R jobs.

Syntax

job *jobName* [**-add** *jobType* |**-remove** |**-type** |**-run** [*mode*] |**-cancel** |
-option *optionName* [*optionValue*]]

job -list

The following table describes the command options.

Option	Description
-run	Runs the P&R job, according to the specified options:
-add *jobType*	Creates a new P&R job for the active implementation.
-cancel	Cancels a P&R job in progress.
-remove	Removes a P&R job from an active implementation
-list	Returns a list of the P&R jobs in the active implementation.
-remove	Removes a P&R job from the active implementation
-option *optionName* [*optionValue*]	Get/set options for *jobName*.
-type	Returns the P&R job type.

Examples

```
% job pr_2 -add par
% job pr_2 -run
% job pr_2 -option enable_run 1
```

log_filter

This command lets you filter errors, notes, and warning messages. The GUI equivalent of this command is the Warning Filter dialog box, which you access by selecting the Warnings tab in the Tcl window and then clicking Filter. For information about using this command, see *Filtering Messages in the Message Viewer*, on page 207 in the *User Guide*.

Syntax

log_filter -field *fieldName*==*value*
log_filter -show_matches
log_filter -hide_matches
log_filter -enable
log_filter -disable
log_filter -clear

The following table shows valid *fieldName* and *value* values for the -field option:

Fieldname	Value
type	Error | Warning | Note
id	The message ID number. For example, MF138
message	The text of the message. You can use wildcards.
source_loc	The name of the HDL file that generated the message.
log_loc	The corresponding srr file (log).
time	The time the message was generated.
report	The log file section. For example, Compiler or Mapper.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Example

```
log_filter -hide_matches
log_filter -field type==Warning -field message==*Una*
   -field source_loc==sendpacket.v -field log_loc==usbHostSlave.srr
   -field report=="Compiler Report"
log_filter -field type==Note
log_filter -field id==BN132
log_filter -field id==CL169
log_filter -field message=="Input *"
log_filter -field report=="Compiler Report"
```

log_report

This command lets you write out the results of the log_filter command to a file. For information about using this command, see *Filtering Messages in the Message Viewer*, on page 207 in the *User Guide*.

Syntax

You specify this command after the log_filter commands.

log_report -print *fileName*

Example

```
log_report -print output.txt
```

message_override

Allows you to suppress or override the log file message ID specifications with another type or limit.

Use -limit and -count, to limit the number of occurrences for all messages or specific messages in each log file. Messages that exceed the limit still show up in the Report Summary page and can be retrieved later from the message database. Suppressing messages is the same as -limit *ID* -count 0; errors cannot be suppressed or limited.

Syntax

message_override [**-suppress** *value*] [**-read_file** *value*] [**-error** *value*] [**-warning** *value*] [**-note** *value*] [**-remove** *value*] [**-global**] [**-clear**] [**-limit** *value*] [**-count** *value*]

The following table describes the command arguments and options.

Option	Description
-suppress *value*	Lists message IDs to suppress in the log file.
-read_file *value*	Reads the specified message override file.
-error *value*	Lists the message ID type as an error.
-warning *value*	Lists the message ID type as a warning.
-note *value*	Lists the message ID type as a note.
-remove *value*	Removes the override and resets the message type to its original value.
-global	Allows message operations to be applied globally. Otherwise, the override operation is only applied on messages for the current project.

© 2020 Synopsys, Inc.

Option	Description
-clear	Removes all overrides and resets messages to their original types.
-limit *value*	Lists message IDs for a specific log file to the specified limit. Use with the -count argument. Also, use • message_override -limit default -count 1000 7 - Changes the default limits for all messages • message_override -limit default -count unlimited - Changes the default limits for all messages to unlimited
-count *value*	Counts the message IDs specified with the -limit argument.

Examples

It is recommended that you set the default limit to something other than unlimited, if possible. To do this, you can specify the following:

```
message_override -default_limit 100
```

More examples are shown below:

1. Upgrade messages with ID MF446 to be treated as an error.

   ```
   message_override -error MF446
   ```

2. Suppress messages with ID BN101 (cannot be done for errors):

   ```
   message_override -suppress BN101
   ```

3. Limit the number of occurrences of messages with IDs MF580 and MF581 to 1000 each in each log file:

   ```
   message_override -limit {MF580 MF581} -count 1000
   ```

4. Unlimit the number of occurrences of messages with ID CL118 in logs and reports:

   ```
   message_override -limit CL118 -count unlimited
   ```

5. Clear existing message overrides:

   ```
   message_override -clear
   ```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

open_design

The open_design command specifies a netlist file (.srs or .srm) that can be used to search the database with the Tcl find command in batch mode. With open_design, you can use find without having to open an RTL or Technology view. Use open_design to read in the .srs or .srm file before issuing the find command. See the example below.

Syntax

open_design *filename*

Where:

- *filename* is the RTL (.srs) or Technology (.srm) file that can be used to search the database. The specified netlist is loaded on-demand to minimize memory resources.

Example

```
project -load ../examples/vhdl/prep2_2.prj
open_design prep2_2.srs
set a [find -inst *]
c_print $a -file a.txt
open_design prep2_2.srm
set b [find -net *]
c_print $b -file b.txt
```

In the example above, prep2_2 is loaded and the information from the RTL view file is read in. Then, the find command searches for all instances in the design and prints them to file a. Next, the technology view file is read in, then find searches for all nets in the design and prints them to file b.

See Also

- *find*, on page 147.

open_file

The open_file command opens views within the tool. The command accepts two arguments: -rtl_view and -technology_view.

Syntax

open_file -rtl_view |-technology_view

The -rtl_view option displays the RTL view for the current implementation, and the -technology_view option displays the technology view for the current implementation. Views remain displayed until overwritten and multiple views can be displayed.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

partdata

The partdata command loads part files and returns information regarding a part such as available families, family parts, vendors, attributes, grades, packages.

Syntax

partdata
-load *filename*
-family
-part *family*
-vendor *family*
-attribute *attribute family*
-grade [*family*:]*part*
-package [*family*:]*part*
-oem [*family*:]*part*

Option	Description
-load *filename*	Loads part file.
-family	Lists available technology families.
-part *family*	Lists all parts in specified family.
-vendor *family*	Returns vendor name for the specified family.
-attribute *attribute family*	Returns the value of the job attribute for the specified family.
-grade [*family*:]*part*	Lists the speed grades available for the specified part.
-package [*family*:]*part*	Lists the packages available for the specified part.
-oem [*family*:]*part*	Returns true if the part entered is an OEM part.

Example

The following example prints out the available vendors, their supported families, and the parts for each family.

```
% foreach vendor [partdata -vendorlist]
% puts VENDOR:$vendor;
% foreach family [partdata -family $vendor]
% puts \tFAMILY:$family;
% puts \t\tPARTS:;
% foreach part [partdata -part $family]
% puts \t\t$part;
```

program_terminate

Immediately terminates the tool session without prompting or saving any data.

Syntax

program_terminate

Arguments and Options

None

Description

The program_terminate command terminates a tool session without prompting or saving data. Use this command with caution as any unsaved data is lost and cannot be recovered.

Examples

```
program_terminate
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

program_version

Returns the product and software release version.

Syntax

program_version

Arguments and Options

None

Description

The program_version command returns the software product version number.

Examples

```
% program_version
Synplify Pro L-2017.09
```

project

The project command runs job flows to create, load, save, and close projects, to change and examine project status, and to archive projects.

Syntax

project -**run** [**-all**] [**-bg**] *implementationList* [**-impl** *implementationName*]
[**-clean**] *implementationList* [**-impl** *implementationName*]
[**-parallel**] *implementationList* [**-impl** *implementationName*]
[**-from** *proccessName*] [**-to** *processName*]

project {**-new** [*projectPath*] |**-load** *projectPath* | **-close** [*projectPath*]
|**-save** [*projectPath*] |**-insert** *projectPath*} |

project {**-active** [*projectName*] |**-dir** |**-file** |**-name** |**-list** |**-filelist** |
-fileorder *filepath1 filepath2* [*... filepathN*] |**-addfile** *filepath* |
-movefile *filepath1* [*filepath2*] |**-removefile** *filepath*}

project {**-result_file** *resultFilePath* |**-log_file** [*logfileName*]

project -propagate_params

project -copy [**-project** *filename*] [**-implement** *implementationName*]
[**-dest_dir** *pathname*] [**-copy_type** {**full** | **local** | **customize**}]
[**-add_srs** [*fileList*] **-no_input**]

project -unarchive [**-archive_file** *pathname/filename*] [**-dest_dir** *pathname*]

run option

The run option lets you synthesize selected implementations of a Project file. You can choose to use the arguments for the run option independently or in any combination. The arguments available are described in the table below.

You can also use the Batch Run Setup dialog box to set the arguments to use with the run option. For details, see *Run Implementations Setup Command*, on page 385.

Option	Description
-run [**-all**] [**-bg**] [**-clean**] [**-from**] [**-parallel**] [**-to**] [*processName*]	Synthesizes the project, according to the specified options:

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Option	Description
	You can use *run* with any of the following arguments: • **-all** - Runs all implementations of the active project. • **-bg** - Runs specified implementations in non-blocking background mode. This is the default. • **-clean** - Runs specified implementations, while ignoring up-to-date checking. This option cleans all previous results and forces a complete rerun. • **-parallel** - Runs specified implementations concurrently. Additional licenses are required for each job. • **-from** - Runs from and including the specified process name. • **-to** - Runs up to and including the specified process name. • *processName* - Specifies the process name.

Option	Description
	The *mode* can be one of the following keywords: • **compile** - Compiles the active project, but does not map it. • **constraint_check** - Validates the syntax and applicability of constraints defined in one or more constraint files. • **fsm_explorer** - Selects optimum FSM-encoding style for finite-state machines. • **netlist_optimizer** - Runs netlist optimization. • **syntax_check -** Verifies that the HDL is syntactically correct; errors are reported in the log file. • **synthesis -** Default mode if no mode is specified. Compiles (if necessary) and synthesizes the currently active project. If followed by the -clean option (project -run synthesis -clean), resynthesizes the entire project, including the top level and *all compile points*, whether or not their constraints, implementation options or source code changed since the last synthesis. If not followed by -clean, only compile points that have been modified are resynthesized. • **synthesis_check -** Verifies that the design is functionally correct; errors are reported in the log file. • **timing** - Runs the Timing Analyst. This is equivalent to clicking the Generate Timing button in the Timing Report Generation dialog box with user-specified values.
	• **write_netlist** - Writes the mapped output netlist to structural Verilog (vm) or VHDL (vhm) format. You can also use this command in an incremental timing analysis flow. For details, see *Run Menu* , on page 381 and *Generating Custom Timing Reports with STA* , on page 366.

The following table describes the rest of the Project file command options.

Option	Description
-load *projectPath*	Opens and loads the project file specified by *projectPath*.

© 2020 Synopsys, Inc.

Option	Description
-close [*projectPath*]	Closes the currently active project. If *projectPath* is specified, closes the specified project.
-save [*projectPath*]	Saves the currently active project. If *projectPath* is specified, saves the specified project.
-insert *projectPath*	Adds the specified project to the workspace project.
-active [*projectName*]	Shows the active project. If *projectName* is specified, makes the specified project the active project.
-dir	Shows the project directory for the active project.
-file	Returns the path to the active project.
-name	Returns the filename (prj) of the active project.
-list	Returns a list of the loaded projects.
-filelist	Returns the pathnames of the files in the active project.
-fileorder *filepath1 filepath2* [... *filepathN*]	Reorders files by adding the specified files to the end of the project file list.
-addfile *filepath*	Adds the specified file to the project.
-movefile *filepath1* [*filepath2*]	Moves *filepath1* to follow *filepath2* in project file list. If *filepath2* is not specified, moves *filepath1* to top of list.
-removefile *filepath*	Removes the specified file from the project.
-result_file *resultFilePath*	Changes the name of the synthesis result file to the path specified.
-log_file [*logfileName*]	Reports the name of the project log file. If *logfileName* is specified, changes the base name of the log file.

Option	Description

-archive
-project *filename*
[**-root_dir** *pathname*]
-archive_file *filename*.**sar**
-archive_type {**full** | **local** | **customize**}
-add_srs [*fileList*]
-no_input
	• **project** *filename* - copies a project other than the active project. If you do not use this option, by default the active project is copied.
	• **root_dir** *pathname* - specifies the top-level directory containing the project files.
	• **archive_file** *filename* - is the name of the archived project file.
	• **archive_type** - specifies the type of archive:
	  • full - performs a complete archive; all input and result files are contained in the archive file.
	  • customize - performs a partial archive; only the project files that you select are included in the archive.
	  • local - includes only project input files in the archive; does not include result files.
	• **add_srs** - adds the listed `srs` files to the archived project. Use the -no_input option with this command. If *fileList* is omitted, adds all `srs` files for the project/implementations. The `srs` files are the RTL schematic views that are output when the design is compiled (Run->Compile Only).
	For more information about, and examples of the project -archive command, see *Archive Utility* , on page 91.

Option	Description

-copy
-project *filename*
-implement *implementationName*
-dest_dir *pathname*
-copy_type {**full** | **local** | **customize**}
-add_srs [*fileList*]
-no_input
	• **project** *filename* - copies a project other than the active project. If you do not use this option, by default the active project is copied.
	• **implement** *implementation_name* - archives all files in the specified implementation.
	• **dest_dir** *directory_pathname* - specifies the directory in which to copy the project files.
	• **copy_type** - specifies the type of file/project copy:
	  • full - performs a complete copy; all input and result files are contained in the archive file.
	  • customize - performs a partial copy; only the project files that you select are included in the archive.
	  • local - includes only project input files in the copy; does not include result files.
	• **add_srs** - adds the listed `srs` files to the archived project. Use the -no_input option with this command. If *fileList* is omitted, adds all `srs` files for the project/implementations. The `srs` files are the RTL schematic views that are output when the design is compiled (Run->Compile Only).
	For more information about, and examples of the project -copy command, see *Archive Utility* , on page 91.

-unarchive **-archive_file** *pathname/filename* **-dest_dir** *pathname*	• **archive_file** *pathname/filename* - is the name of the archived project file. • **dest_dir** *pathname* - specifies the directory in which to write the project files. For more information about, and examples of the project -unarchive command, see *Archive Utility* , on page 91.

project Command Examples

Load the project top.prj and compile the design without mapping it. Compiling makes it possible to create a constraint file with the SCOPE spreadsheet and display an RTL schematic representation of the design.

```
% project -load top.prj
% project -run compile
```

Load a project and synthesize the design.

```
% project -load top.prj
% project -run synthesis
```

In the example above, you can also use the command project -run, since the default is synthesis.

Archive Utility

The archive utility provides a way to archive, extract, or copy your design projects. An archive file is in Synopsys proprietary format and is saved to a file name using the .sar extension. You can also use this utility to submit your design along with a request for technical support.

The archive utility is available through the Project menu in the GUI or through the project Tcl command. See the following for details:

For information about ...	See ...
Archiving, un-archiving, or copying projects	*Archiving Files and Projects* , on page 105 in the *User Guide*

Project Archive Examples

The following example archives all files in the project and stores the files in the specified .sar file:

```
project -archive -project c:/proj1.prj
      -archive_file c:/archive/proj1.sar
```

The next example archives the project file (.prj) and all local input files into the specified .sar file.

```
project -archive -project c:/proj1.prj -archive_type local
      -archive_file c:/archive/proj1.sar
```

The following example archives the project file (.prj) only for selected .srs files into the specified sar file. Any input source files that are in the project are not included.

```
project -archive -project c:/proj1.prj -archive_type customize
      -add_srs -no_input -archive_file c:/archive/proj1.sar
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

Project Unarchive Example

The following example extracts the project files from c:/archive/proj1.sar to directory c:/proj1. All directories and sub-directories are created if they do not already exist.

```
project -unarchive -archive_file c:/archive/proj1.sar
        -dest_dir c:/proj1
```

Project Copy Examples

The following example copies only selected .srs files for the project to the destination project file directory.

```
project -copy -project d:/test/proj_2.prj -copy_type customize
        –add_srs -no_input -dest_dir d:/test_1
```

The next example copies all input source files and .srs files selected for the project to the destination project file directory.

```
project -copy -project d:/test/proj_2.prj -copy_type customize
        -dest_dir d:/test_1
```

project_data

The project_data command shows or sets properties of a project.

Syntax

project_data {**-active** [*projectName*] | **-dir** | **-file**}

The following table describes the command options.

Option	Description
-active	Set/show active project. With no argument, shows the active project. If *projectName* is specified, changes the active project to *projectName*.
-dir	Show directory of active project.
-file	Show the project file for the active project. The full path is included with the file name.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

project_file

The project_file command manipulates and examines project files.

Syntax

project_file {**-lib** *fileName* [*libName*] | **-name** *fileName* [*newPath*] |
-time *fileName* [*format*] | **-date** *fileName* | **-type** *fileName* |
-savetype *fileName* [**relative** | **absolute**] **-move** *fileName1* [*fileName2*] |
-remove *fileName* | **-top** *topModule* |
-tooltag *applicationTagName* | **-toolargs** [*arguments*] *fileName* }

The following table describes the command options.

Option	Description
-lib	Shows the project file library associated with *fileName*. If *libName* is specified, changes the project file library for the specified file to *libName*.
-name	Shows the project file path for the specified file. If *newPath* is specified, changes t1he location of the specified project file to the directory path specified by *newPath*.
-time	Shows the file time stamp. If a *format* is specified, changes the composition of the time stamp according to the combination of the following time formatting codes: **%H** (hour 00-23) **%M** (minute 00-59) **%S** (second 00-59) **%d** (day 01-31) **%b** (abbreviated month) **%Y** (year with century)
-date	Shows the file date.
-type	Shows the file type.
-savetype	Sets or shows whether a file is saved relative to the project or its absolute path.
-move	Positions *fileName1* after *fileName2* in HDL file list. If *fileName2* is not specified, moves *fileName1* to the top of the list.
-remove	Removes the specified file from the project file list.

Option	Description
-top	Sets or shows the top-level module of the specified file for the active implementation.
-tooltag	Sets or shows the third-party tool tag for the specified file.
-toolargs	Sets or shows the third-party tool tag arguments for the specified file.

Examples

List the files added to a project. Remove a file.

```
% project -filelist path_name1/cpu.v path_name1/cpu_cntrl.v
   path_name2/cpu_cntrl.vhd

% project_file -remove path_name2/cpu_cntrl.vhd
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

project_folder

The project_folder command manipulates and examines attributes for project folders.

Syntax

project_folder [*folderName*] [**-folderlist**] [**-filelist**] [**-printout**] [**-add**] [**-remove**] [**-r**] [**-tooltag**] [**-toolargs**]

The following table describes the command options.

Option	Description
folderName	Specifies the name of the folder for which attributes are examined.
-folderlist	Lists folders contained in the specified project folder.
-filelist	Lists files contained in the specified project folder.
-printout	Prints the specified project folder hierarchy including its files.
-add	Adds a new project folder.
-remove	Removes the specified project folder.
-r	Removes the specified project folder and all its containing sub-folders. Files are removed from the project folder, but are not deleted.
-tooltag	Sets or shows the third-party tool tag name.
-toolargs	Sets or shows the additional arguments for the third-party tool tag.

Examples

Add a folder and list the files added to a project folder.

```
% project_folder -add newfolder
% project_folder -filelist newfolder
```

© 2020 Synopsys, Inc.

query_available_metrics

Shows metrics that can be queried for the design. If specified, only metrics matching the required values are shown. Otherwise, shows all metrics for all tables. You should use the query_available_metrics command primarily for scripting, since it returns a Tcl list. For a more readable format, use the dump_metrics command.

Syntax

query_available_metrics [[*table*.]*name*]

[*table*.]*name*
Name of the metric to query, optionally preceded by '*table*.'.

For details about specifying metrics, see the *Naming Conventions for Metrics, on page 52*.

Examples

This command returns values in a list of the form as follows:

{{*name1 object1 jobname1*} {*name2 object2 jobname2*}...}

1. Show a Tcl list of metrics that can be queried for the current implementation:

```
% query_available_metrics

Format is {{name1 object1 jobname1} {name2 object2 jobname2}...}:

{clock_conversion.icg_removed {} premap} {clock_conversion.icg_retained {}
premap} {clock_conversion.clean_clock_trees {} fpga_mapper}
{clock_conversion.clean_clock_pins {} fpga_mapper}
{clock_conversion.gated_clock_trees {} fpga_mapper}
{clock_conversion.gated_clock_pins {} fpga_mapper}
{clock_conversion.instances_converted {} fpga_mapper}
{clock_conversion.instances_notconverted {} fpga_mapper}
{hdl_compile.modified_files {} compiler} {hdl_compile.modified_modules {}
compiler} {hdl_compile.total_modules {} compiler} {hdl_compile.total_files
{} compiler} {misc.Part {} fpga_mapper} {runtime.realtime {} compiler}
{runtime.cputime {} compiler} {runtime.realtime {} premap}
{runtime.cputime {} premap} {runtime.realtime {} fpga_mapper}
{runtime.cputime {} fpga_mapper} {{timing.Worst Slack} {} fpga_mapper}
{utilization.LUT1 {} fpga_mapper} {utilization.LUT2 {} fpga_mapper}
{utilization.LUT3 {} fpga_mapper} {utilization.LUT4 {} fpga_mapper}
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

```
{utilization.LUT5 {} fpga_mapper} {utilization.LUT6 {} fpga_mapper}
{utilization.IBUF {} fpga_mapper} {utilization.IBUFG {} fpga_mapper}
{utilization.IOBUF {} fpga_mapper} {{utilization.I/O primitives} {}
fpga_mapper} {utilization.BUFG {} fpga_mapper} {{utilization.I/O Register
bits} {} fpga_mapper} {{utilization.Total Luts} {} fpga_mapper}
```

2. Use a simple loop to show the values of all available metrics:

```
% foreach amt [query_available_metrics] {set metric
      [lindex $amt 0]; set object [lindex $amt 1];
      set job [lindex $amt 2]; puts "$metric $object:
      [query_metric $metric -object $object -jobname $job]"}

Format is {{name1 object1 jobname1} {name2 object2 jobname2} ...}:

clock_conversion.icg_removed : 0
clock_conversion.icg_retained : 0
clock_conversion.clean_clock_trees : 1
clock_conversion.clean_clock_pins : 270
clock_conversion.gated_clock_trees : 0
clock_conversion.gated_clock_pins : 0
clock_conversion.instances_converted : 0
clock_conversion.instances_notconverted : 0
hdl_compile.modified_files : 28
hdl_compile.modified_modules : 11
hdl_compile.total_modules : 11
hdl_compile.total_files : 28
misc.Part : xc7vx485tffg1157-1
runtime.realtime : 9.881000
runtime.cputime : 9.828063
timing.Worst Slack : -0.445800
utilization.LUT1 : 31
utilization.LUT2 : 64
utilization.LUT3 : 45
utilization.LUT4 : 84
utilization.LUT5 : 57
utilization.LUT6 : 160
utilization.IBUF : 1
utilization.IBUFG : 1
utilization.IOBUF : 24
utilization.I/O primitives : 26
utilization.BUFG : 1
utilization.I/O Register bits : 0
utilization.Total Luts : 411
```

3. Show a Tcl list of metrics for the specified metric name:

```
% query_available_metrics cputime

Format is {{name1 object1 jobname1} {name2 object2 jobname2} ...}:

{runtime.cputime {} compiler} {runtime.cputime {} premap} {runtime.cputime
{} fpga_mapper}
```

4. Optionally, show a Tcl list of metrics for the specified table value:

```
% query_available_metrics runtime.cputime

Format is {{name1 object1 jobname1} {name2 object2 jobname2} ...}:

{runtime.cputime {} compiler} {runtime.cputime {} premap} {runtime.cputime
{} fpga_mapper}
```

5. Enclose within curly braces {} whenever metrics contain spaces.

```
% query_available_metrics {timing.worst slack}

Format is {{name1 object1 jobname1} {name2 object2 jobname2} ...}:

{{timing.Worst Slack} {} fpga_mapper}
```

See Also

See the related query commands below:

- dump_metrics, on page 50
- query_metric, on page 100
- query_metric_details, on page 102

query_metric

Queries specific QoR metrics for the current implementation of a design.

Syntax

query_metric *table.name* [**-object** *value*] [**-jobname** *value*]

table.name
Name of the metric to query, preceded by '*table.*'.

-object *value*
Queries metrics associated with a specific object or global metrics if not specified.

-jobName value
Queries metrics associated with a specific job name or any job name if not specified.

For details about specifying metrics, see the *Naming Conventions for Metrics*, on page 52.

Examples

Here are examples of how to query metrics for a design:

```
% query_metric clock_conversion.icg_removed

0

% query_metric clock_conversion.instances_converted -jobname
fpga_mapper

1

% query_metric runtime.realtime -jobname compiler

3.15400

% query_metric {timing.worst slack} -jobname fpga_mapper

-0.445800

% query_metric {utilization.total luts}

411
```

Suppose you have a design with compile points.

```
dump_metrics -show_queries

...
query_metric {timing.Worst Slack} -object r2p_cordic -jobname fpga_mapper
...
```

Then you can query worst slack for one of the compile points.

```
% query_metric {timing.Worst Slack} -object r2p_cordic
   -jobname fpga_mapper

-1.339500 ns {Estimated slack during compile point synthesis}
```

See Also

See the related query commands below:

- dump_metrics, on page 50
- query_available_metrics, on page 97
- query_metric_details, on page 102

© 2020 Synopsys, Inc.

query_metric_details

Queries information about a QoR metric from the current implementation. Exactly one metric must match.

Syntax

query_metric *table.name* [**-object** *value*] [**-jobname** *value*]

table.name
Name of the metric to query, preceded by '*table*.'.

-object *value*
Queries metrics associated with a specific object or a global metric if not specified.

-jobName value
Queries metrics associated with a specific job name or any job name if not specified.

For details about specifying metrics, see the *Naming Conventions for Metrics*, on page 52.

Examples

The Tcl command returns a list of three values: <*value*> <*units*> <*comment*>.

```
% query_metric_details clock_conversion.clean_clock_pins

270 {} {Number of clock pins driven by non-gated/non-generated clock
trees}

% query_metric_details clock_conversion.icg_removed

0 {} {Number of ICG latches removed}

% query_metric_details runtime.realtime -jobname compiler

3.154000 seconds {}

% query_metric_details runtime.cputime -jobname fpga_mapper

9.828063 seconds {}

% query_metric_details {timing.worst slack} -jobname fpga_mapper

-0.445800 ns {}

% query_metric_details utilization.lut1
```

```
31 {} {}
```

Suppose you have a design with compile points.

```
dump_metrics -show_queries

...
query_metric {timing.Worst Slack} -object r2p_cordic -jobname fpga_mapper
...
```

Then you can query worst slack for one of the compile points.

```
% query_metric_details {timing.Worst Slack} -object r2p_cordic
   -jobname fpga_mapper

-1.339500 ns {Estimated slack during compile point synthesis}
```

See Also

See the related query commands below:

- dump_metrics, on page 50
- query_available_metrics, on page 97
- query_metric, on page 100

Synopsys Confidential Information

recording

Allows you to record and store the Tcl commands generated when you work on your projects in the GUI. You can use this command for creating job scripts. The complete syntax for the recording command is:

recording
-**on**|**-off**
-file [*historyLogFile*]
-save [*historyLogFile*]
-state
-edit [*filename*]

In the command line:

- **-on**|**off**– turns Tcl command recording on or off. Recording mode is off by default.
- **-file** - if you specify a history log file name, this option uses the specified file in which to store the recorded Tcl commands for the current session. If you do not specify a history log name, reports the name of the current history log file.
- **-save** - if you do not specify a file name, updates the current history log. If you specify a history log file name, saves Tcl command history to the specified file.
- **-state** - returns the Boolean value of recording mode.
- **-edit -** displays the Tcl command log file in a text editor.

Examples

Turn on recording mode and save the Tcl commands in the cpu_tcl_log file created.

```
% recording -on
% recording -file cpu_tcl_log
```

report_clocks

Reports the clocks in the design database.

Syntax

report_clocks -netlist [*srsNetlistFile*] [**-csv_format**] [**-out** *fileName*]

Arguments and Options

srsNetlistFile
: The name of the srs netlist file. If this optional argument is not specified, the netlist file is taken from the active project implementation.

-csv_format
: Displays the report in spread-sheet format.

-out
: Specifies the name of the output report file (default name is *designName*_clk.rpt).

Description

The report_clocks command generates a report of the clocks found in the design database. The report includes a listing of the clock domain, parent clock, and clock type for each clock. If the -csv_format option is included, the report is output in spread-sheet format.

Examples

```
report_clocks c:/designs/mem_ctrl/mem_ctrl.srs -csv_format
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

report_messages

Queries messages from jobs based on ID or severity. This can be used to show duplicate messages that were suppressed in the log files.

Syntax

report_messages *logFile* [**-id** *value*] [**-severity** *value*] [**-out** *value*] [**-outa** *value*]

logFile
Specifies one or more log files to query for message details.

-id *value*
Restricts report to messages matching this id. Use the * or % wildcard to match any string.

-severity *value*
Limits messages to a specific severity. Can specify one or more messages as an error, warning, note, or advice. Multiple severities can be specified. If none is specified, all types are shown.

-out *value*
Name of the output file to be written rather than writing to the Tcl window.

-outa *value*
Name of the output file to be appended rather than writing to the Tcl window.

Output Format

Running the report_messages command produces a list of messages. Messages are displayed in the following format with each message beginning on a new line:

ID {*messageText*}

Examples

1. Query BN132 messages from the test.srr log file.

   ```
   report_messages test.srr -id BN132
   ```

2. Query error and warning messages from the test.srr log file.

   ```
   report_messages test.srr -severity error warning
   ```

3. Query critical warnings (CW) or downgradable errors (DE). You can search for these messages using wildcards. For example:

   ```
   report_messages test.srr -id DE*
   ```

4. Query error and warning messages from the test.srr log file. and writes output to a file called messages in the current working directory.

   ```
   report_messages test.srr -severity not warning -out messages
   ```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

report_message_summary

Retrieves a summary of the messages in the log file. This summary contains a list of the message IDs and the number of occurrences for each type, along with their message descriptions.

Syntax

report_message_summary *logFile*

logFile
Specifies a log file to query for the message summary.

Examples

```
report_message_summary test.srr
```

run_tcl

The run_tcl command lets you synthesize your project using a Tcl script file from the Tcl Script window of the synthesis tool.

Syntax

run_tcl [**-fg**] *tclFile*

You can also use the following command:

source *tclFile*

These commands are equivalent.

The following table describes the run_tcl command options.

Option	Description
-fg	Synthesizes the project in foreground mode.
TclFile	Specifies the name of the Tcl file used to synthesize the project. To create a Tcl Script file, see *Creating a Tcl Synthesis Script* , on page 515.

© 2020 Synopsys, Inc.

select

Selects specified objects.

Syntax

select *collection*
-append
-clear
-instances

The following table describes the select command options.

Option	Option
-append	Appends objects to the existing selection list.
-clear	Clears the selection list.
-instances	Selects all instances in the current view.

sdc2fdc

Translates legacy FPGA timing constraints to Synopsys FPGA timing constraints.

Syntax

sdc2fdc

Run it from the Tcl window in the synthesis tool.

See also

- *Converting SDC to FDC*, on page 164 in the *User Guide*
- *sdc2fdc Conversion*, on page 133 in the *Reference Manual*

Examples of sdc2fdc Translation

The following are examples of feedback after running the command. For information about the translated FDC file and handling the error messages, see *sdc2fdc Conversion*, on page 133 in the *Reference Manual*.

```
% sdc2fdc

INFO: Translation successful.
See:"D:/bugs/timing_88/clk_prior/scratch/FDC_constraints/rev_2
   /top_translated.fdc"
Replace your current *.sdc files with this one.

INFO: Automatically updating your project to reflect the new
constraint file(s)
Do "Ctrl+S" to save the new settings.

% sdc2fdc

ERROR: Bad -from list for define_false_path: {my_inst}
Missing qualifier(s) (i: p: n: ...)
ERROR: Translation problems were found.
See:"D:/bugs/timing_88/clk_prior/scratch/FDC_constraints/rev_2
   /top_translate.log" for details.
_translate.log

ERROR:  Bad -from list for define_false_path  {my_inst}
       Missing qualifier(s) (i: p: n: ...)
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

```
"define_false_path -from {my_inst} -to i:abc.def.g_reg
   -through {n:bar}"
Synplicity SDC source file: D:/bugs/timing_88/clk_prior/scratch
   /top.sdc.
Line number: 79
```

set_option

The set_option command sets options for the technology (device) as well as for the design project.

Syntax

set_option **-***optionName optionValue*

For syntax and descriptions of the options and related values, see one of the following tables:

- Device Options for set_option/get_option
- Project Options for set_option/get_option

Device Options for set_option/get_option

The following table lists *generic* device arguments for the technology, part, and speed grade. These are the options on the Implementation Options-> Device tab.

Information on all other Implementation Options tabs are listed in the next section, *Project Options for set_option/get_option*, on page 114.

Option Name	Description
-technology *parameter*	Sets the target technology for the implementation. *parameter* is the string for the vendor architecture. Check the Device panel in the GUI or see *Device Panel* , on page 347, for a list of supported families.
-part *part_name*	Specifies a part for the implementation. Check the Device panel of the Implementation Options dialog box (see *Device Panel* , on page 347) for available choices.
-speed_grade **-***value*	Sets the speed grade for the implementation. Check the Device panel of the Implementation Options dialog box (see *Device Panel* , on page 347) for available choices.

Synopsys Confidential Information

Option Name	Description
-package *value*	Sets the package for the implementation. This option is not available for certain vendor families, because it is set in the place-and-route software. Check the Device panel of the Implementation Options dialog box (see *Device Panel* , on page 347) for available choices.
-grade -*value*	Same as -speed_grade. Included for backwards compatibility.

In general, device options are technology-specific, or have technology-specific defaults or limitations. For vendor-specific details, see *synhooks File Syntax*, on page 527.

Project Options for set_option/get_option

Below is a list of options for the set_option and get_option commands. click the option below for the corresponding description and GUI equivalents. Options set through the Device tab are listed in *Device Options for set_option/get_option*, on page 113.

analysis_constraint	areadelay
area_delay_percent	auto_constrain_io
automatic_compile_point	autosm
beta_vfeatures	block
compiler_compatible	compiler_constraint
constraint	continue_on_error
default_enum_encoding	disable_io_insertion
dup	enable64bit
fanout_limit	force_gsr
frequency auto	frequency
globalthreshold	hdl_param
hdl_define	help
identify_debug_mode	ignore_undefined_libs
include_path	job (PR)

libext	library_path
log_file	looplimit
maxfan	max_parallel_jobs
multi_file_compilation_unit	no_sequential_opt
num_critical_paths	num_startend_points
opcond	project_relative_includes
preserve_registers	report_path
reporting_reportType	resolve_multiple_driver
resource_sharing	result_file
retiming	run_prop_extract
rw_check_on_ram	safe_case
supporttypedflt	symbolic_fsm_compiler
synthesis_onoff_pragma	top_module
update_models_cp	use_fsm_explorer
vlog_std	write_apr_constraint
write_verilog	write_vhdl

Option	Description	GUI Equivalent
-analysis_constraint *path/filename*.**adc**	Specifies the analysis design constraint file (.adc) you can use to modify constraints for the stand-alone Timing Analyst only.	Constraint File section on the Timing Report Generation Parameters dialog box
-areadelay *percentValue* **-area_delay_percent** *percentValue*	Sets the percentage of paths you want optimized. This option is available only in certain device technologies.	Percent of design to optimize for timing, Device Panel

© 2020 Synopsys, Inc.

Option	Description	GUI Equivalent
-automatic_compile_point 1|0	Enables/disables the automatic compile point flow, which can analyze a design and identify modules that can automatically be defined as compile points and mapped in parallel using Multiprocessing.	Automatic compile point check box, Options Panel
-autosm 1|0 **-symbolic_fsm_compiler 1|0**	Enables/disables the FSM compiler.	FSM Compiler check box, Options Panel
-beta_vfeatures 1|0	Enables/disables the use of Verilog compiler beta features.	Beta Features for Verilog, Verilog Panel
-block 1|0 **-disable_io_insertion 1|0**	Enables/disables I/O insertion in some technologies.	Disable I/O Insertion check box, Device Panel
-compiler_compatible 1|0	Disables pushing of tristates across process/block boundaries.	*Complement* of the Push Tristates Across Process/ Block Boundaries check box, VHDL Panel and Verilog Panel
-compiler_constraint *constraintFile*	When multiple constraint files are defined, specify which constraint files are to be used from the Constraints tab of the Implementation Options panel.	Constraints Files, Constraints Panel
-constraint -*option*	Manipulates constraint files in the project: -enable/disable *filename* - adds or removes constraint file from active implementation -list - lists all enabled constraint files in active implementation -all - enables all constraint files in active implementation -clear - disables all constraint files in active implementation	Constraint Files, Constraints Panel

Option	Description	GUI Equivalent
-continue_on_error 1|0	The continue_on_error option serves two related, but separate functions. **Mapper** - When enabled during compile-point synthesis, allows the mapping operation to continue on error and synthesize the remaining compile points. The default for this option (0) is to stop on any compilation or synthesis error.	Continue on Error, Project View checkbox or Options Panel, or Configure Parallel or Compile Point Process Command
-default_enum_encoding default|onehot|gray | sequential	(VHDL only) Sets the default for enumerated types.	Default Enum Encoding, VHDL panel (see VHDL Panel and Verilog Panel)
-disable_io_insertion 1|0 **-block 1|0**	Enables/disables I/O insertion in some technologies. For more information about the impact of using this command, see *syn_insert_pad* , on page 107.	Disable I/O Insertion, Device Panel
-dup	For Verilog designs, allows the use of duplicate module names. When true, the last definition of the module is used by the software and any previous definitions are ignored. You should not use duplicate module names in your Verilog design, therefore, this option is disabled by default. However, if you need to, you can allow for duplicate modules by setting this option to 1. Recommended to use -allow_duplicate_modules option instead of -dup option.	Allow Duplicate Modules, Verilog Panel
-enable64bit 1|0	Enables/disables the 64-bit mapping switch. When enabled, this switch allows you to run client programs in 64-bit mode, if available on your system.	Enable 64-bit Synthesis, Options Panel

© 2020 Synopsys, Inc.

Option	Description	GUI Equivalent
-fanout_limit *value* **-maxfan** *value*	Sets the fanout limit guideline for the current project.	Fanout Guide, Device Panel
-frequency *value*	Sets the global frequency.	Frequency, Constraints Panel
-frequency auto	Enables/disables auto constraints.	Auto Constrain, Constraints Panel
-hdl_define	For Verilog designs; used for extracting design parameters and entering compiler directives.	Compiler Directives and Design Parameters, Verilog Panel
-**hdl_param**	Shows or sets HDL parameter overrides. See *hdl_param* , on page 68 for command syntax.	Use this command in the Tcl window of the UI.
-help	This option is useful for getting syntax help on the various implementation options used for compiling and mapping a design. For examples, see *help for set_option* , on page 128.	Use this command in the Tcl window of the UI.
-identify_debug_mode 1|0	When set option is 1, creates an Identify implementation in the Project view. Then, you can launch the Identify Instrumentor or Debugger from within the FPGA synthesis tools.	Select the Identify implementation, then launch: • Launch Identify Instrumentor or • Launch Identify Debugger

Option	Description	GUI Equivalent
-ignore_undefined_libs 1|0	(VHDL only) When enabled (default), the compiler will ignore any declared library files not included with the source file. In previous releases, the missing library file would cause the synthesis tool to error out. To set this option to error out when a library file is missing (as in previous releases), use 0 for the command value.	Not available in the UI
-include_path *path* **./extra_input/**	(Verilog only) Defines the search path used by the ‘include commands in Verilog design files. Argument *path* is a string that is a semicolon-delimited list of directories where the included design files can be found. The software searches for include files in the following order: • First, the source file directory. • Then, looks in the included path directory order and stops at the first occurrence of the included file it finds. • Finally, the project directory. The include paths are relative. Use the project_relative_includes option to update older project files. The archive utility allows you to add the extra_input directory path for all include files and copies them to your project. Use the Add extra input path to project option on the Un-Archive Utility dialog box.	Include Path Order, Verilog panel (see Verilog Panel)
-job *PR_job_name* **-option enable_run 1|0**	If enabled, runs the specified place-and-route job with the appropriate vendor-specific place-and-route tool after synthesis.	Specify the place-and-route job you want to run for the specified implementation. See Place and Route Panel.

Synopsys Confidential Information

Option	Description	GUI Equivalent
-libext *.libextName1 .libextName2 ...*	Adds library extensions to Verilog library files included in your design for the project and searches the directory paths you specified that contain these Verilog library files. To use library extensions, see *Using Library Extensions for Verilog Library Files* , on page 45 in the *User Guide*.	Library Extensions (space separated) for each unique file extension, Verilog Panel.
-library_path *directory_pathname*	For Verilog designs, specifies the paths to the directories which contain the library files to be included in your design for the project. Defines the search path used by the tool to include all the Verilog design files for your project. The argument *directory_pathname* is a string that specifies the directories where these included library files can be found. The software searches for all included Verilog files and the tool determines the top-level module. The names of files read from the library path must match module names. Mismatches result in error messages.	Library Directories on Verilog Panel.
-log_file *logFileName*	Allows you to change the name for the default log file. For example: `set_option -log_file test` generates the `synlog\test_fpga_mapper.srr` file in the Implementation Directory after synthesis is run.	Enter command from the Tcl window
looplimit *loopLimitValue*	Allows you to override the default compiler loop limit value of 2000 in the HDL. You can also apply loop limits using the Verilog loop_limit or the VHDL syn_looplimit directive. For details about these directives, see *loop_limit* , on page 47 and *syn_looplimit* , on page 121 in the *Attribute Reference*.	Loop Limit, Verilog Panel and VHDL Panel.

Option	Description	GUI Equivalent
-maxfan *value* **-fanout_guide** *value*	Sets the fanout limit for the current project. The limit is value is guideline for the tool rather than a hard limit.	Fanout Guide, Device Panel
max_parallel_jobs *n*	Lets you run multiprocessing with compile points. This allows the synthesis software to run multiple, independent compile point jobs simultaneously, providing additional runtime improvements for the compile point synthesis flow. For information on setting the maximum number of parallel synthesis jobs, see *Setting Number of Parallel Jobs* , on page 513 in the *User Guide*.	Maximum number of parallel mapper jobs, on the Configure Compile Point Process dialog box.
-multi_file_compilation_unit 1|0	When you enable the Multiple File Compilation Unit switch, the Verilog compiler uses the compilation unit for modules defined in multiple files.	Verilog Panel
-no_sequential_opt 1|0	Enables or disables the sequential optimizations for the design. (Note that unused registers will still be removed from the design.) The default value is false (sequential optimizations are performed). When true, delay and area size might increase. Value can be 1 or true, 0 or false. With this option enabled, the FSM Compiler and FSM Explorer options are effectively disabled.	Device Panel
-num_critical_paths *value*	Specifies the number of critical paths to report in the timing report.	Number of Critical Paths, Timing Report Panel

Synopsys Confidential Information

Option	Description	GUI Equivalent
-num_startend_points *value*	Specifies the number of start and end points to include when reporting paths with the worst slack in the timing report.	Number of Start/End Points, Timing Report Panel. Number of Start/End Points, Timing Report Generation dialog box.
-opcond *value*	*Microchip technologies* Sets the operating condition for device performance in the areas of optimization, timing analysis, and timing reports. Values are Default, MIL-WC, IND-WC, COM-WC, and Automotive-WC. See *Operating Condition Device Option* , on page 412 for more information.	Device Panel
-preserve_registers 1|0	*Microchip* When enabled, the software uses less restrictive register optimizations during synthesis if area is not as great a concern for your device. The default for this option is disabled (0).	Conservative Register Optimization switch on the Device Panel
-project_relative_includes 1|0	Enables/disables the Verilog include statement to be relative to the project, rather than a verilog file. For projects built with software after 8.0, the include statement is no longer relative to the files but is relative to the project: project_relative (1). See *Updating Verilog Include Paths in Older Project Files* , on page 68 in the *User Guide* for information about updating older project files.	Include Path Order, Verilog Panel
-report_path *integer*	*Microchip technologies* Sets the maximum number of critical paths in a forward-annotated SDF constraint file	Max Number of Critical Paths in SDF, Device Panel

© 2020 Synopsys, Inc.

Option	Description	GUI Equivalent
-**reporting_***reportType*	Sets parameters for the stand-alone Timing Analyst report. See *Timing Report Parameters for set_option* , on page 126 for details.	Analysis->Timing Analyst command: Timing Report Generation Parameters
-resolve_multiple_driver 1|0	When a net is driven by a VCC or GND and active drivers, enable this option to connect the net to the VCC or GND driver. The default for this option is disabled (0). See *Resolve Mixed Drivers Option* , on page 129 for details.	Resolve Multiple Drivers, Device Panel
-resource_sharing 1|0	Enables or disables resource sharing globally. This is a compiler-specific optimization, and does not affect resource sharing in the mapper. To enable or disable individual modules, use the syn_sharing directive.	Resource Sharing, Device Panel
-result_file *filename*	Specifies the name of the results file.	Result File Name and Result Format, Implementation Results Panel
-retiming 1|0	When enabled (1), registers may be moved into combinational logic to improve performance. The default value is 0 (disabled).	Retiming, Device Panel
-run_prop_extract 1|0	Enables/disables the annotation of certain generated properties relating to clocks and expansion onto the RTL view. This enables the Tcl expand and find commands to work correctly with clock properties.	Options Panel

© 2020 Synopsys, Inc.

Option	Description	GUI Equivalent
-rw_check_on_ram 1|0	Enabling this option automatically inserts bypass logic when required to prevent simulation mismatch in read-during-write scenarios. For asynchronous clocks, the tool will not generate bypass logic which can cause unintended CDC paths between the clocks. For more information about using this option in conjunction with the syn_ramstyle attribute, see *syn_ramstyle* , on page 199.	Automatic Read/Write Check Insertion for RAM, Device Panel
-safe_case 1|0	When enabled, the high reliability safe case option turns off sequential optimizations for counters, FSM, and sequential logic to increase the circuit's reliability.	Preserve and Decode Unreachable States (FSM, Counters, Sequential Logic), High Reliability Panel
-supporttypedflt 1|0	When enabled (1), the compiler passes init values through a syn_init property to the mapper. For more information, see *VHDL Implicit Data-type Defaults* , on page 336.	Implicit Initial Value Support, VHDL Panel

Option	Description	GUI Equivalent
-symbolic_fsm_compiler 1|0 **-autosm 1|0**	Enables/disables the FSM compiler. Controls the use of FSM synthesis for state machines. The default is false (FSM Compiler disabled). Value can be 1 or true, 0 or false. When this option is true, the FSM Compiler automatically recognizes and optimizes state machines in the design. The FSM Compiler extracts the state machines as symbolic graphs, and then optimizes them by re-encoding the state representations and generating a better logic optimization starting point for the state machines. However, if you turn off sequential optimizations for the design, FSM Compiler and/or the syn_state_machine directive and syn_encoding attribute are effectively disabled. See -no_sequential_opt 1|0 for more information on turning off sequential optimizations.	FSM Compiler check box, Device Panel
-synthesis_onoff_pragma 1|0	Determines whether code between synthesis on/off directives is ignored. When enabled, the software ignores any VHDL code between synthesis_on and synthesis_off directives. It treats these third-party directives like translate_on/ off directives (see *translate_off/translate_on* , on page 283 for details).	Synthesis on/off Implemented as Translate on/Off, VHDL Panel

Option	Description	GUI Equivalent
-top_module *name*	Specifies the top-level module. If the top-level entity does not use the default work library to compile the VHDL/Verilog files, you must specify the library file where the top-level entity can be found. To do this, the top-level entity name must be preceded by the VHDL/Verilog library followed by the dot (.).	Top-level Entity/Module, VHDL Panel or Verilog Panel
-update_models_cp 1|0	Determines whether (1) or not (0) changes inside a compile point can cause the compile point (or top-level) containing it to change accordingly.	Update Compile Point Timing Data, Device Panel
-use_fsm_explorer 1|0	Enables/disables the FSM Explorer.	FSM Explorer, Device Panel
-vlog_std v2001 | v95 | sysv	The default Verilog standard for new projects is SystemVerilog. Turning off both options in the Verilog panel defaults to v95.	Verilog 2001, SystemVerilog, Verilog Panel
-write_apr_constraint 1|0	Writes vendor-specific constraint files.	Write Vendor Constraint File, Implementation Results Panel
-write_verilog 1|0 **-write_vhdl 1|0**	Writes Verilog or VHDL mapped netlists.	Write Mapped Verilog/VHDL Netlist, Implementation Results Panel

Timing Report Parameters for set_option

The following lists the parameters for the stand-alone timing report (.ta file).

async_clock
ctd
filename
filter
gen_output_srm
margin
netlist
number_paths
output_srm

Reporting Option	Description
-reporting_async_clock	Generates a report for paths that cross between clock groups using the stand-alone Timing Analyst.
-reporting_ctd slack | end_point | off	Controls how the *design*_ctd.txt (correlation timing dump) file is generated when the Timing Analyst is run. You can specify one of the following values: • slack - The timing information in the ctd file is sorted by slack. This is the default. • end_point - The timing information in the ctd file is sorted by end points. • off - Turns off generating the ctd file. The ctd file contains a timing summary of the design that is used by the Timing Report View to display and analyze the synthesis timing for the design and correlate this synthesis timing with the P&R timing in the GUI.
-reporting_filename *filename*.**ta**	Specifies the standard timing report file (ta) generated from the stand-alone Timing Analyst.
-reporting_filter *filter options*	Generates the standard timing report based on the filter options you specify for paths, such as: • From points • Through points • To points For more information, see: • *Timing Report Generation Parameters* , on page 401. • *Combining Path Filters for the Timing Analyzer* , on page 405 • *Timing Analyzer Through Points* , on page 404. • *Specifying From, To, and Through Points* , on page 250.

© 2020 Synopsys, Inc.

Reporting Option	Description
-**reporting_gen_output_srm 1|0**	Specifies the new name of the output SRM File when you change the default name. If this option is set to 1, this new name is used for the output `srm` file after you run the stand-alone Timing Analyst.
-reporting_margin *slackValue*	You can specify a slack margin to obtain a range of paths within the worst slack time for the design after you run the stand-alone Timing Analyst.
-reporting_netlist *filename*.**srm**	Specifies the associated gate-level netlist file (`srm`) generated from the stand-alone Timing Analyst.
-reporting_number_path *numberOfPaths*	You can specify the number of critical paths to report after you run the stand-alone Timing Analyst.
-reporting_output_srm 1|0	Allows you to change the name of the output `srm` file. If you enable the output SRM File option, you can change this default name.

For GUI equivalent switches for these parameters, see *Timing Report Generation Parameters*, on page 401.

help for set_option

This option is useful for getting syntax help on the various implementation options used for compiling and mapping a design, especially since this list of options keeps growing.

Syntax

```
% set_option -help
```

Usage:

set_option *optionName optionValue* [**-help** [*value*]]

Where:

- *optionName*—specifies the option name.
- *optionValue*—specifies the option value.

- -help [*value*]—to get help on options. Use:
  - -help * for the list of options
  - -help *optionName* for a description of the option

Examples

To list all option commands in the Tcl window:

```
set_option -help *
```

To list all option commands beginning with the letters fi in the Tcl window:

```
% set_option -help fi*

fixgatedclocks
fixgeneratedclocks
fixsmult
```

To get help on a specific option in the Tcl window:

```
% set_option -help fixgatedclocks

0: Don't fix; 1: fix, no report; 2: fix, report exception
registers; 3: fix, report all registers
```

Use the following Tcl commands to print a description of the options:

```
% set_option -help c*
% set hl [set_option -help c*]
% puts $hl
% foreach option $hl { puts "$option:\t [set_option -help
$option]"; }
```

This example will print a list of set_option options that begin with the letter c.

Resolve Mixed Drivers Option

Use the Resolve Mixed Drivers option when mapping errors are generated for input nets with mixed drivers. You might encounter the following messages in the log file:

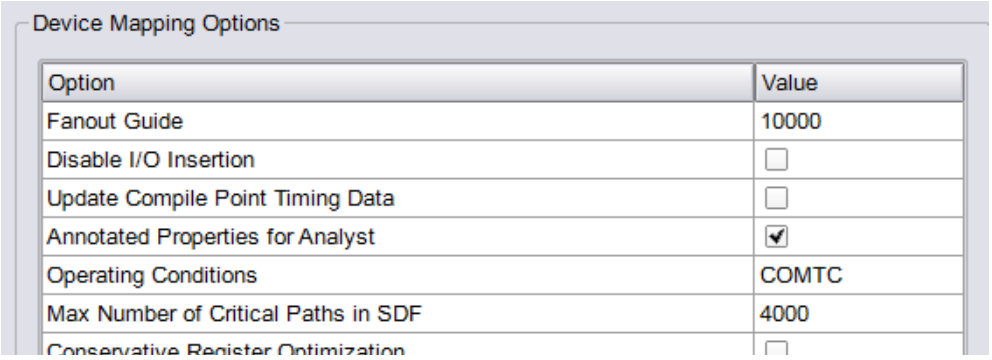
```
@A:BN313 | Found mixed driver on pin pin:data_out inst:dpram_lut3
of work.dpram(verilog), use option "Resolve Mixed Drivers" in
"Device" tab of "Implementation Options" to automatically resolve
this
@E:BN314 | Net "GND" in work.test(verilog) has mixed drivers
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

```
@A:BN313 | Found mixed driver on pin pin:Q[0] inst:dff1.q of
PrimLib.sdffr(prim), use option "Resolve Mixed Drivers" in
"Device" tab of "Implementation Options" to automatically resolve
this
@E:BN314) | Net "VCC" in work.test(rtl) has mixed drivers
```

Whenever a constant net (GND or VCC) and an active net are driving the same output net, enable the Resolve Mixed Drivers option so that synthesis can proceed. To set this switch:

- Check Resolve Mixed Drivers on the Device tab of the Implementation Options panel.



- Use the Tcl command, `set_option -resolve_multiple_driver 1`.

  By default this option is disabled and set to:

  `set_option -resolve_multiple_driver 0.`

When you rerun synthesis, you should now see messages like the following in the log file:

```
@W:BN312 | Resolving mixed driver on net GND, connecting output
pin:data_out inst:dpram_lut3 of work.dpram(verilog) to GND
@N:BN116) | Removing sequential instance dpram_lut3.dout of
view:PrimLib.dffe(prim) because there are no references to its
outputs
@N:BN116 | Removing sequential instance dpram_lut3.mem of
view:PrimLib.ram1(prim) because there are no references to its
outputs
```

```
@W:BN312 | Resolving mixed driver on net VCC, connecting output
pin:Q[0] inst:dff1.q of PrimLib.sdffr(prim) to VCC
@N:BN116 | Removing sequential instance dff1.q of
view:PrimLib.sdffr(prim) because there are no references to its
outputs
```

Example - Active Net and Constant GND Driving Output Net (Verilog)

```
module test(clk,data_in,data_out,radd,wradd,wr,rd);
input clk,wr,rd;
input data_in;
input [5:0]radd,wradd;
output data_out;
// component instantiation for shift register module
shrl srl_lut0 (
   .clk(clk),
   .sren(wr),
   .srin(data_in),
   .srout(data_out)
   );
// Instantiation for ram
dpram dpram_lut3 (
   .clk(clk),
   .data_in(data_in),
   .data_out(data_out),
   .radd(radd),
   .wradd(wradd),
   .wr(wr),
   .rd(rd)
   );

endmodule

module shrl (clk,sren,srin,srout);
input clk;
input sren;
input srin;
output srout;

parameter width = 32;
reg [width-1:0] sr;

always@(posedge clk)
begin
   if (sren == 1)
   begin
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

```
      sr <= {sr[width-2:0], srin};
   end
end
// Constant net driving

// the output net
assign srout = 1'b0;
endmodule

module dpram(clk,data_in,data_out,radd,wradd,wr,rd);
input clk,wr,rd;
input data_in;
input [5:0]radd,wradd;
output data_out;

reg dout;
reg [0:0]mem[63 :0];

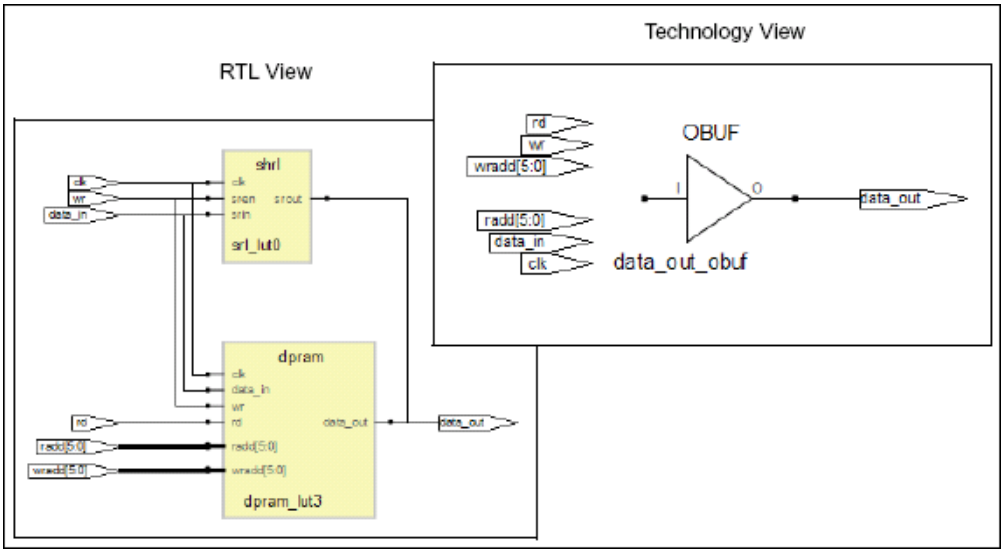
always @ (posedge clk)
begin
   if(wr)
      mem[wradd] <= data_in;
end

always @ (posedge clk)
begin
   if(rd)
      dout <= mem[radd];
   end

assign  data_out = dout;

endmodule
```

See the following RTL and Technology views; the Technology view shows the constant net tied to the output.



Example - Active Net and Constant VCC Driving Output Net (VHDL)

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

entity test is
port (clk,rst : in std_logic;
      sr_en : in std_logic;
      data : in std_logic;
      data_op : out std_logic );
end entity test;

architecture rtl of test is
component shrl
generic (sr_length : natural);
port (clk : in std_logic;
      sr_en : in std_logic;
      sr_ip : in std_logic;
      sr_op : out std_logic );
end component shrl;

component d_ff
port (data, clk, rst : in std_logic;
      q : out std_logic );
end component d_ff;
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

```
begin
-- instantiation of shift register
shift_register : shrl
generic map (sr_length => 64)
port map (clk => clk,
          sr_en => sr_en,
          sr_ip => data,
          sr_op => data_op );
-- instantiation of flipflop
dff1 : d_ff
port map (data => data,
          clk => clk,
          rst => rst,
          q => data_op );
end rtl;

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

entity shrl is
generic (sr_length : natural);
port (clk : in std_logic;
      sr_en : in std_logic;
      sr_ip : in std_logic;
      sr_op : out std_logic );
end entity shrl;

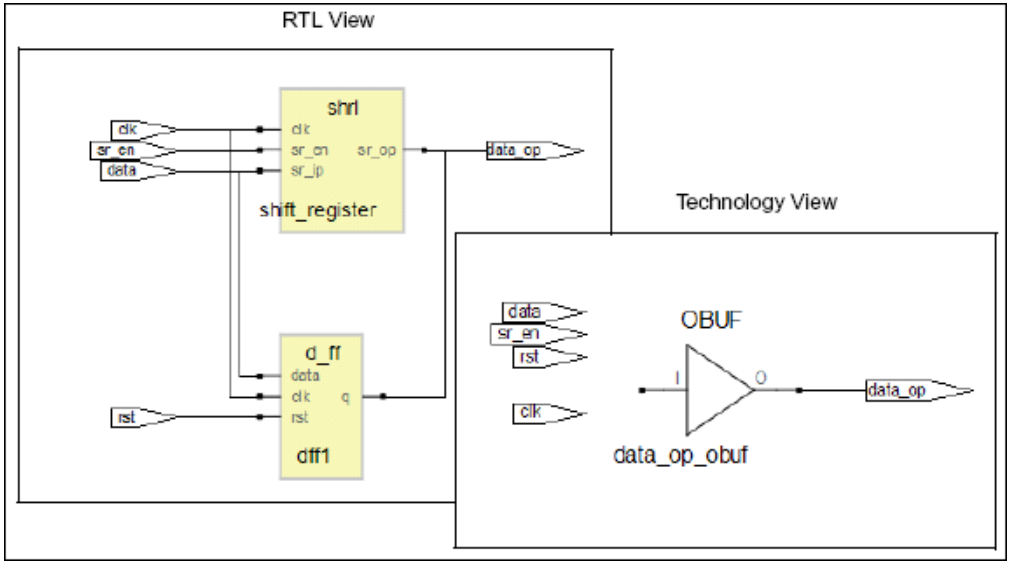
architecture rtl of shrl is
signal sr_reg : std_logic_vector(sr_length-1 downto 0);
begin
   shreg_lut: process (clk)
   begin
      if rising_edge(clk) then
         if sr_en = '1' then
            sr_reg <= sr_reg(sr_length-2 downto 0) & sr_ip;
         end if;
      end if;
   end process shreg_lut;
-- Constant net driving output net
sr_op <= '1';
end architecture rtl;

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

```
entity d_ff is
port (data, clk, rst : in std_logic;
      q : out std_logic );
end d_ff;

architecture behav of d_ff is
begin
   FF1:process (clk) begin
      if (clk'event and clk = '1') then
         if (rst = '1') then
            q <= '0';
         else q <= data;
         end if;
      end if;
   end process FF1;
end behav;
```

See the following RTL and Technology views; the Technology view shows the constant net tied to the output.



© 2020 Synopsys, Inc.

status_report

Writes out the results of reports displayed in the Project Status view after synthesizing a design.

Syntax

status_report -name *reportName* [**-parameter** *reportSectionName*] [**-csv**] [**-output_file** *fileName*] [**-msgtype** *msgStatus*] [**-status**] [**-help**]

Examples

```
status_report -name area_report

status_report -name timing_report -csv -output_file reports

status_report -name area_report -parameter io_port

status_report -name run_status -msgtype warnings

status_report -name timing_report -help
```

Option	Description
-name *reportName*	The type of report to access. Use any of the following keywords for *reportName*: • area_report • timing_report • opt_report • cp_report • hier_area_report • run_status
-parameter *reportSectionName*	Specifies a specific section of the area, timing, or message reports to access. See *Parameters* , on page 137 for details of the appropriate keywords to use for the section names.
-csv	Generates the report as a comma-separated list.
-output_file *fileName*	Specifies the name of the file that writes out the report. If you do not specify an output file, the report is displayed in the Tcl window.

© 2020 Synopsys, Inc.

Option	Description
-msgtype *msgStatus*	Generates the number of messages found for the following types of messages: • Errors • Warnings • Notes
-status	Generates the status for a job. The results of the job can be specified with one of the following conditions: • Completed • Failed
-help	Allows you to get help on a parameter list. Use -help * for a list of parameters.

Parameters

For the area, timing, and message reports you can report results for specific sections by specifying the appropriate keywords for the -parameter argument.

Area Report section keywords for -parameter	io_port non_io_reg total_io_reg v_ram dsp_used total_luts
Timing Report section keywords for -parameter	clock_name req_freq est_freq slack
Run Status section keywords for -parameter	compiler premap fpga_mapper

© 2020 Synopsys, Inc.
Synopsys Confidential Information

For example:

```
% status_report -name area_report
I/O ports(io_port) 26
Non I/O Register bits(non_io_reg)242 (0%)
I/O Register bits(total_io_reg) 24
Block Rams(v_ram) 0 (1030)
DSP Blocks(dsp_used) 1 (2800)
LUTs(total_luts) 310 (0%)
```

Additional Reporting Commands

There are other commands available from the command line to report commonly-required information: report_timing_summary, report_area, and report_opt.

- Report Timing Summary

  ```
  % report_timing_summary
  Timing Summary
  Clock Name Req Freq Est Freq Slack
  eight_bit_uc|clock 198.9 MHz 169.1 MHz-0.887
  ```

- Report Area

  ```
  % report_area
  LUTs for combinational functions0
  Non I/O Registers 0
  I/O Pins 66
  I/O registers 0
  DSP Blocks 0 (256)
  Memory Bits 32768
  ```

- Report Optimizations

  ```
  % report_opt
  Combined Clock Conversion 1 / 0
  ```

Messages Reporting Commands

Here are examples of commands available from the command line to report message information using the option: run_status.

```
%status_report -name run_status
{compiler {notes "8"}{warnings "0"}{errors "0"}
   {job_status "Completed"}}
{fpga_mapper {notes "46"}{warnings "1"}{errors "0"}
   {job_status "Completed"}}
{premap {notes "3"}{warnings "2"}{errors "0"}
   {job_status "Completed"}}

% status_report - name run_status -msgtype warnings
{compiler {warnings "0"}}
{fpga_mapper {warnings "10"}}
{premap {warnings "0"}}

% status_report - name run_status -parameter {compiler premap}
   -msgtype warnings
{compiler {warnings "0"}}
{premap {warnings "0"}}

% status_report -name run_status -parameter compiler -status
{compiler {job_status "Completed"}}

% status_report -name run_status -parameter premap -status
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

sub_impl

Sub-implementation editing command.

Syntax

sub_impl
implName **-add** *jobType*
implName **-remove**
implName **-type**
implName **-run** *mode*
implName -**cancel**
implName -**option** *optionName* [*optionValue*]
-list

Arguments and Options

Option	Description
implName **-add** *jobType*	Creates a new sub-implementation that belongs to an active implementation.
implName **-remove**	Removes an active sub-implementation.
implName **-type**	Lists a sub-implementation type.
implName **-run** *mode*	Runs sub-implementation.
implName **-cancel**	Cancels a running sub-implementation.
implName **-option** *optionName* [*optionValue*]	Sets option for sub-implementation.
-list	Lists all sub-implementations for an active implementation.

synplify_pro

Starts the FPGA synthesis tool and runs synthesis from the command line. Use the appropriate command for the tool you are using. The command to start the synthesis tool from the command line includes a number of command line options.

Syntax

synplify_pro
[*options* **...**]
[*projectFile*]

projectFile	Specifies the project (`prj`) file to use. If no file is specified, the tool defaults to the last project file opened.
options	Any of the command line options described in the next table. These options control tool action on startup and, in many cases, can be combined on the same command line. See the next table for a description of the *options* you can specify.

The following table describes the *options* you can specify:

Option	Description
-batch	*Synplify Pro (except node-locked)* Starts the synthesis tool in batch mode from the specified project or Tcl file without opening the Project window.
-compile	Compiles the project, but does not map it.
-evalhostid	Reports host ID for node-locked and floating licenses.
-help	Lists available command line options and descriptions.
-history *filename*	Records all Tcl commands and writes them to the specified history log file when the command exits.
-Identify_dir *dir*	Specifies the location of the Identify installation directory for launching the Identify tool set. The installation path specified appears in the Configure Identify Launch dialog box (Options->Configure Identify Launch).
-impl *impName*	Runs only the specified implementation. You can use this option in conjunction with the -batch keyword. The Synplify tool supports only a single implementation.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Option	Description
-ip_license_wait *waitTime*	Specifies how long to wait for a Synopsys DesignWare IP license when one is not immediately available. If you do not specify the -ip_license_wait option, license queuing is not enabled. If all requested licenses are checked out or if the specified wait time elapses, the tool excludes the IP and continues to process the rest of the design. Any IP block without a license is treated either as an error or a black box. License queuing allows you to wait until a license becomes available or specify a wait time in seconds. You can use this option in conjunction with the -batch keyword. For details, see *Queuing Licenses* , on page 508 in the *User Guide*. The *waitTime* value determines license queuing and sets a maximum wait time: • Undefined or 0 = Queuing off • 1 = Queuing enabled, indefinite wait time • >1 = Queuing enabled for the specified time
-license_release	Releases FPGA synthesis licenses for a session after the place-and-route job is launched. The software allows place and route to continue running even after exiting the synthesis tool so that it does not consume an FPGA license. This command option must be run in batch mode. Specify the following command: `toolName -batch -license_release` For details, see *Releasing the Synthesis License During Place and Route* , on page 554.
-licensetype *featureName*	Specifies a license if you work in an environment with multiple Synopsys FPGA licenses. You can use this option in conjunction with the -batch keyword. If you have licenses for multiple products, separate each feature license by a colon so that licenses can be searched in the order they are read until an available license is found.

Option	Description
-license_wait *waitTime*	Specifies how long to wait for a Synopsys FPGA license. If you do not specify the -license_wait option, license queuing is not enabled. License queuing allows you to wait until a license becomes available or specify a wait time in seconds. You can use this option in conjunction with the -batch keyword. For details, see *Queuing Licenses* , on page 508 in the *User Guide*. The *waitTime* value determines license queuing and sets a maximum wait time in seconds: • Undefined or 0 = Queuing off • 1 = Queuing enabled, indefinite wait time • >1 = Queuing enabled for the specified wait time
-log *filename*	Writes all output to the specified log file.
-loga *filename*	Writes all output to be appended to the specified log file.
-max_parallel_jobs *value*	Specifies the maximum number of concurrent processes used for synthesis.
-nopopup	Suppresses popup dialog boxes.
-run *implName*	Runs the specified implementation in the project file.
-runall	Runs all the implementations in the project file. The Synplify tool supports only a single implementation.
-shell	Starts synthesis tool in shell mode. Note: The FPGA synthesis tools only support the -shell option on UNIX and Linux platforms.
-tcl *prjFile* | *Tclscript*	Starts the synthesis tool in the graphical user interface using the specified project or Tcl file.
-tclcmd *command*	Specifies Tcl command to be executed on startup.
-verbose_log	Writes messages to stdout.log in verbose mode.
-version	Reports version of specified synthesis tool.

Synopsys Confidential Information

Tcl Command Categories

The following tables group Tcl commands together by type or functionality.

Log File Commands, on page 144

Technology-Specific Tcl Commands, on page 144

Technology-Specific Tcl Commands, on page 144

Log File Commands

These Tcl commands let you filter messages in the log file.

log_filter	Lets you filter errors, notes, and warning messages.
log_report	Lets you write out the results of the log_filter command to a file.
message_override	Allows you to suppress or override the log file message ID specification.

Technology-Specific Tcl Commands

You can find vendor-specific Tcl commands in the appropriate vendor chapter.

Vendor/Family	Tcl Command Lists
Microchip	*Microchip Tcl set_option Command Options* , on page 416

SYNOPSYS®
Silicon to Software™

CHAPTER 3

Tcl Find, Expand, and Collection Commands

The FPGA synthesis software includes powerful search functionality in the Tcl find and expand commands. Objects located by these commands can be grouped into collections and manipulated. The following sections describe the commands and collections in detail:

- find, on page 147
- find -filter, on page 154
- expand, on page 161
- Collection Commands, on page 164
- Query Commands, on page 174
- Synopsys Standard Collection Commands, on page 204

The find, expand, and collection commands are listed alphabetically in the following table.

add_to_collection	all_clocks
all_fanin	all_fanout
all_inputs	all_outputs
all_registers	append_to_collection
c_diff	c_info
c_intersect	c_list

© 2020 Synopsys, Inc.

c_print	c_symdiff
c_union	copy_collection
define_collection	define_scope_collection
expand	find (Tcl find)
foreach_in_collection	get_cells
get_clocks	get_flat_cells
get_flat_nets	get_flat_pins
get_nets	get_object_name
get_pins	get_ports
get_prop	index_collection
object_list	remove_from_collection
report_timing	set
sizeof_collection	

find

The Tcl find command identifies design objects based on specified criteria. Use this command to locate multiple objects with a common characteristic. If you want to locate objects that share connectivity, use the expand command instead of the find command (*expand*, on page 161).

You can specify the find command from the SCOPE environment or enter it as a Tcl command. This command operates on the RTL database.

You can define objects identified by find as a group or *collection*, and operate on all the objects in the collection at the same time. To do this, you embed the find command as part of a collection creation or manipulation command to do this in a single step. The combination of find and collection commands provides you with very powerful functionality to operate on and manipulate multiple design objects simultaneously.

The table summarizes where to find detailed information:

For ...	See ...
Command syntax	*Tcl Find Syntax* , on page 148
Syntax details: object types, expressions, case sensitivity, and special characters	*Tcl Find Command Object Types* , on page 149 *Wildcards and Special Characters* , on page 150 *Tcl Find Command Case Sensitivity* , on page 150
Examples of find syntax	*Tcl Find Syntax Examples* , on page 151
Filtering find searches by property	*find -filter* , on page 154 *Find Filter Properties* , on page 155 *Refining Tcl Find Results with -filter* , on page 147 in the *User Guide*.
Using find search patterns and using find in collections	*Finding Objects with Tcl find and expand* , on page 145 in the *User Guide*.

© 2020 Synopsys, Inc.

Tcl Find Syntax

Finds design objects based on specified criteria.

Find is available as part of the HDL Analyst tool.

Syntax

find
[**-flat**]
[**-inst**]
[**-net**]
[**-port**]
[**-pin**]
[**-view**]
[**-nocase**]
[**-print**]
[**-depth** *value*]
[**-filter** *expression*]
[**-seq**]
[*pattern*]

-flat
Extends the search to all levels. The * wildcard character matches hierarchy separators as well as characters. See *Wildcards and Special Characters, on page 150* for additional information.

-inst
Finds matching instances. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

-net
Finds matching nets. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

-port
Finds matching ports. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

-pin
Finds matching pins. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

-view
Finds matching views. If no -type (-inst, -net, -port, -pin, or -view) option is set, results include instances, nets, and ports.

-nocase
The -nocase option makes the search case-insensitive.

-print
Prints the first 20 search results. For a full list of objects found, use c_print or c_list. If you use find from the shell, the results are printed to the Tcl window; if you find in the constraint file, the results are printed to the log file at the beginning of the Mapper section. Reported object names have prefixes that identify the object type and are contained in curly braces ({}).

-depth *value*
Sets the starting depth for the search. Value may be a single number or a range. When -depth with a range is used, for example `-depth 4-7`, -hier and -flat arguments are ignored.

-filter *expression*
Further refines the results of find by filtering the results using the specified object property. For syntax details, refer to *find -filter*, on page 154.

-seq
Finds sequential (clocked) instances (the -inst object type is not required). This argument is equivalent to -filter @is_sequential.

pattern
The value to search for.

Tcl Find Command Object Types

You can specify the following types of objects:

Object	Prefix	Example	Synopsys
view (Design)	v:	work.cpu.rt1|p:rst is the hierarchical rst port in the cpu view which points to all instances of cpu.	lib_cell
inst (Instance)	i:	Default object type. i:core.i_cpu.reg1 points to the reg1 instance inside i_cpu.	cell
port	p:	p:data_in[3] points to bit 3 of the primary data_in port. work.cpu.rt1|p:rst is the hierarchical rst port in the cpu view which eventually points to all instances of cpu.	port

Synopsys Confidential Information

Object	Prefix	Example	Synopsys
pin	t:	t:core.i_cpu.rst points to the hierarchical rst pin of instance i_cpu.	pin
net	n:	n:core.i_cpu.rst points to the rst net driven in i_cpu.	net
seq (Sequential instance)	i:	i:core.i_cpu.reg[7:0]	cell

Wildcards and Special Characters

The Tcl find command significantly differs from a simple Tcl search. A simple Tcl search does not treat any character, except for the backslash (\), as a special character, so * matches everything in a string. The Tcl find command uses various special characters, as shown in the following table.

Use curly brackets {} or double quotes to prevent the interpretation of special characters within a pattern, and the backslash to escape a single character.

Syntax	Matches ...
*	A sequence of 0 or more matches If you do not specify -hier, the search is restricted to the current view only. To traverse downward through the hierarchy, either use the -hier argument or specify the hierarchical levels to be searched by adding the hierarchical delimiter to the pattern. For example, *.* matches objects one level below the current view.
?	A sequence of 0 or 1 matches

Tcl Find Command Case Sensitivity

Case sensitivity depends on the rules of the language used to specify the object. In mixed-language designs, the case-sensitivity rules for the parent object prevail, even when another language is used to define the lower-level object.

Tcl Find Syntax Examples

The following are examples of find syntax:

Example	Description
find {a*}	Finds any object in the current view that starts with a
find {a*} -hier -nocase	Finds any object that starts with a or A
find -net {*synp*} -hier	Finds any net the contains synp
find -seq * -filter {@clock==myclk}	Finds any register in the current view that is clocked by myclk
find -flat -seq {U1.*}	Finds all sequential elements at any hierarchical level under U1 (* matches hierarchy separator)
find -hier -flat -inst {i:A.B.C.*} -filter @view==ram*	Finds all RAM instances starting from a submodule and all lower hierarchical levels from A downwards
find -hier-seq {*} -filter @clock_enable==ena	Finds all registers enabled by the ena signal
find -hier-seq {*} -filter @slack <{-0.0}	Finds all sequential elements with negative slack
find -hier-seq {*} -filter {@clock ==clk1}	Finds all sequential elements within the clk1 clock domain
find -hier-net {*} -filter {@fanout >20}	Finds high fanout nets that drive more than 20 destinations
find -hier-seq * -in $all_inst_coll	Finds sequential elements inside the all_inst_coll collection
find -net -regexp {[a-b].*}	Finds all nets in hierarchy a and b. This means {n:a.*} and {n:b.* ; regular expressions are only supported in the earlier standard version of the HDL Analyst and are not supported in the current version.

Use the {} characters to protect patterns that contain [] from Tcl evaluation. For example, use the following command to find instance reg[4]:

© 2020 Synopsys, Inc.
Synopsys Confidential Information

```
find -inst {reg[4]}
```

Example: Custom Report Showing Paths with Negative Slack

Use the following commands:

```
open_design implementation_a/top.srm
set find_negslack[find -hier -seq -inst {*} -filter @slack <
   {-0.0}]
c_print -prop slack -prop view $find_negslack -file negslack.txt
```

The result of running these commands is a report called negslack.txt:

```
Object Name                          slack  view
{i:CPU_A_SOC.CPU.DATAPATH.GBR[0]} -3.264  "FDE"
{i:CPU_A_SOC.CPU.DATAPATH.GBR[1]} -3.158  "FDE"
{i:CPU_A_SOC.CPU.DATAPATH.GBR[2]} -3.091  "FDE"
```

Example: Custom Report for Negative Slack FFs in a Clock Domain

The following procedure steps through the commands used to find all negative slack flip-flops with a given clock domain:

1. Create a collection that contains all sequential elements with negative slack:

   ```
   set negFF  [find -tech -hier -seq {*} -filter @slack < {-0.0}]
   ```

2. Create a collection of all sequential elements within the clk clock domain

   ```
   set clk1FF  find -hier -seq * -filter {@clock==clk1}
   ```

3. Isolate the common elements in the two collections:

   ```
   set clk1Slack [c_intersect $negFF $clk1FF]
   ```

4. Generate a report using the c_print command:

   ```
   c_print [find -hier -net * -filter @fanout>=2]
      {n:ack1_tmp}
      {n:ack2_tmp}
      ...
      {n:blk_xfer_cntrl_inst.lfsr_data[20:14]}
      {n:blk_xfer_cntrl_inst.lfsr_inst.blk_size[6:0]}
      {n:blk_xfer_cntrl_inst.lfsr_inst.clk_c}
      ...
   ```

Custom Fanout Report Example

The following command generates a fanout report:

```
% c_print -prop fanout [find -hier -net * -filter @fanout>=2]
```

This is an example of the report generated by the command:

```
Object Name                                                fanout
{n:ack1_tmp}                                                    3
{n:ack2_tmp}                                                    4
...
{n:blk_xfer_cntrl_inst.lfsr_data[14]}                           3
{n:blk_xfer_cntrl_inst.lfsr_data[15]}                           3
{n:blk_xfer_cntrl_inst.lfsr_data[16]}                           2
...
```

You can add additional information to the report, by specifying more properties. For example:

```
% c_print -prop fanout [find -hier -net * -filter @fanout>=2] -prop
pins
```

This command generates a report like the one shown below:

```
Object Name    Fanout     Pins
{n:ack1_tmp}    3         "t:word_xfer_cntrl_inst.ack1_tmp
                          t:word_xfer_inst.ack1_tmp"
{n:ack2_tmp}    4         "t:blk_xfer_cntrl_inst.ack2_tmp
                          t:blk_xfer_inst.ack2_tmp"
{n:adr_o_axb_1} 2         "t:blk_xfer_inst.adr_o_axb_1
                          t:adr_o_cry_1_0.S t:adr_o_s_1.LI"
{n:adr_o_axb_2} 2         "t:blk_xfer_inst.adr_o_axb_2
                          t:adr_o_cry_2_0.S t:adr_o_s_2.LI"
{n:adr_o_axb_3} 2         "t:blk_xfer_inst.adr_o_axb_3
                          t:adr_o_cry_3_0.S t:adr_o_s_3.LI"
{n:adr_o_axb_4} 2         "t:blk_xfer_inst.adr_o_axb_4
                          t:adr_o_cry_4_0.S t:adr_o_s_4.LI"
{n:adr_o_axb_5} 2         "t:blk_xfer_inst.adr_o_axb_5
                          t:adr_o_cry_5_0.S t:adr_o_s_5.LI"
{n:adr_o_axb_6} 2         "t:blk_xfer_inst.adr_o_axb_6
                          t:adr_o_cry_6_0.S t:adr_o_s_6.LI"
...
```

To save the report as a file, use a command like this one:

```
c_print -prop fanout [find -hier -net * -filter @fanout>=2]
   -prop pins -file prop.txt
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

find -filter

The Tcl find command includes the optional -filter option, which provides a powerful way to further refine the results of the find command and filter objects based on properties. See the following for details about the find -filter command:

- Find -filter Syntax, on page 154
- Find Filter Properties, on page 155
- Find Filter Examples, on page 159

For the Tcl find command syntax, see

Find -filter Syntax

find *pattern otherOtions* **-filter {[!]@***propertyName operator value***}**

!	Optional character to specify the negative. Include the ! character if you are checking for the absence of a property; leave it out if you are checking for the presence of a property.
@propertyName	Property name to use for filtering. The name must be prefixed with the @ character. For example, if clock is the property name, specify {@clock==myclk}.
operator	Evaluates and determines the property value used for the filter expression. For the operators you can use in the expressions, see *Filter Operators* , on page 155.
value	Property value for the property in the filter expression, when the property has a value. The value can either be an object name such as myclk in {@clock==myclk}, or a value, such as 60 in {@fanout>=60}.

When specified, the -filter option must be the last option specified for the find command.

Filter Operators

You can use the following relational operators with the -filter option:

==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=~	Matches pattern
!~	Does not match pattern

You can use the following logical operators with the -filter option:

&&	And
||	Or
!	Not

Find Filter Properties

The object properties are based on the design or constraint, and are used to qualify searches and build collections. To generate these properties, open Project->Implementation Options->Device and enable the Annotated Properties for Analyst check box. The properties display in the Tcl window when the RTL or Technology view is active. Some properties are only available in a certain view. The tool creates .sap and .tap files (design and timing properties, respectively) in the project folder.

The table below lists the common filter object properties. It does not include some vendor-specific properties. Use the table as a guide to filter the properties you want. Here is how to read the columns:

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Property Name	Property Value	HDL View	Description
Common Properties			
type	view | port | net | instance | pin	All	Specifies the type of object to be filtered from the netlist find * -filter -object -print
View Properties			
compile_point	locked	Tech	Filters the view based on compile point properties find * -view -filter @compile_point==locked -print
is_black_box	1	All	Filters the black box view find * -view -filter @is_black_box==1 -print
is_verilog	0 | 1	All	Filters the Verilog based view find * -view -filter @is_verilog=={0|1} -print
is_vhdl	0 | 1	All	Filters the VHDL based view find * -view -filter @is_vhdl=={0|1} -print
orig_inst_of	*viewName*	RTL and Tech	Filters the view based on original instance find * -view -filter @orig_inst_of==viewName -print
syn_hier	remove | flatten | soft | firm | hard	Tech	Filters the view based on the syn_hier attribute value find * -view -filter @syn_hier=={remove|flatten|soft|firm|hard} -print

Property Name	Property Value	HDL View	Description
Port Properties			
direction	input | output | inout	All	Filters the port based on port direction find * -port -filter @direction=={input|output|inout} -print
fanout	*value*	All	Filters the port based on fanout value find * -port -filter @fanout==value -print
Instance Properties			
area	*areaValue*	Tech	
arrival_time	*value*	Tech	Corresponds to worst slack
async_reset	**n:***netName*	All	
async_set	**n:***netName*	All	
clock	*clockName*	All	Could be a list if there are multiple clocks
clock_edge	rise | fall | high | low	All	Could be a list if there are multiple clocks
clock_enable	**n:***netName*	All	Highest branch name in the hierarchy, and closest to the driver
compile_point	locked	Tech	Automatically inherited from its view
hier_rtl_name	*hierInstanceName*	All	
inout_pin_count	*value*	All	
input_pin_count	*value*	All	
inst_of	*viewName*	All	
is_black_box	1 (Property added)	All	Automatically inherited from its view

Synopsys Confidential Information

Property Name	Property Value	HDL View	Description
is_hierarchical	1 (Property added)	All	
is_sequential	1 (Property added)	All	
is_combinational	1 (Property added)	All	
is_pad	1 (Property added)	All	
is_tristate	1 (Property added)	All	
is_keepbuf	1 (Property added)	All	
is_clock_gating	1 (Property added)	All	
is_vhdl	0 | 1	All	Automatically inherited from its view
is_verilog	0 | 1	All	Automatically inherited from its view
kind	*primitive* For example: inv | and |dff | mux | statemachine | ...)	All	Tech view contains vendor-specific primitives
location	(*x*, *y*)	Tech	Format can differ
name	*instanceName*	All	
orientation	N | S | E | W	Tech	
output_pin_count	*value*	All	
pin_count	*value*	All	
placement_type	unplaced | placed	All	
rtl_name	*nonhierInstanceName*	All	
slack	*value*	Tech	Worst slack of all arcs
slow	1	Tech	
sync_reset	**n:***netName*	All	
sync_set	**n:***netName*	All	
syn_hier	remove | flatten | soft | firm | hard	Tech	Automatically inherited from its view
view	*viewName*	All	

Property Name	Property Value	HDL View	Description
Pin Properties			
arrival_time	*timingValue*	Tech	
clock	*clockName*	All	Could be a list if there are multiple clocks
clock_edge	rise | fall | high | low	All	Could be a list if there are multiple clocks
direction	input | output | inout	All	
fanout	*value*	All	Total fanout (integer)
is_clock	0 | 1	All	
is_const	0 | 1	All	
is_gated_clock	0 | 1	All	Set in addition to is_clock
slack	*value*	Tech	
Net Properties			
clock	*clockName*	All	Could be a list if there are multiple clocks
is_clock	0 | 1	All	
is_gated_clock	0 | 1	All	Set in addition to is_clock
fanout	*value*	All	Total fanout (integer)

Find Filter Examples

The following examples show how find -filter is used to check for the presence or absence of a property, with the ! character indicating a negative check:

`c_print [find -hier -view {*} -filter (@is_black_box)]`	Finds all objects that are black boxes.
`c_print [find -hier -view {*} -filter (!@is_black_box)]`	Finds all objects that are not black boxes

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Positive Check Examples

Finds all ports, pins, and nets from the top level with a fanout greater than 8:

```
find * -filter {@fanout>8}
```

Finds all instances other than andv and orv in the design:

```
find * -hier -filter {!(@view=andv||@view=orv)}
```

Finds all instances of statemachine throughout the hierarchy:

```
find -hier -inst * -filter {@inst_of==statemachine}
```

```
find -hier -inst * -filter {@kind==statemachine}
```

Finds all instances throughout the hierarchy that include the string reg and are clocked by CLK:

```
find -hier -inst {*reg*} -filter {@clock==CLK}
```

Finds all nets throughout the hierarchy that have a fanout greater than 4:

```
find -hier -net {*} -filter {@fanout>4}
```

Negative Check Example

Finds all instances from the top level that have the include string big, that are not black boxes, and that have more than 10 pins:

```
find -inst *big* -filter {!@is_black_box&&(@pin_count>10)}
```

Example of Boolean Expression Specified on Multiple Properties

Finds all instances from the top level that have more than eight pins and negative slack:

```
find * -filter {(@pin_count>8)&&(@slack<0)}
```

Example of Pin Property Specified for Constants

Finds pins driven by constant 0 or constant 1:

```
find -pin *.* -filter @const==value -print
```

Where value of 1 lists all the pins that are tied to a constant.

expand

The expand command identifies objects based on their connectivity, by expanding forward from a given starting point. For more information, see *Using the Tcl expand Command to Define Collections*, on page 150 of the *User Guide*.

Tcl expand Syntax

The syntax for the expand command is as follows:

expand [**-***objectType*] [**-from** *object*] [**-thru** *object*] [**-to** *object*] [**-level** *integer*] [**-hier**] [**-leaf**] [**-seq**] [**-print**]

Argument	Description
-from *object*	Specifies a list or collection of ports, instances, pins, or nets for expansion forward from all the pins listed. Instances and input pins are automatically expanded to all output pins of the instances. Nets are expanded to all output pins connected to the net. If you do not specify this argument, backward propagation stops at all sequential elements.
-hier	Searches for the pattern from every level of hierarchy, instead of just the top level and identifies objects to be expanded based on their connectivity. The default for the current view is the top level and is defined with the define_current_design command as in the compile-point flow.
-leaf	Returns only non-hierarchical instances.
-level *integer*	Limits the expansion to N logic levels of propagation. You cannot specify more than one -from, -thru, or -to point when using this option.
-*objectType*	Optionally specifies the type of object to be returned by the expansion. If you do not specify an *objectType*, all objects are returned. The object type is one of the following: • **-instance** returns all instances between the expansion points. This is the default. • **-pin** returns all instance pins between the expansion points. • **-net** returns all nets between the expansion points. • **-port** returns all top-level ports between the expansion points.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Argument	Description

-print	Evaluates the expand function and prints the first 20 results. If you use this command from HDL Analyst, these results are printed to the Tcl window; for constraint file commands, the results are printed to the log file at the start of the Mapper section.
	For a full list of objects found, you must use c_print or c_list. Reported object names have prefixes that identify the object type. There are double quotes around each name to allow for spaces in the names. For example:
	`"i:reg1"`
	`"i:reg2"`
	`"i:\weird_name[foo$]"`
	`"i:reg3"`
	`<<found 233 objects. Displaying first 20 objects. Use c_print or c_list for all. >>`

-seq	Modifies the range of any expansion to include only sequential elements. By default, the expand command returns all object types. If you want just sequential instances, make sure to define the *object_type* with the -inst argument, so that you limit the command to just instances.
-thru *object*	Specifies a list or collection of instances, pins, or nets for expansion forward or backward from all listed output pins and input pins respectively. Instances are automatically expanded to all input/output pins of the instances. Nets are expanded to all input/output pins connected to the net. You can have multiple -thru lists for product of sum (POS) operations.
-to *object*	Specifies a list or collection of ports, instances, pins, or nets for expansion backward from all the pins listed. Instances and output pins are automatically expanded to all input pins of the instances. Nets are expanded to all input pins connected to the net. If you do not specify this argument, forward propagation stops at all sequential elements.

Tcl expand Syntax Examples

Example	Description
expand -hier -from {i:reg1} -to {i:reg2}	Expands the cone of logic between two registers. Includes hierarchical instances below the current view.
expand -inst -from {i:reg1}	Expands the cone of logic from one register. Does not include instances below the current view.
expand -inst -hier -to {i:reg1}	Expands the cone of logic to one register. Includes hierarchical instances below the current view.
expand -pin -from {t:i_and2.z} -level 1	Finds all pins driven by the specified pin. Does not include pins below the current view.
expand -hier -to {t:i_and2.a} -level 1	Finds all instances driving an instance. Includes hierarchical instances below the current view.
expand -hier -from {n:cen}	Finds all elements in the transitive fanout of a clock enable net, across hierarchy.
expand -hier -from {n:cen} -level 1	Finds all elements directly connected to a clock enable net, across hierarchy.
expand -hier -thru {n:cen}	Finds all elements in the transitive fanout and transitive fanin of a clock enable net, across hierarchy.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Collection Commands

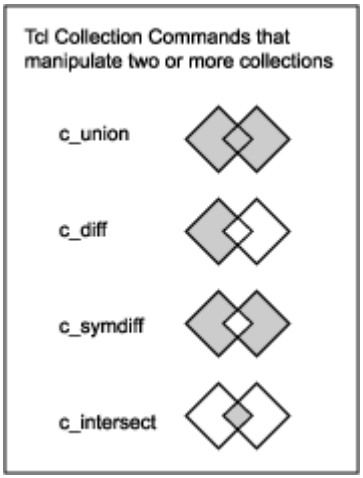
A collection is a group of objects. Grouping objects lets you operate on multiple group members at once; for example you can apply the same constraint to all the objects in a collection. You can do this from both the SCOPE editor (see *Collections*, on page 225) or in a Tcl file.

The following table lists the commands for creating, copying, evaluating, traversing, and filtering collections, and subsequent sections describe the collections, except for find and expand, in alphabetical order. For information on using collections, see *Using Collections*, on page 154 in the *User Guide*.

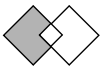
Command	Description
Creation	
define_collection	Creates a collection from a list
set modules	Creates a collection
set modules_copy $*modules*	Copies a collection
Creation from Objects Identified by Embedded Commands	
find	Does a targeted search and finds objects. Embedding the find command in a collection creation command first finds the objects, and then creates a collection out of the identified group of objects.
expand	Identifies related objects by expanding from a selected point. Embedding the expand command in a collection creation command first finds the objects, and then creates a collection out of the identified group of objects.
Operators for Comparison and Analysis	
c_diff	Identifies differences between lists or collections
c_intersect	Identifies objects common to a list and a collection
c_symdiff	Identifies objects that belong exclusively to only one list or collection
c_union	Concatenates a list to a collection

© 2020 Synopsys, Inc.

Command	Description
Operators for Evaluation and Statistics	
c_info	Prints statistics for a collection
c_list	Converts a collection to a Tcl list for evaluation
c_print	Displays collections or properties for evaluation



c_diff



Identifies differences by comparing collections, or a list and a collection. For this command to work, the design must be open in the GUI.

Syntax

c_diff {**$***collection1* **$***collection2* | **$***collection* **{***list***}**} [**-print**]

This command also includes a -print option to display the result.

Examples

The following examples combine the set with the c_diff command to create a new collection that contains the results of the c_diff command. The first example compares two collections and puts the results in diffCollection:

```
set diffCollection [c_diff $collection1 $collection2]
```

The next example creates collection1 consisting of objects i:reg1 and i:reg2, compares this collection to a Tcl list containing object i:reg1, puts the results in the collection diffCollection and prints the result (i:reg2).

```
%set collection1 {i:reg1 i:reg2}
%set diffCollection [c_diff $collection1 {i:reg1}]
%c_print $diffCollection
{i:reg2}
```

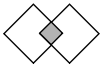
c_info

Returns specifics of a collection, including database name, number of objects per type, and total number of objects. You can save the results to a Tcl variable (array) using the -array *name* option.

Syntax

c_info $*mycollection* [**-array** *name*]

c_intersect



Defines common objects that are included in each of the collections or lists being compared.

Syntax

c_intersect $*collection1* **$***collection2* | *list* [**-print**]

This command also includes a -print option to display the result.

Example

The following example uses the set command to create a new collection that contains the results of the c_intersect command. The example compares a list to a collection (myCollection) and puts the common elements in a new collection called commonCollection:

```
%set myCollection {i:reg1 i:reg2}
%set commonCollection [c_intersect $myCollection {i:reg1 i:reg3}]
%c_print $commonCollection
   {i:reg1}
```

c_list

Converts a collection to a Tcl list of objects. You can evaluate any collection with this command. If you assign the collection to a variable, you can then manipulate the list using standard Tcl list commands like lappend and lsort. Optionally, you can specify object properties to add to the resulting list with the -prop option:

```
(object prop_value ... prop_value)...
   (object prop_value ... prop_value)
```

Syntax

c_list **$***collection*|*list* [**-prop** *propertyName*]*****

Example

```
%set myModules [find -view *]
%c_list $myModules
{v:top}{v:block_a}{v:block_b}

%c_list $myModules -prop is_vhdl -prop is_verilog
```

Name	is_vhdl	is_verilog
{v:top}	0	1
{v:block_a}	1	0
{v:block_b}	1	0

c_print

Displays collections or properties in column format. Object properties are printed using one or more -prop *propertyName* options.

Syntax

c_print {**$***collection* | **{***list***}**} [**-prop** *propertyName*]***** [**-file** *filename*] [**-append**] [**-foot** *footer*] [**-head** *header*]

To print to a file, use the `-file` option. Use -append to append to the specified file instead of overwriting it. Use the -head and the -foot options to add the header and footer respectively, before printing the file. The following command in a constraint file prints the whole collection to a file:

```
c_print -file foo.txt $col
```

The following example adds a header and a footer to the file before printing:

```
% c_print [find -inst *] -head "Search for all..." -foot "Ends
here."

Total finding took: 0 seconds
Search for all...
{i:inst1}
{i:inst2}
<<Collection has 2 objects>>
Ends here.
```

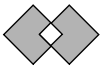
Note that the command prints the file to the current working directory. If you have multiple projects loaded, check that the file is written to the correct location. You can use the pwd command in the Tcl window to echo the current directory and then use cd directoryName to change the directory as needed.

Example

```
%set modules [find -view *]
%c_print $modules
{v:top}
{v:block_a}
{v:block_b}
```

```
%c_print -prop is_vhdl -prop is_verilog $modules
Name is_vhdl is_verilog
{v:top}0 1
{v:block_a}1 0
{v:block_b}1 0
```

c_symdiff



Compares a collection to another collection or Tcl list and finds the objects that are unique, not shared between the collections or Tcl lists being compared. It is the complement of the c_intersect command (*c_intersect*, on page 166).

Syntax

c_symdiff {**$***collection1* **$***collection2* | **$***collection* **{***list***}**} [**-print**]

This command also includes a -print option to display the result.

Examples

The following example uses the set command together with the c_symdiff command to compare two collections and create a new collection (symDiffCollection) that contains the results of the c_symdiff command.

```
set symDiff_collection [c_symdiff $collection1 $collection2]
```

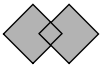
The next example is more detailed. It compares a list to a collection (collection1) and creates a new collection called symDiffCollection from the objects that are different. In this case, reg1 is excluded from the new collection because it is common to both the list and collection1.

```
set collection1 {i:reg1 i:reg2}
set symDiffCollection [c_symdiff $collection1 {i:reg1 i:reg3}]
c_list $symDiffCollection
   {"i:reg2" "i:reg3"}
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

You can also use the command to compare two collections:

c_union



Adds a collection, or a list to a collection, and removes any redundant instances. For this command to work, the design must be open in the GUI.

Syntax

c_union {**$***collection1* **$***collection2* | **$***collection* **{***list***}**} [**-print**]

The c_union command automatically removes redundant elements. This command also includes a -print option to display the result.

Examples

You can concatenate two collections into a new collection using the c_union and set commands, as shown in the following example where collection1 and collection2 are concatenated into combined_collection:

```
set combined_collection [c_union $collection1 $collection2]
```

The following example creates a new collection called sumCollection, which is generated by adding a Tcl list with one object (reg3) to collection1, which consists of reg1 and reg2. The new collection created by c_union contains reg 1, reg2, and reg3.

```
%set collection1 [find -instance {reg?} -print]
   {i:reg1}
   {i:reg2}
%set sumcollection [c_union $collection1 {i:reg3}]
%c_list $sumcollection
   {i:reg1} {i:reg2} {i:reg3}
```

If instead you added reg2 and reg3 to collection1 with the c_union command, the command removes redundant instances (reg2), so that the new collection still consists of reg1, reg2, and reg3.

```
%set collection1 {i:reg1 i:reg2}
%set sumcollection [c_union $collection1 {i:reg2 i:reg3}]
%c_list $sumcollection
   {i:reg1} {i:reg2} {i:reg3}
```

define_collection

Creates a collection from any combination of single elements, Tcl lists, and collections. You get a warning message about empty collections if you define a collection with a leading asterisk and then define an attribute for it, as shown here:

```
set noretimesh [define_collection [find -hier -seq *uc_alu]]
define_attribute {$noretimesh} {syn_allow_retiming} {0}
```

To avoid the error message, remove the leading asterisk and change `*uc_alu` to `uc_alu`.

Example

```
set modules [define_collection {v:top} {v:cpu} $mycoll $mylist]
```

define_scope_collection

The define_scope_collection command combines set and define_collection to create a collection and assigns it to a variable.

```
define_scope_collection my_regs {find -hier -seq *my*}
```

get_prop

Returns a single property value for each member of the collection in a Tcl list.

Examples

```
get_prop -prop clock [find -seq *]
```

```
get_prop -array arr [find A1] -all
```

```
get_prop $listExpandedInst -prop rtl_name LOROM32X1inst
```

```
get_prop $listExpandedInst -prop location SLICE_X1y36
```

```
get_prop $listExpandedInst -prop bel C6LUT

get_prop $listExpandedInst -prop slack 0.678
```

If this command is used in a Tcl script and the results need to be printed, use a puts command.

```
foreach cel [c_list $all_hier] {puts [get_prop -prop view $cel];}
```

set

Copies a collection to create a new collection. This command copies the collection but not the name, so the two are independent. Changes to the original collection do not affect the copied collection.

Syntax

set *collectionName collectionCriteria*

set *copyName* **$***collectionName*

collectionName	The name of the new collection.
collectionCriteria	Criteria for defining the elements to be included in the collection. Use this argument to embed other commands, like Tcl find and expand, as shown in the examples below, or other collection commands like define_collection, c_intersect, c_diff, c_union, and c_symdiff. Refer to these commands for examples.
copyName	The name assigned to the copied collection.
$*collectionName*	Name of an existing collection to copy.

Examples

The following syntax examples illustrate how to use the set command:

- Use the set command to copy a collection:

  ```
  set my_mod_copy $my_module
  ```

- Use the set command with a variable name and an embedded find command to create a collection from the find command results:

  ```
  set my_module [find -view *]
  ```

- Use the set command with define_collection to create a collection:

  ```
  set my_module [define_collection {v:top} {v:cpu} $col_l $mylist]
  ```

  For more examples of the set command used with embedded Tcl collection commands, see the examples in *c_diff*, on page 165, *c_intersect*, on page 166, *c_symdiff*, on page 169, *c_union*, on page 170, and *define_collection*, on page 171.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Query Commands

The query commands are Synopsys SDC commands from the Design Compiler® tool for creating collections of specific object types. Functionally, they are equivalent to the Tcl find and expand commands (*find*, on page 147 and *expand*, on page 161).

These query commands are intended to be used in the FDC file or the HDL Analyst view (see *Query Commands in HDL Analyst Tool*, on page 175) to create collections of objects for constraints. This section describes the syntax for the query commands supported in the FPGA synthesis tools. For complete documentation on these commands, see the Design Compiler documentation.

all_clocks
all_fanin
all_fanout
all_inputs
all_outputs
all_registers
get_cells
get_clocks
get_nets
get_pins
get_ports
object_list
report_timing

Note: Since all the query commands above are used to create Tcl collections of objects for constraints, they must be enclosed in [ ] to be applied. For example:

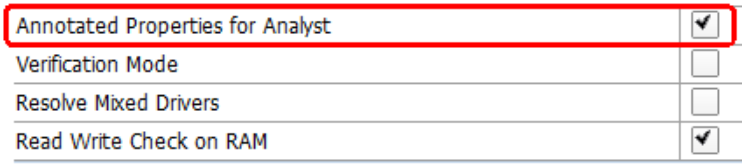
```
set_input_delay 0.5 [all_inputs] -clock clk
```

Query Commands in HDL Analyst Tool

Most of the query commands can be used in both the FDC file and the HDL Analyst view to create collections of objects for constraints. However, the all_clocks command cannot be implemented in the HDL Analyst view.

To use the query commands in the HDL Analyst RTL view:

1. Enable the option: Implementation Options->Device->Annotate Properties for Analyst.
2. If results are not as expected, check that this option is turned on during compile and before you open the SRS view.



Query Commands and Tcl find and expand Commands

The Synopsys get* commands and all* commands are functionally similar to the Tcl find and expand commands. The get* commands and all* commands are better suited to use with constraints and the fdc file, because they handle properties like @clock better than the Tcl find and expand commands. In certain cases, the .fdc file does not support the find and expand commands, although you can still enter them in the Tcl window. See *Query Commands and Tcl find and expand Commands*, on page 175 for examples.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Query and Tcl find/expand Examples

The following table lists parallel examples that compare how to use either the Tcl find/expand or the get/all commands to query design objects and set constraints.

Return the output pins of top-level registers clocked by clkb (e.g. inst1.inst2.my_reg.Q)	
all_registers	FDC Constraint: set_multicycle_path {4} -from [all_registers -no_hierarchy -output_pins -clock [get_clocks {clkb}]] set_multicycle_path {4} -from [get_pins -of_objects [get_cells * -filter {@clock == clkb}] -filter {@name == Q}]
find	Tcl Window: % define_collection [regsub -all {i:([^\s]+)} [join [c_list [find -inst * -filter @clock == clkfx]]] {t:\1.Q}]
Return all registers in the design clocked by the rising edge of clock clkfx	
all_registers	FDC Constraint: set_multicycle_path {3} -to [all_registers -cells -rise_clock [get_clocks {clkfx}]] set_multicycle_path {3} -to [get_cells -hier * -filter {@clock == clkfx && @clock_edge == rise}]
find	Tcl Window: find -hier -inst * -filter {@clock == clkfx && @clock_edge == rise}
Return clock pins of all registers clocked by the falling edge of cklfx	
all_registers	FDC Constraint: set_multicycle_path {2} -from [all_registers -clock_pins -fall_clock [get_clocks {clkfx}]] set_multicycle_path {2} -from [get_pins -of_objects [get_cells -hier * -filter {@clock == clkfx && @clock_edge == fall}] -filter {@name == C}]
find	Tcl Window: % find -hier -inst * -filter {@clock == clkfx && @clock_edge == fall}
Return the E pins of all instances of dffre cells (e.g. inst1.inst2.my_reg.E)	
get_pins	FDC Constraint: set_multicycle_path -to [get_pins -filter {@name == E} -of_objects [get_cells -hier * -filter {@inst_of == dffre}]
find	Tcl Window and FDC Constraint: % regsub -all {i:([^\s]+)} [join [c_list [find -hier -inst * -filter @inst_of == dffre]]] {t:\1.E}]

all_clocks

Use this command in the .fdc constraint file to return a collection of objects. This command is not supported in the HDL Analyst view.

Returns a collection of clocks in the current design.

Syntax

This is the supported syntax for the all_clocks command:

all_clocks

This command has no arguments. All clocks must be defined in the design before using this command. To create clocks, you can use the create_clock command.

Example

The following constraint sets a multicycle path from all the starting points.

```
set_multicycle_path 3 -from [all_clocks]
```

all_fanin

Use this command in the .fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects.

Reports pins, ports, or cells for the fanin of the specified sinks in the to list.

Syntax

This is the supported syntax for the all_fanin command:

all_fanin
[**-break_on_bboxes**]
[**-endpoints_only**]
[**-exclude_bboxes**]
[**-flat**]

[**-levels** *integer*]
[**-only_cells**]
[**-startpoints_only**]
-to *listC*
[**-trace_arcs all** | **timing**]

Arguments

-break_on_bboxes	Stops timing fanin from traversing on black boxes.
-endpoints_only	Returns only timing end points.
-exclude_bboxes	Excludes black boxes from the final result.
-flat	The fanin function operates in flat mode. This function can be specified in hierarchical (default) or flat mode. For hierarchical mode, only objects in the same hierarchy level as the current sink are returned. The pins within a level of hierarchy below the sink are traversed, but are not reported.
-levels *integer*	Stops traversal when the perimeter of the search *integer* hops is reached. For example, a level 2 hop traverses through two levels of combinational logic and stops, instead of hopping through all levels and stopping at the first sequential or port object. Counting is performed for the layers of cells that are equidistant from the sink.
-only_cells	Results include a set of all cells from the timing fanin for *listC*.
-startpoints_only	Returns only timing start points.
-to *listC*	*Required*. Reports a list of sink pins, ports, or nets in the design and the timing fanin of each sink in the *listC* Tcl list or collection specified. When you specify a net, effectively all drivers on the net are listed.
-trace_arcs all | **timing**	Specifies the type of combinational arcs to trace while traversing the fanin. You can specify either: • all - Permits tracing of all combinational arcs. This is the default. • timing - Permits tracing of valid timing arcs only.

Examples

The following examples show the timing fanin of a port in the design.

```
all_fanin -to [get_ports y*]
   {t:y_obuf[4].O t:y_obuf[3].O t:y_obuf[0].O t:y_obuf[1].O
   t:y_obuf[2].O t:y_obuf[5].O t:y_obuf[6].O t:y_obuf[7].O t:GND.G
   t:moduley_inst.y_c[0] t:moduley_inst.y_c[1]
   t:moduley_inst.y_c[2]}

all_fanin -to [get_ports y*] -startpoints_only -flat
   {t:moduley_inst.q[2].Q t:moduley_inst.q[1].Q
   t:moduley_inst.q[0].Q}

all_fanin -to [get_ports y*] -startpoints_only -flat -only_cells
   {i:moduley_inst.q[0] i:moduley_inst.q[1] i:moduley_inst.q[2]}
```

all_fanout

Use this command in the .fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects.

Returns a set of pins, ports, or cells for the fanout of the specified sources in the from list.

Syntax

This is the supported syntax for the all_fanout command:

all_fanout
[**-break_on_bboxes**]
-clock_tree | **-from** *listC*
[**-endpoints_only**]
[**-exclude_bboxes**]
[**-flat**]
[**-levels** *integer*]
[**-only_cells**]
[**-trace_arcs all** |**timing**]

Arguments

-break_on_bboxes Stops timing fanout from traversing on black boxes.

-clock_tree	Uses all clock source pins and/or ports in the design as its list of sources. Clock sources are specified with the create_clock command. If there are no clocks or clocks have no sources, then the report is empty. The -clock_tree option generates a report displaying the clock trees or networks in the design. The -clock_tree and -from options are mutually exclusive.
-endpoints_only	Returns only timing end points.
-exclude_bboxes	Excludes black boxes from the final result.
-flat	The fanout function operates in flat mode. This function can be specified in hierarchical (default) or flat mode. For hierarchical mode, only objects in the same hierarchy level as the current source are returned. The pins within a level of hierarchy below the source are traversed, but are not reported.
-from *listC*	Specifies a list of source pins, ports, or nets in the design. The timing fanout for each source of the *listC* Tcl list or collection is reported. When you specify a net, effectively all load pins on the net are listed. The -clock_tree and -from options are mutually exclusive.
-levels *integer*	Stops traversal when the perimeter of the search *integer* hops is reached. For example, a level 2 hop traverses through two levels of combinational logic and stops, instead of hopping through all levels and stopping at the first sequential or port object. Counting is performed for the layers of cells that are equidistant from the source.
-only_cells	Results include a set of all cells from the timing fanout for the *listC*.
-trace_arcs all | timing	Specifies the type of combinational arcs to trace while traversing the fanout. You can specify either: • all - Permits tracing of all combinational arcs. This is the default. • timing - Permits tracing of valid timing arcs only.

Examples

The following examples show the timing fanout of a port in the design.

```
all_fanout -from [get_ports {a*}]
   {t:a_ibuf[0].I t:a_ibuf[1].I t:a_ibuf[2].I t:hold_a.D
   t:modulex_inst.a_c[0] t:modulex_inst.a_c[1]
   t:modulex_inst.a_c[2]}

all_fanout -from [get_ports {a*}] -level 1
   {t:a_ibuf[0].I t:a_ibuf[1].I t:a_ibuf[2].I}

all_fanout -from [get_ports {a*}] -flat -endpoints_only
   {t:hold_a.D t:modulex_inst.qa[0].D t:modulex_inst.qa[1].D
   t:modulex_inst.qa[2].D t:modulex_inst.qa_fast[0].D}
```

all_inputs

Use this command in the .fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects.

Returns a collection of input or inout ports in the current design.

Syntax

This is the supported syntax for the all_inputs command:

all_inputs
[**-clock** *clockName*]
[**-exclude_clock_port**]

Arguments

-clock *clockName*	Limits the search to ports that have input delay relative to *clockName*.
-exclude_clock_port	Excludes clock ports from the search.

Examples

The following constraints set a default input delay.

```
set_input_delay 3 [all_inputs]

set_input_delay 3 -clock {clk} [all_inputs]
```

all_outputs

Use this command in the .fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects.

Returns a collection of output or inout ports in the current design.

Syntax

This is the supported syntax for the all_outputs command:

all_outputs
[**-clock** *clockName*]

Arguments

-clock *clockName*	Limits the search to ports that have output delay relative to *clockName*.

Examples

The following constraints set a default output delay.

```
set_output_delay 2 [all_outputs]
```

```
set_output_delay 2 -clock {clk} [all_outputs]
```

all_registers

Use this command in the .fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects.

Returns a collection of sequential cells or pins in the current design.

Syntax

This is the supported syntax for the all_registers command:

all_registers
[**-clock** *clockName*]
[**-rise_clock** *clockName*]
[**-fall_clock** *clockName*]
[**-cells**]
[**-data_pins**]
[**-clock_pins**]
[**-output_pins**]
[**-no_hierarchy**]

Arguments

-clock *clockName*	Searches only sequential cells that are clocked by the specified clock. By default, all sequential cells in the current design are searched.
-rise_clock *clockName*	Searches only sequential cells triggered by the rising edge of the specified clock. By default, all sequential cells in the current design are searched.
-fall_clock *clockName*	Searches only sequential cells triggered by the falling edge of the specified clock. By default, all sequential cells in the current design are searched.
-cells	Returns a collection of sequential cells that meet the search criteria. If you do not specify any of the object types, the command returns a collection of sequential cells.
-data_pins	Returns a collection of data pins for the sequential cells that meet the search criteria.
-clock_pins	Returns a collection of clock pins for the sequential cells that meet the search criteria.
-output_pins	Returns a collection of output pins for the sequential cells that meet the search criteria.

-no_hierarchy Limits the search to only the current level of hierarchy. Sub-designs are not searched.

By default, the entire hierarchy is searched.

Examples

The following constraint sets a max delay target for timing paths leading to all registers.

```
set_max_delay 10.0 -to [all_registers]
```

The following constraint sets a max delay target for timing paths leading to all registers clocked by PHI2.

```
set_max_delay 10.0 -to [all_registers -clock [get_clocks PHI2]]
```

get_cells

Use this command in the .fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects.

Creates a collection of cells from the current design that is relative to the current instance.

Syntax

This is the supported syntax for the get_cells command:

get_cells
[**-hierarchical**]
[**-nocase**]
[-**regexp**]
[**-filter** *expression*]
[*pattern*]

Arguments

-hierarchical Searches each level of hierarchy for cells in the design relative to the current instance. The object name at a particular level must match the patterns. For the cell `block1/adder`, a hierarchical search uses `"adder"` to find this cell name.

By default, the search is *not* hierarchical.

-nocase	Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (=~ and !~).
-regexp	Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (=~ and !~)have also been modified to use regular expression rather than simple wildcard patterns. When using the `-regexp` option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding `".*"` to the beginning or end of the expressions, as needed.
-filter *expressions*	Filters the collection with the specified expression. For each cell in the collection, the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the cell is included in the result.
pattern	Creates a collection of cells whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option.

Examples

The following example creates a collection of cells that begin with o and reference an FD2 library cell.

```
get_cells "o*" -filter "@ref_name =~ FD2"
```

The following example creates a collection of cells connected to a collection of pins.

```
set pinsel [get_pins o*/cp]
get_cells -of_objects $pinsel
```

The following example creates a collection of cells connected to a collection of nets.

```
set netsel [get_nets tmp]
get_cells -of_objects $netsel
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

This example creates a collection of cells with the string BSCAN in its name. Make sure to use the "=~" operator when performing wildcard matching.

```
get_cells -hier * -filter {@hier_rtl_name =~ *BSCAN*}
```

get_clocks

Use this command in the .fdc constraint file and in the HDL Analyst view (on a limited basis) to return a collection of objects.

Creates a collection of clocks from the current design.

Syntax

This is the supported syntax for the get_clocks command:

get_clocks
[**-nocase**]
[**-regexp**]
[**-filter** *expression*]
[*pattern* | **-of_objects** *objects*]
[**-include_generated_clocks**]

Arguments

-nocase	Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (=~ and !~).
-regexp	Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (=~ and !~) have also been modified to use regular expression rather than simple wildcard patterns. When using the `-regexp` option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding `".*"` to the beginning or end of the expressions, as needed.

-filter *expressions*	Filters the collection with the specified expression. For each clock in the collection, the expression is evaluated based on the clock's attributes. If the expression evaluates to true, the clock is included in the result. This option is not supported in the HDL Analyst view.
pattern	Creates a collection of clocks whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option.
-of_objects *objects*	Creates a collection of clocks that are defined for the given net or pin objects.
-include_generated_clocks	Creates a collection of clocks matching the search criteria and includes any clocks derived or generated from the source clocks found. This option is not supported in the HDL Analyst view.

Examples

The following example creates a collection of clocks that match the wildcard pattern.

```
get_clocks {*BUF_1*derived_clock*}
```

The following example creates a collection of clocks that match the given regular expression.

```
get_clocks -regexp {.*derived_clock}
```

The following example creates a collection that includes clka and any generated or derived clocks of clka.

```
get_clocks -include_generated_clocks {clka}
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

get_flat_cells

Creates a collection of leaf cells that match certain criteria in the current design.

Syntax

get_flat_cells
[**-regexp | -exact**]
[**-nocase**]
[**-filter exper**]
[*patterns* **| -of_objects** *objects*]
[**-print**]

Arguments

-regexp	Views the *patterns* argument as a regular expression rather than a simple wildcard pattern. This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns. When using the **-regexp** option, be careful how you quote the patterns argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" at the beginning or end of the expressions, as needed. The **-regexp** and **-exact** options are mutually exclusive; you can use only one.
-exact	Considers wildcards to be plain characters, and does not interpret their meaning as wildcards. The **-regexp** and **-exact** options are mutually exclusive; you can use only one.
-nocase	Makes matches case-insensitive, both for the patterns argument and for the ==, =~, and !~ filter operators.

-**filter** *expression*	Filters the collection with the specified expression. For each cell in the collection, the expression is evaluated based on the cell's attributes. If the expression evaluates to true, the cell is included in the result.
patterns	Creates a collection of cells whose full names match the specified *patterns*. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the **-nocase** option. When using the *patterns* argument, the command searches all leaf cells to match the patterns argument on their full names regardless of their hierarchical level. The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.
-of_objects *objects*	Creates a collection of cells connected to the specified objects. The patterns and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.
-print	Prints the contents of the collection.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

get_flat_nets

Creates a collection of top-level nets of hierarchical net groups in the current design that match the specified criteria.

Syntax

get_flat_pins
[**-regexp | -exact**]
[**-nocase**]
[**-filter exper**]
[*patterns* **| -of_objects** *objects*]
[**-print**]

Arguments

-regexp	Views the *patterns* argument as a regular expression rather than a simple wildcard pattern. This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns. When using the **-regexp** option, be careful how you quote the patterns argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" at the beginning or end of the expressions, as needed. The **-regexp** and **-exact** options are mutually exclusive; you can use only one.
-exact	Considers wildcards to be plain characters, and does not interpret their meaning as wildcards. The **-regexp** and **-exact** options are mutually exclusive; you can use only one.
-nocase	Makes matches case-insensitive, both for the patterns argument and for the ==, =~, and !~ filter operators.

-**filter** *expression*	Filters the collection with the specified expression. For each net in the collection, the expression is evaluated based on the net's attributes. If the expression evaluates to true, the net is included in the result.
patterns	Creates a collection of top-level nets of hierarchical net groups whose full names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Patterns can include the asterisk (*) and question mark (?) wildcard characters. Pattern matching is case sensitive unless you use the **-nocase** option. When using the patterns argument, the command searches all top-level nets of hierarchical net groups to match the patterns argument on their full name. The patterns and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.
-of_objects *objects*	Creates a collection of top-level nets of hierarchical net groups that are connected to the specified objects. Each object is either a named pin, port, net, cell, or a collection of these objects. The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.
-print	Prints the contents of the collection.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

get_flat_pins

Creates a collection of leaf-cell pins that match specified criteria in the current design.

Syntax

get_flat_pins
[**-regexp | -exact**]
[**-nocase**]
[**-filter exper**]
[*patterns* **| -of_objects** *objects*]
[**-print**]

Arguments

-regexp	Views the *patterns* argument as a regular expression rather than a simple wildcard pattern. This option also modifies the behavior of the =~ and !~ filter operators to use regular expressions rather than simple wildcard patterns. When using the **-regexp** option, be careful how you quote the patterns argument and filter expression. Using rigid quoting with curly braces around regular expressions is recommended. Note that regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" at the beginning or end of the expressions, as needed. The **-regexp** and **-exact** options are mutually exclusive; you can use only one.
-exact	Considers wildcards to be plain characters, and does not interpret their meaning as wildcards. The **-regexp** and **-exact** options are mutually exclusive; you can use only one.
-nocase	Makes matches case-insensitive, both for the patterns argument and for the ==, =~, and !~ filter operators.
-filter *expression*	Filters the collection with the specified expression. For each pin in the collection, the expression is evaluated based on the pin's attributes. If the expression evaluates to true, the pin is included in the result.

patterns	Creates a collection of leaf-cell pins whose full names match the specified patterns. Patterns can include the asterisk (*) and question mark (?) wildcard characters. Pattern matching is case sensitive unless you use the **-nocase** option. When using the patterns argument, the command searches all pins of leaf cells to match the patterns argument on their full names regardless of their hierarchical level. The patterns and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.
-of_objects *objects*	Creates a collection of leaf-cell pins connected to the specified objects. Each object is a named leaf cell, a net, or a collection of these objects. The *patterns* and **-of_objects** arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses the * (asterisk) as the default pattern.
-print	Prints the contents of the collection.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

get_nets

Use this command in the .fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects.

Creates a collection of nets from the current design.

Syntax

This is the supported syntax for the get_nets command:

get_nets
[**-hierarchical**]
[**-nocase**]
[**-regexp | -exact**]
[**-filter** *expression*]
[*pattern* | **-of_objects** *objects*]

Arguments

-hierarchical	Searches each level of hierarchy for nets in the design relative to the current instance. The object name at a particular level must match the patterns. For the net `block1/muxsel` a hierarchical search uses `muxsel` to find this net name. By default, the search is *not* hierarchical.
-nocase	Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (=~ and !~).
-regexp	Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (=~ and !~) have also been modified to use regular expression rather than simple wildcard patterns. When using the `-regexp` option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding `".*"` to the beginning or end of the expressions, as needed.

-filter *expressions*	Filters the collection with the specified expression. For any nets in the collection, the expression is evaluated based on the net's attributes. If the expression evaluates to true, the net is included in the result.
pattern	Creates a collection of nets whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The patterns and -of_objects arguments are mutually exclusive; you can specify only one. If you do not specify any of these arguments, the command uses * (asterisk) as the default pattern.
-of_objects *objects*	Creates a collection of nets connected to the specified objects. Each object can be a pin, port, or cell.

Examples

The following example creates a collection of nets connected to a collection of pins.

```
set pinsel [get_pins {o_reg1.Q o_reg2.Q}]
get_nets -of_objects $pinsel
```

The following example creates a collection of nets connected to the E pin of any cell in the modulex_inst hierarchy.

```
get_nets {*.*} -filter {@pins =~ modulex_inst.*.E}
```

get_pins

Use this command in the .fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects. Creates a collection of pins from the current design that match the specified criteria.

When used without -hierarchical, include a dot (.) as a pin separator between the name of the instance and the pin name. Not including the hierarchy separator results in a warning message.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Syntax

This is the supported syntax for the get_pins command:

get_pins
[**-hierarchical**]
[**-nocase**]
[**-regexp** | **-exact**]
[**-filter** *expression*]
[*pattern* |**-of_objects** *objects* [**-leaf**]

Arguments

-hierarchical

Searches each level of hierarchy for pins, relative to the current instance, and reports all instances with that pin name. By default, searches are not hierarchical.

You can use wildcards with the -hier argument. The object name at a particular level must match the pattern. For the cell block1/adder/D[0], a hierarchical search uses adder/D[0] to find pin names.

The pin separator is not required with -hier, although it is required if you use get_pins without -hier (see *Examples of get_pins*, on page 197). However, when narrowing searches by specifying instance names as well as pin names, make sure to include the hierarchy separator. Otherwise, you will not get any search results:

```
% get_pins -hier {*reset_pipe*Q}
{}
```

```
% get_pins -hier {*reset_pipe*.Q}
{t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[0].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[1].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[2].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[3].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[4].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[5].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe[6].Q
t:sysip_inst.I_haps80_core\.I_umr_clk_gen\.reset_pipe_0[7].Q}
```

You cannot use the -hierarchical option with the -of_objects option.

© 2020 Synopsys, Inc.

-nocase
Ensures that matches are case-insensitive. This applies for both the *pattern* argument and the filter operators (== and !=).

-regexp|-exact
Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (== and !=) have also been modified to use regular expressions rather than simple wildcard patterns. When using the -regexp option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding ".*" to the beginning or end of the expressions, as needed. he -exact option treats wildcards as plain characters, so the meanings of these wildcard are not interpreted.

-filter *expressions*
Filters the collection with the specified expression. For each pin in the collection, the expression is evaluated based on the pin's attributes. If the expression evaluates to true, the pin is included in the result.

pattern | **-of_objects** *objects* [**-leaf**]
Creates a collection of pins whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The -of_objects option creates a collection of pins connected to the specified objects. Each object can be a cell or net. By default, the command considers only pins connected to the specified nets at the same hierarchical level. To consider only pins connected to leaf cells on the specified nets, use the -leaf option (the tool can cross hierarchical boundaries to find pins on leaf cells). You cannot use the -hierarchical option with the -of_objects option.

Examples of get_pins

This example creates a collection of all pins in the design.

```
get_pins -hier *.*
```

This example shows that without a separator, the command returns no results and generates a warning message:

```
% get_pins {Q}
```

```
Warning: No pin separator ('.') specified. Pattern must include a
pin separator.
```

The following example creates a collection of pins from the top-level hierarchy that match the regular expression.

```
get_pins -regexp {.*\.ena}
```

The following example creates a collection of pins throughout the hierarchy that match the regular expression.

```
get_pins -hier - regexp {.*\.ena}
```

This example illustrates that you do not need the pin separator when you specify the -hier argument:

```
% get_pins -hier {Q}

{t:haps_system_capim.capi_di[0].Q t:haps_system_capim.capi_di[1].Q
t:haps_system_capim.capi_di[2].Q t:haps_system_capim.capi_di[3].Q
t:haps_system_capim.capi_di[4].Q t:haps_system_capim.capi_di[5].Q
t:haps_system_capim.capi_di[6].Q t:haps_system_capim.capi_di[7].Q
t:haps_system_capim.capi_di[8].Q t:haps_system_capim.capi_di[9].Q
t:haps_system_capim.capi_di[10].Q
t:haps_system_capim.capi_di[11].Q
...}
```

The next example creates a collection of hierarchical pin names for the library cell pin DQSFOUND, and for each instantiation of a library cell named PHASER_IN_PHY.

```
get_pins -filter {@name == DQSFOUND} -of_objects [get_cells -hier
* -filter {@inst_of == PHASER_IN_PHY}]
```

The following example creates a collection of library cell pins with the string DRCK in its name, for each instantiation of a library cell with the string BSCAN in its name. Whenever you use wildcards to match names, make sure to specify the "=~" operator instead of the "==" operator.

```
[get_pins -filter {@name=~*DRCK} -of_objects [get_cells -hier *
-filter {@hier_rtl_name =~ *BSCAN*}]]
```

get_ports

Use this command in the .fdc constraint file to return a collection of objects and/or in the HDL Analyst view to return a Tcl list of objects.

Creates a collection of top-level ports from that match the specified criteria.

Syntax

This is the supported syntax for the get_ports command:

get_ports
[**-nocase**]
[**-regexp**]
[**-filter** *expression*]
[*pattern*]

Arguments

-nocase	Ensures that matches are case-insensitive. This applies for both the patterns argument and the filter operators (=~ and !~).
-regexp	Views the patterns argument as a regular expression rather than a simple wildcard pattern. The behavior of the filter operators (=~ and !~) have also been modified to use regular expression rather than simple wildcard patterns. When using the `-regexp` option, be careful how you quote the patterns argument and filter expression. Rigidly quoting with curly braces around regular expressions is recommended. Regular expressions are always anchored; that is, the expression assumes matching begins at the beginning of the object name and ends matching at the end of an object name. You can expand the search by adding `".*"` to the beginning or end of the expressions, as needed.
-filter *expressions*	Filters the collection with the specified expression. For each port in the collection, the expression is evaluated based on the port's attributes. If the expression evaluates to true, the port is included in the result.

pattern	Creates a collection of ports whose names match the specified patterns. Patterns can include the * (asterisk) and ? (question mark) wildcard characters. Pattern matching is case sensitive unless you use the -nocase option. The patterns and -of_objects arguments are mutually exclusive, so only specify one of them. If you do not specify either argument, the command uses * (asterisk) as the default pattern.

Examples

The following example queries all input ports beginning with mode.

```
get_ports mode* -filter {@direction =~ input}
```

object_list

Translates object strings returned by query commands in the HDL Analyst tool to proper Tcl lists. This allows you to process the results using Tcl commands.

Syntax

This is the supported syntax for the object_list command:

object_list *objectString*

Arguments

objectString	Converts the object string returned by an FDC query command to proper Tcl lists.

Examples

For example:

```
% foreach x [object_list [get_cells -hier {q[*]}]]
   {puts "Match: $x"}

   Match: i:modulex_inst.q[7:0]
   Match: i:moduley_inst\[4\].q[7:0]
```

report_timing

Alternatively, use this command to generate timing reports for a design from the Tcl window, which follows the standards set for the Design Compiler or PrimeTime® commands. One advantage this command has over the Timing Analyst GUI in the synthesis tool is that the filter options (for example, -from/-to/-through) support the FDC query commands.

Syntax

This is the supported syntax for the report_timing command:

report_timing
[**-delay_type max**]
[**-fall_from** *clock*]
[**-fall_to** *clock*]
[**-file** *string*]
[**-from** *list*]
[**-max_paths** *int*]
[**-nworst 1**]
[**-path_type full**]
[**-rise_from** *clock*]
[**-rise_to** *clock*]
[**-slack_margin** *float*]
[**-through** *instance*]
[**-to** *list*]

Arguments

-delay_type max	Specifies the path type for the end points. This default value can be specified as max; the maximum delay. All other values are ignored.
-fall_from *clock*	Reports paths from the falling edge of the specified clock. For a given clock, selects the starting points for paths clocked on the falling edge of the clock source.
-fall_to *clock*	Reports paths to the falling edge of the specified clock. For a given clock, selects the ending points for paths clocked on the falling edge of the clock source.
-file *string*	Allows the report to be re-directed to the specified file.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

-from *list*	Reports paths from the specified port, register, register pin, or clock.
-max_paths *int*	Reports the number of paths to report for the path group. The default integer value is 1.
-nworst 1	Reports the maximum number of paths to report for each timing end point. The default is 1, which reports the single worst path at a given end point. All other values are ignored.
-path_type full	Specifies how to display the timing path. This default value can be specified as full. A complete timing report is displayed, for example, containing timing start and end points, required time, and slack. All other values are ignored.
-rise_from *clock*	Reports paths from the rising edge of the specified clock. For a given clock, selects the starting points for paths clocked on the rising edge of the clock source.
-rise_to *clock*	Reports paths to the rising edge of the specified clock. For a given clock, selects the ending points for paths clocked on the rising edge of the clock source.
-slack_margin *float*	Reports paths for the specified slack margin, which allows you to specify a floating point value for the range of worst slack times.
-through *instance*	Reports paths that pass through the specified pins or nets.
-to *list*	Reports paths to the specified ports, register, register pin, or clock.

Examples

The following example reports timing to all registers clocked by clkb.

```
%report_timing -to [all_registers -clock {clkb}]

##### START OF TIMING REPORT #####[
# Timing Report written on Mon Dec 16 10:35:02 2013|
#

Top view:               top
Requested Frequency:    50.0 MHz
Wire load mode:         top
Paths requested:        1
to:                     moduley_inst.qa[7:0] moduley_inst.qb[7:0]

Worst From-To Path Information
******************************
```

```
Path information for path number 1:
   Requested Period:                      10.000
   - Setup time:                          -1.000
   + Clock delay at ending point:          0.909
   = Required time:                        11.909

   - Propagation time:                     0.000
   = Slack  :                              11.909

   Number of logic level(s):               1
   Starting point:                         c[7:0] / c[0]
   Ending point:                           moduley_inst.qa[0] / D
   The start point is clocked by           clkb [rising]
   The end   point is clocked by           clkb [rising] on pin C


 Instance/Net                Pin      Pin              Arrival  No. of
 Name                Type    Name     Dir      Delay   Time     Fan Out(s)

 c[7:0]              Port    c[0]     In       0.000   0.000    -

 c[0]                Net     -        -        0.000   -        1

 c_ibuf[0]           IBUF    I        In       -       0.000    -

 c_ibuf[0]           IBUF    O        Out      0.000   0.000    -

 c_c[0]              Net     -        -        0.000   -        1

 moduley_inst.qa[0]  FDE     D        In       -       0.000    -


Path delay compensated for clock skew. Clock skew is added to clock-to-out
value, and is subtracted from setup time value

End clock path:


 Instance/Net                Pin      Pin              Arrival  No. of
 Name                Type    Name     Dir      Delay   Time     Fan Out(s)

 clkb                Port    c[0]     In       0.000   0.000    -

 clkb                Net     -        -        0.000   -        1

 clkb_IBUFG          IBUF    I        In       -       0.000    -

 clkb_IBUFG          IBUF    O        Out      0.000   0.000    -

 clkb_i              Net     -        -        0.000   -        1

 moduley_inst.qa[0]  FDE     D        In       -       0.000    -


##### END OF TIMING REPORT #####]
```

Synopsys Standard Collection Commands

There are a number of Synopsys standard SDC collection commands that can be included in the .fdc file. These commands are not compatible with the define_scope_collection command.

The collection commands let you manipulate or operate on multiple design objects simultaneously by creating, copying, evaluating, iterating, and filtering collections. This section describes the syntax for the following collection commands supported in the FPGA synthesis tools; for the complete syntax for these commands, refer to the Design Compiler documentation.

add_to_collection	append_to_collection
copy_collection	foreach_in_collection
get_object_name	index_collection
remove_from_collection	sizeof_collection

Use these commands in the FDC constraint file to facilitate the shared scripting of constraint specifications between the FPGA synthesis and Design Compiler tools.

add_to_collection

Adds objects to a collection that results in a new collection. The base collection remains unchanged. Any duplicate objects in the resulting collection are automatically removed from the collection.

Syntax

This is the supported syntax for the add_to_collection command:

add_to_collection
[*collection1*]
[*objectSpec*]

Arguments

collection1	Specifies the base collection to which objects are to be added. This collection is copied to a resulting collection, where objects matching *objectSpec* are added to this results collection.
objectSpec	Specifies a list of named objects or collections to add. Depending on the base collection type (heterogeneous or homogeneous), the searches and resulting collection may differ. For more information, see *Heterogeneous Base Collection* , on page 205 and *Homogeneous Base Collection* , on page 205.

Description

The add_to_collection command allows you to add elements to a collection. The result is a new collection representing the objects added from the *objectSpec* list to the base collection. Objects are duplicated in the resulting collection, unless they are removed using the -unique option. If *objectSpec* is empty, then the new collection is a copy of the base collection. Depending on the base collection type (heterogeneous or homogeneous), the searches and resulting collection may differ.

Heterogeneous Base Collection

If the base collection is heterogeneous, then only collections are added to the resulting collection. All implicit elements of the *objectSpec* list are ignored.

Homogeneous Base Collection

If the base collection is homogeneous and any elements of *objectSpec* are not collections, then the command searches the design using the object class of the base collection.

When *collection1* is an empty collection, special rules apply to *objectSpec*. If *objectSpec* is not empty, at least one homogeneous collection must be in the *objectSpec* list (can be any position in the list). The first homogeneous collection in the *objectSpec* list becomes the base collection and sets the object class for the function.

Synopsys Confidential Information

Examples

```
set result [get_cells{u*}]
get_object_name $result

==> {u:u1} {i:u2} {i:u3}

set result_1 [add_to_collection $result {get_cells {i:clkb_IBUFG}]
get_object_name $result_1

==> {i:u1} {i:u2} {i:u3} {i:clkb_IBUFG}
```

See Also

- append_to_collection

append_to_collection

Adds objects to the collection specified by a variable, modifying its value. Objects must be unique, since duplicate objects are not supported.

Syntax

This is the supported syntax for the append_to_collection command:

append_to_collection]
[*variableName*]
[*objectSpec*]

Arguments

variableName	Specifies a variable name. The objects matching *objectSpec* are added to the collection referenced by this variable.
objectSpec	Specifies a list of named objects or collections to add to the resulting collection.

Description

The append_to_collection command allows you to add elements to a collection. This command treats the *variableName* option as a collection, and appends all the elements of *objectSpec* to that collection. If the variable does not exist, it creates a collection with elements from the *objectSpec* as its value. So, a

collection is created that was referenced initially by *variableName* or automatically if the *variableName* was not provided. However, if the variable exists but does not contain a collection, then an error is generated.

The append_to_collection command can be more efficient than the add_to_collection command (*add_to_collection*, on page 204) when you are building a collection in a loop.

Examples

```
set result [get_cells{u*}]
get_object_name $result

==> {u:u1} {i:u2} {i:u3}

append_to_collection result {get_cells {i:clkb_IBUFG}]
get_object_name $result

==> {i:u1} {i:u2} {i:u3} {i:clkb_IBUFG}
```

See Also

- add_to_collection

copy_collection

Duplicates the contents of a collection that results a new collection. The base collection remains unchanged.

Syntax

This is the supported syntax for the copy_collection command:

copy_collection
[*collection*1]

Arguments

collection1	Specifies the collection to be copied.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Description

The copy_collection command is an efficient mechanism to create a duplicate of an existing collection. It is sometimes more efficient and usually sufficient to simply have more than one variable referencing the same collection. However, whenever you want to copy the collection instead of reference it, use the copy_collection command.

Be aware that if an empty string is used for the *collection1* argument, the command returns an empty string. This means that a copy of the empty collection is an empty collection.

Examples

```
set insts [define_collection {u1 u2 u3 u4}]
set result_copy [copy_collection $insts]
get_object_name $result_copy

==> {u1} {u2} {u3} {u4}
```

foreach_in_collection

Iterates on the elements of a collection.

Syntax

This is the supported syntax for the foreach_in_collection command:

foreach_in_collection
[*iterationVariable*]
[*collections*]
[*body*]

Arguments

iterationVariable	Specifies the name of the iteration variable. It is set to a collection of one object. Any argument that accepts collections as an argument can also accept the *iterationVariable*, as they are the same data type.
collections	Specifies a list of collections on which to iterate.

body	Specifies a script to execute for the iteration. If the body of the iteration is modifying the netlist, all or part of the collection involved in the iteration can be deleted. The foreach_in_collection command is safe for such operations. A message is generated that indicates the iteration ended prematurely.

Description

The foreach_in_collection command is a Design Compiler and PrimeTime command used to iterate on each element of a collection. This command requires the following arguments: an iteration variable (do not specify a list), the collection on which to iterate, and the script to apply for each iteration.

You can nest this command within other control structures, including another foreach_in_collection command.

You can include the command in an FDC file, but if you are using the Tcl window and the HDL Analyst tool, you must use the standard Tcl foreach command instead of foreach_in_collection.

Examples

The following examples show valid methods to reference a collection for this command:

```
set seqs[all_registers]
set port[all_inputs]

foreach_in_collection x [all_registers] {body}
foreach_in_collection x $ports {body}
foreach_in_collection x [list $seqs $ports] {body}
foreach_in_collection x {$seqs} {body}
foreach_in_collection x {$seqs $ports} {body}
```

get_object_name

Returns a list of names for objects in a collection.

Syntax

This is the supported syntax for the get_object_name command:

get_object_name
[*collection1*]

Arguments

collection1	Specifies the name of the collection that contains the requested objects.

Examples

```
set c1[define_collection {u1 u2}]
get_object_name $c1

==> {u1} {u2}
```

index_collection

Creates a new collection that contains only the single object for the index specified in the base collection. You must provide an index to the collection.

Syntax

This is the supported syntax for the index_collection command:

index_collection
[*collection1*]
[*index*]

Arguments

collection1	Specifies the collection to be searched.

index	Specifies an index to the collection. Allowed values are integers from 0 to sizeof_collection -1.

Description

You can use the index_collection command to extract a single object from a collection. The result is a new collection that contains only this object. The range of indices can be from 0 to one less than the size of the collection. If the specified index is outside that range, an error message is generated.

Commands that create a collection of objects do not impose a specific order on the collection, but they do generate the objects in the same, predictable order each time. Applications that support the sorting of collections allow you to impose a specific order on a collection.

If you use an empty string for the *collection1* argument, then any index to the empty collection is not valid. This results in an empty collection and generates an error message.

Be aware that all collections cannot be indexed.

Examples

```
set c1[get_cells {u1 u2}]]
get_object_name [index_collection $c1 0]

==> {u1}
```

See Also

- sizeof_collection

© 2020 Synopsys, Inc.
Synopsys Confidential Information

remove_from_collection

Removes objects from a collection that results in a new collection. The base collection remains unchanged.

Syntax

This is the supported syntax for the remove_from_collection command:

remove_from_collection
[**-intersect**]
[*collection1*]
[*objectSpec*]

Arguments

-intersect	Removes objects from the base collection that are *not* found in *objectSpec*. By default, when this option is not specified, objects are removed from the base collection that are found in the *objectSpec*.
collection1	Specifies the base collection that is copied to a resulting collection, where objects matching *objectSpec* are removed from this results collection.
objectSpec	Specifies a list of named objects or collections to remove. The object class for each element in this list must be the same in the base collection. If the name matches an existing collection, that collection is used. Otherwise, objects are searched in the design using the object class for the base collection.

Description

The remove_from_collection command removes elements from a collection and creates a new collection.

When the -intersect option is not specified and there are no matches for *objectSpec*, the resulting collection is just a copy of the base collection. If everything in the *collection1* option matches the *objectSpec*, this results in an empty collection. When using the -intersect option, nothing is removed from the resulting collection.

Heterogeneous Base Collection

If the base collection is heterogeneous, then any elements of *objectSpec* that are not collections are ignored.

Homogeneous Base Collection

If the base collection is homogeneous and any elements of *objectSpec* are not collections, then the command searches the design using the object class of the base collection.

Examples

```
set c1[define_collection {u1 u2 u3}]]
set c2[define_collection {u2 u3 u4}]]
get_object_name [remove_from_collection $c1 $c2]

==> {u1}

get_object_name [remove_from_collection $c2 $c1]

==> {u4}

get_object_name [remove_from_collection -intersect $c1 $c2]

==> {u2} {u3}
```

See Also

- add_to_collection

sizeof_collection

Returns the number of objects in a collection.

Syntax

This is the supported syntax for the sizeof_collection command:

sizeof_collection
[*collection1*]

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Arguments

collection1	Specifies the name of the collection for which the number of objects is requested. If no collection argument is specified, then the command returns 0.

Examples

```
set c1[define_collection {u1 u2 u3}]
sizeof_collection $c1

==> 3
```

SYNOPSYS®
Silicon to Software™

CHAPTER 4

Constraint Commands

The SCOPE (Synthesis Constraints OPtimization Environment®) editor automatically generates syntax for synthesis constraints. Enter information in the SCOPE tabs, panels, columns, and pull-downs to define constraints and parameter values. You can also drag and drop objects from the HDL Analyst UI to populate values in the constraint fields.

This interface creates Tcl-format *Synopsys Standard timing constraints* and *Synplify-style design constraints* and saves the syntax to an FPGA design constraints (FDC) file that is automatically added to your synthesis project. See *Constraint Types*, on page 124 for definitions of synthesis constraints.

Topics in this section include:

- SCOPE Constraints Editor, on page 216
- SCOPE Tabs, on page 217
- Industry I/O Standards, on page 242
- Delay Path Timing Exceptions, on page 245
- Specifying From, To, and Through Points, on page 250
- Conflict Resolution for Timing Exceptions, on page 258

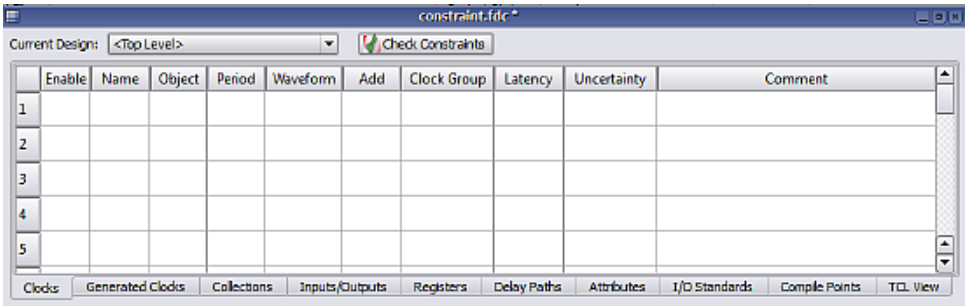
You can also specify Tcl equivalents for the timing and design constraints that are included in the SCOPE editor or a constraint file. For the constraint command syntax, see:

- Timing Constraints, on page 262.
- Design Constraints, on page 301

Synopsys Confidential Information

SCOPE Constraints Editor

The SCOPE editor contains a number of panels for creating and managing timing constraints and design attributes. This GUI offers the easiest way to create constraint files for your project. The syntax is saved to a file using an FDC extension and can be included in your design project.



From this editor, you specify timing constraints for clocks, ports, and nets as well as design constraints such as attributes, collections, and compile points. However, you cannot set black-box constraints from the SCOPE window.

To bring up the editor, use one of the following methods from the Project view:

- For a new file (the project file is open and the design is compiled):
  - Choose File->New-> FPGA Design Constraints; select FPGA Constraint File (SCOPE).
  - Click the SCOPE icon in the toolbar; select FPGA Constraint File (SCOPE).
- You can also open the editor using an existing constraint file. Double-click the constraint file (FDC), or use File->Open, specifying the file type as FPGA Design Constraints File (*.fdc).

For more information about using FPGA timing constraints with your project, see *Using the SCOPE Editor*, on page 122 in the *User Guide*.

SCOPE Tabs

Here is a summary of the constraints created through the SCOPE editor:

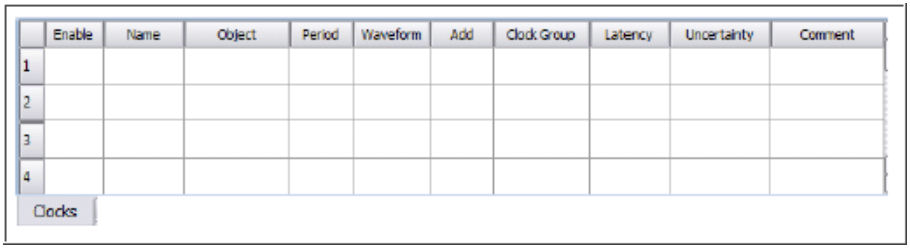
SCOPE Panel	See ...
Clocks	Clocks, on page 218
Generated Clocks	Generated Clocks, on page 223
Collections	Collections, on page 225
Inputs/Outputs	Inputs/Outputs, on page 228
Registers	Registers, on page 231
Delay Paths	Delay Paths, on page 232
Attributes	Attributes, on page 235
I/O Standards	I/O Standards, on page 236
Compile Points	Compile Points, on page 237
TCL View	TCL View, on page 240

If you choose an object from a SCOPE pull-down menu, it has the appropriate prefix appended automatically. If you drag and drop an object from an RTL view, for example, make sure to add the prefix appropriate to the language used for the module. See *Naming Rule Syntax Commands*, on page 298 for details.

Synopsys Confidential Information

Clocks

You use the Clocks panel of the SCOPE spreadsheet to define a signal as a clock.



The Clocks panel includes the following options:

Field	Description
Name	Specifies the clock object name. Clocks can be defined on the following objects: • Pins • Ports • Nets For virtual clocks, the field must contain a unique name not associated with any port, pin, or net in the design.
Period	Specifies the clock period in nanoseconds. This is the minimum time over which the clock waveform repeats. The period must be greater than zero.
Waveform	Specifies the rise and fall edge times for the clock waveforms of the clock in nanoseconds, over an entire clock period. The first time in the list is a rising transition, typically the first rising transition after time zero. There must be two edges, and they are assumed to be rise and then fall. The edges must be monotonically increasing. If you do not specify this option, a default waveform is assumed, which has a rise edge of 0.0 and a fall edge of period/2.
Add Delay	Specifies whether to add this delay to the existing clock or to overwrite it. Use this option when multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. When you use this option, you must also specify the clock, and clocks with the same source must have different names.

© 2020 Synopsys, Inc.

Field	Description
Clock Group	Assigns clocks to asynchronous clock groups. The clock grouping is inclusionary (for example, clk2 and clk3 can each be related to clk1 without being related to each other). For details, see *Clock Groups* , on page 219.
Latency	Specifies the clock latency applied to clock ports and clock aliases. Applying the latency constraint on a port can be used to model the off-chip clock delays in a multichip environment. Clock latency can only: • Apply to clocks defined on input ports. • Be used for source latency. • Apply to port clock objects.
Uncertainty	Specifies the clock uncertainty (skew characteristics) of the specified clock networks. You can only apply latency to clock objects.

Clock Groups

Clock grouping is associative; two clocks can be asynchronous to each other but both can be synchronous with a third clock.

The SCOPE GUI prompts you for a clock group for each clock that you define. By default, the tool assigns all clocks to the default clock group. When you add a name that differs from the default clock group name, the clock is assigned its own clock group and is asynchronous to the default clock group as well as all other named clock groups.

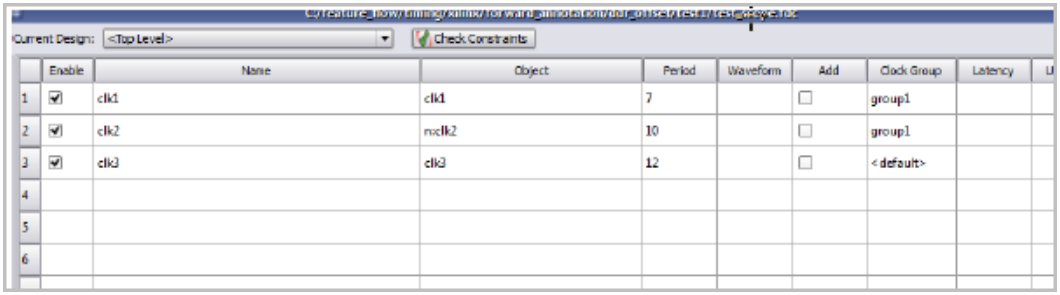
This section presents scenarios for defining clocks and includes the following examples:

- *Example 1 - SCOPE Definition*
- *Example 2 - Equivalent Tcl Syntax*
- *Example 3 - Establish Clock Relationships*
- *Example 4 - Using a Single Group Option*
- *Example 5 - Legacy Clock Grouping*

© 2020 Synopsys, Inc.

Example 1 - SCOPE Definition

A design has three clocks, clk1, clk2, clk3. You want clk1 and clk2 to be in the same clock group—synchronous to each other but asynchronous to clk3. You can apply this clock definition by adding a name in the Clock Group column, as shown below:



This definition assigns clk1 and clk2 to clock group group1, synchronous to each other and asynchronous to clk3. The equivalent Tcl command for this appears in the text editor window as follows:

```
set_clock_groups -derive -asynchronous -name {group1}
                 -group {{c:clk1} {c:clk2}}
```

Example 2 - Equivalent Tcl Syntax

A design has three clocks: clk1, clk2, clk3. Use the following commands to set clk2 synchronous to clk3, but asynchronous to clk1:

```
set_clock_groups -asynchronous -group [get_clocks {clk3 clk2}]

set_clock_groups -asynchronous -group [get_clocks {clk1}]
```

© 2020 Synopsys, Inc.

Example 3 - Establish Clock Relationships

A design has the following clocks defined:

```
create_clock -name {clka} {p:clka} -period 10 -waveform {0 5.0}
create_clock -name {clkb} {p:clkb} -period 20 -waveform {0 10.0}
create_clock -name {my_sys} {p:sys_clk} -period 200 -waveform {0
100.0}
```

You want to define clka and clkb as asynchronous to each other and clka and clkb as synchronous to my_sys.

For the tool to establish these relationships, multiple -group options are needed in a single set_clock_groups command. Clocks defined by the first –group option are asynchronous to clocks in the subsequent -group option. Therefore, you can use the following syntax to establish the relationships described above:

```
set_clock_groups -asynchronous -group [get_clocks {clka}]
            -group [get_clocks {clkb}]
```

Example 4 - Using a Single Group Option

set_clock_groups has a unique behavior when a single -group option is specified in the command. For this example, the following constraint specifications are applied:

```
set_clock_groups -asynchronous -name {default_clkgroup_0} -group
[get_clocks {clka my_sys}]

set_clock_groups -asynchronous -name {default_clkgroup_1} -group
[get_clocks {clkb my_sys}]
```

The first statement assigns clka AND my_sys as asynchronous to clkb, and the second statement assigns clkb AND my_sys as asynchronous to clka. Therefore, with this specification, all three clocks are established as asynchronous to each other.

Example 5 - Legacy Clock Grouping

This section shows how the legacy clock group definitions (Synplify-style timing constraints) are converted to the Synopsys standard timing syntax (FDC). Legacy clock grouping can be represented through Synopsys standard constraints, but the multi-grouping in the Synopsys standard constraints cannot be represented in legacy constraints.

For example, the following table shows legacy clock definitions and their translated FDC equivalents:

Legacy Definition	define_clock -name{clka}{p:clka}-period 10 -clockgroup default_clkgroup_0 define_clock -name {clkb}{p:clkb} -freq 150 -clockgroup default_clkgroup_1 define_clock -name {clkc} {p:clkc} -freq 200 -clockgroup default_clkgroup_1
FDC Definition	###==== BEGIN Clocks - (Populated from SCOPE tab, do not edit) create_clock -name {clka} {p:clka} -period 10 -waveform {0 5.0} create_clock -name {clkb} {p:clkb} -period 6.667 -waveform {0 3.3335} create_clock -name {clkc} {p:clkc} -period 5.0 -waveform {0 2.5} set_clock_groups -derive -name default_clkgroup_0 -asynchronous -group {c:clka} set_clock_groups -derive -name default_clkgroup_1 -asynchronous -group {c:clkb c:clkc} ###==== END Clocks

The create_generated_clock constraints used in legacy SDC are preserved in FDC. The -derive option directs the create_generated_clock command to inherit the -source clock group. This behavior is unique to FDC and is an extension of the Synopsys SDC standard functionality.

See Also

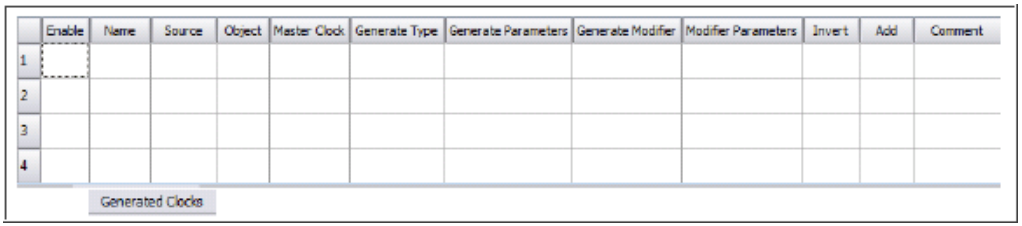
For equivalent Tcl syntax, see the following sections:

- create_clock, on page 263
- create_generated_clock, on page 265
- set_clock_latency, on page 277
- set_clock_uncertainty, on page 279

For information about other SCOPE panels, see *SCOPE Tabs*, on page 217.

Generated Clocks

Use the Generated Clocks panel of the SCOPE spreadsheet to define a signal as a generated clock. The equivalent Tcl constraint is create_generated_clock; its syntax is described in create_generated_clock, on page 265.



The Generated Clocks panel includes the following options:

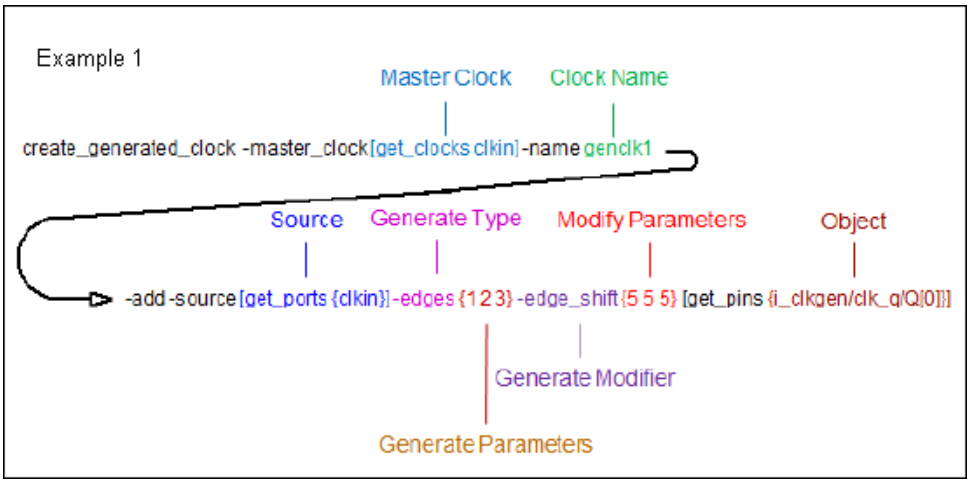
Field	Description
Name	Specifies the name of the generated clock. If this option is not used, the clock gets the name of the first clock source specified in the source.
Source	Specifies the master clock pin, which is either a master clock source pin or a fanout pin of the master clock driving the generated clock definition pin. The clock waveform at the master pin is used for deriving the generated clock waveform.
Object	Generated clocks can be defined on the following objects: • Pins • Ports • Nets • Instances
Master Clock	Specifies the master clock to be used for this generated clock, when multiple clocks fan into the master pin.

Synopsys Confidential Information

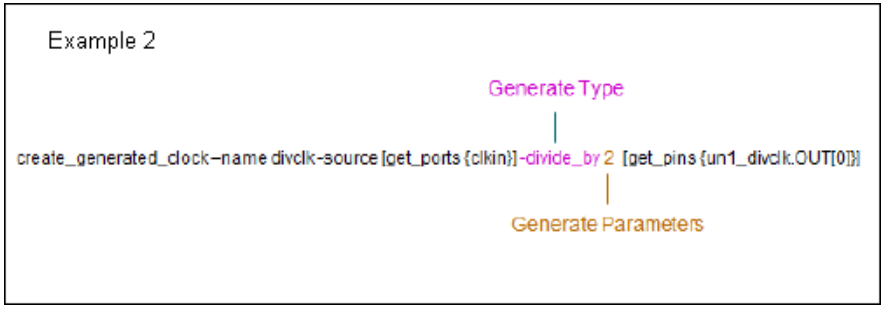
Field	Description
Generate Type	Specifies any of the following: edges - Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must not be less than its previous edge. The number of edges must be an odd number and not less than 3 to make one full clock cycle of the generated clock waveform. For example, 1 represents the first source edge, 2 represents the second source edge, and so on. divide_by - Specifies the frequency division factor. If the divide factor value is *2*, the generated clock period is twice as long as the master clock period. multiply_by - Specifies the frequency multiplication factor. If the multiply factor value is *3*, the generated clock period is one-third as long as the master clock period.
Generate Parameters	Specifies integers that define the type of generated clock.
Generate Modifier	Defines the secondary characteristics of the generated clock.
Modify Parameters	Defines modifier values of the generated clock.
Invert	Specifies whether to use invert - Inverts the generated clock signal (in the case of frequency multiplication and division).
Add	Either add this clock to the existing clock or overwrite it. Use this option when multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin. Ideally, one generated clock must be specified for each clock that fans into the master pin. If you specify this option, you must also specify the clock and master clock. The clocks with the same source must have different names.

Examples

In the following example, the generated clock genclk1 is created with the same frequency as the source clock clkin, but its phase is shifted by 180. Each of the edges of the generated clock shifts by 5 ns, which is specified by the -edges and -edge_shift options.



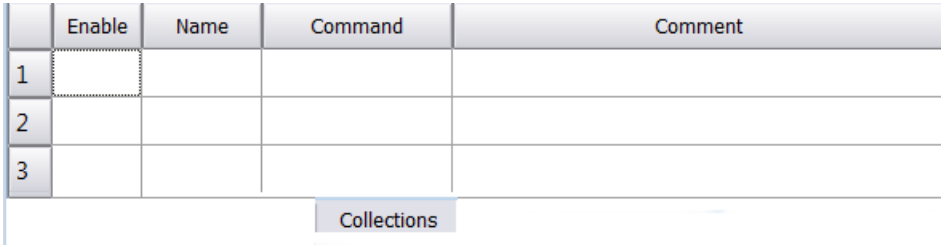
For this example, a generated clock is created with half the frequency of the source clock.



For more information about other SCOPE options, see *SCOPE Tabs*, on page 217.

Collections

The Collections tab allows you to set constraints for a group of objects you have defined as a collection with the Tcl command. For details, see *Creating and Using SCOPE Collections*, on page 155 of the *User Guide*.



Field	Description
Enable	Enables the row.
Name	Enter the collection name.
Command	Select a collection creation command from the drop-down menu. See *Collection Commands* , on page 226 for descriptions of the commands.
Comment	Enter comments that are included in the constraints file.

You can crossprobe the collection results to an HDL Analyst view. To do this, right-click in the SCOPE cell and select the option Select in Analyst.

Collection Commands

You can use the collection commands on collections or Tcl lists. Tcl lists can be just a single element long.

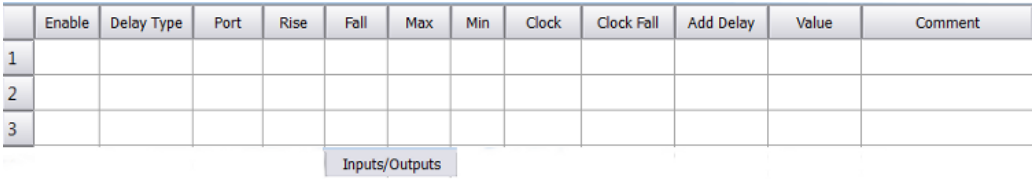
To ...	Use this command ...
Create a collection	set modules To create and save a collection, assign it to a variable. You can also use this command to create a collection from any combination of single elements, TCL lists and collections: set modules [define_collection {v:top} {v:cpu} $mycoll $mylist] Once you have created a collection, you can assign constraints to it in the SCOPE interface.
Copy a collection	set modules_copy $modules This copies the collection, so that any change to $modules does not affect $modules_copy.
Evaluate a collection	c_print This command returns all objects in a column format. Use this for visual inspection. c_list This command returns a Tcl list of objects. Use this to convert a collection to a list. You can manipulate a Tcl list with standard Tcl list commands.
Concatenate a list to a collection	c_union
Identify differences between lists or collections	c_diff Identifies differences between a list and a collection or between two or more collections. Use the -print option to display the results.
Identify objects common to a list and a collection	c_intersect Use the -print option to display the results.
Identify objects common to two or more collections	c_sub Use the -print option to display the results.
Identify objects that belong exclusively to only one list or collection	c_symdiff Use this to identify unique objects in a list and a collection, or two or more collections. Use the -print option to display the results.

For information about all SCOPE panels, see *SCOPE Tabs*, on page 217.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Inputs/Outputs

The Inputs/Outputs panel models the interface of the FPGA with the outside environment. You use it to specify delays outside the device.



The Inputs/Outputs panel includes the following options:

Field	Description
Delay Type	Specifies whether the delay is an input or output delay.
Port	Specifies the name of the port.
Rise	Specifies that the delay is relative to the rising transition on specified port. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -rise option is preserved and forward annotated to the place-and-route tool.
Fall	Specifies that the delay is relative to the falling transition on specified port Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.
Max	Specifies that the delay value is relative to the longest path. **Note:** The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.
Min	Specifies that the delay value is relative to the shortest path. **Note:** The synthesis tool does not optimize for hold time violations and only reports -min delay values in the `synlog/topLevel_fpga_mapper.srr_Min` timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.

Field	Description
Clock	Specifies the name of a clock for which the specified delay is applied. If you specify the clock fall, you must also specify the name of the clock.
Clock Fall	Specifies that the delay relative to the falling edge of the clock. For examples, see *Input Delays* , on page 229 and *Output Delays* , on page 230.
Add Delay	Specifies whether to add delay information to the existing input delay or overwrite the input delay. For examples, see *Input Delays* , on page 229 and *Output Delays* , on page 230.
Value	Specifies the delay path value.

Input Delays

Here is how this constraint applies for input delays:

- Clock Fall - The default is the rising edge or rising transition of a reference pin. If you specify clock fall, you must also specify the name of the clock.
- Add Delay - Use this option to capture information about multiple paths leading to an input port relative to different clocks or clock edges.

  For example, set_input_delay 5.0 -max -rise -clock phi1 {A} removes all maximum rise input delay from A, because the -add_delay option is not specified. Other input delays with different clocks or with -clock_fall are removed.

  In this example, the -add_delay option is specified as set_input_delay 5.0 -max -rise -clock phi1 -add_delay {A}. If there is an input maximum rise delay for A relative to clock phi1 rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is maximum rise input delay relative to a different clock or different edge of the same clock, it remains with the new delay.

Synopsys Confidential Information

Output Delays

Here is how this constraint applies for output delays:

- Clock Fall - If you specify clock fall, you must also specify the name of the clock.
- Add Delay - By using this option, you can capture information about multiple paths leading from an output port relative to different clocks or clock edges.

  For example, the set_output_delay 5.0 -max -rise -clock phi1 {OUT1} command removes all maximum rise output delays from OUT1, because the -add_delay option is not specified. Other output delays with a different clock or with the -clock_fall option are removed.

  In this example, the -add_delay option is specified: set_output_delay 5.0 -max -rise -clock phi1 -add_delay {Z}. If there is an output maximum rise delay for Z relative to the clock phi1 rising edge, the larger value is used. The smaller value does not result in critical timing for maximum delay. For minimum delay, the smaller value is used. If there is a maximum rise output delay relative to a different clock or different edge of the same clock, it remains with the new delay.

Priority of Multiple I/O Constraints

You can specify multiple input and output delays constraints for the same I/O port. This is useful for cases where a port is driven by or feeds multiple clocks. The priority of a constraint and its use in your design is determined by a few factors:

- The software applies the tightest constraint for a given clock edge, and ignores all others. All applicable constraints are reported in the timing report.
- You can apply I/O constraints on three levels, with the most specific overriding the more global:
  - Global (top-level netlist), for all inputs and outputs
  - Port-level, for the whole bus
  - Bit-level, for single bits

If there are two bit constraints and two port constraints, the two bit constraints override the two port constraints for that bit. The other bits get the two port constraints. For example, take the following constraints:

```
a[3:0]3 clk1:r
a[3:0]3 clk2:r
a[0]2 clk1:r
```

In this case, port a[0] only gets one constraint of 2 ns. Ports a[1], a[2], and a[3] get two constraints of 3 ns each.

- If at any given level (bit, port, global) there is a constraint with a reference clock specified, then any constraint without a reference clock is ignored. In this example, the 1 ns constraint on port a[0] is ignored.

```
a[0]2 clk1:r
a[0]1
```

See Also

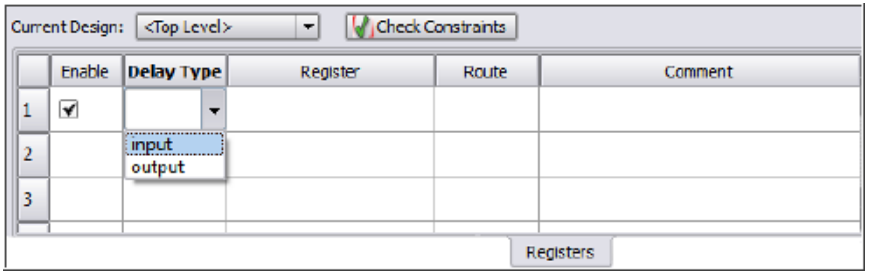
For equivalent Tcl syntax, see:

- set_input_delay, on page 284
- set_output_delay, on page 294

For information about all SCOPE panels, see *SCOPE Tabs*, on page 217.

Registers

This panel lets the advanced user add delays to paths feeding into/out of registers, in order to further constrain critical paths. You use this constraint to speed up the paths feeding a register. See set_reg_input_delay, on page 297, and set_reg_output_delay, on page 298 for the equivalent Tcl commands.

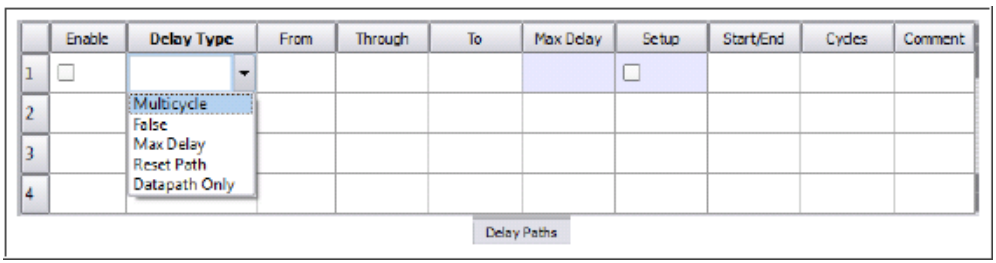


The Registers SCOPE panel includes the following fields:

Field	Description
Enabled	(Required) Turn this on to enable the constraint.
Delay Type	(Required) Specifies whether the delay is an input or output delay.
Register	(Required) Specifies the name of the register. If you have initialized a compiled design, you can choose from the pull-down list.
Route	(Required) Improves the speed of the paths to or from the register by the given number of nanoseconds. The value shrinks the effective period for the constrained registers without affecting the clock period that is forward-annotated to the place-and-route tool.
Comment	Lets you enter comments that are included in the constraints file.

Delay Paths

Use the Delay Paths panel to define the timing exceptions.



The Path Delay panel includes the following options:

Field	Description
Delay Type	Specifies the type of delay path you want the synthesis tool to analyze. Choose one of the following types: • Multicycle • False • Max Delay • Reset Path • Datapath Only
From	Starting point for the path. From points define timing start points and can be defined for clocks (c:), registers (i:), top-level input or bi-directional ports (p:), black box output pins (i:) or sequential cell clock pins. For details, see the following: • Defining From/To/Through Points for Timing Exceptions • Naming Rule Syntax Commands, on page 298
Through	Specifies the intermediate points for the timing exception. Intermediate points can be combinational nets (n:), hierarchical ports (t:), or instantiated cell pins (t:). If you click the arrow in a column cell, you open the Product of Sums (POS) interface where you can set through constraints. For details, see the following: • Product of Sums Interface • Defining From/To/Through Points for Timing Exceptions • Naming Rule Syntax Commands, on page 298
To	Ending point of the path. To points must be timing end points and can be defined for clocks (c:), registers (i:), top-level output or bi-directional ports (p:), or black box input pins (i:). For details, see the following: • Defining From/To/Through Points for Timing Exceptions • Naming Rule Syntax Commands, on page 298
Max Delay	Specifies the maximum delay value for the specified path in nanoseconds.

Synopsys Confidential Information

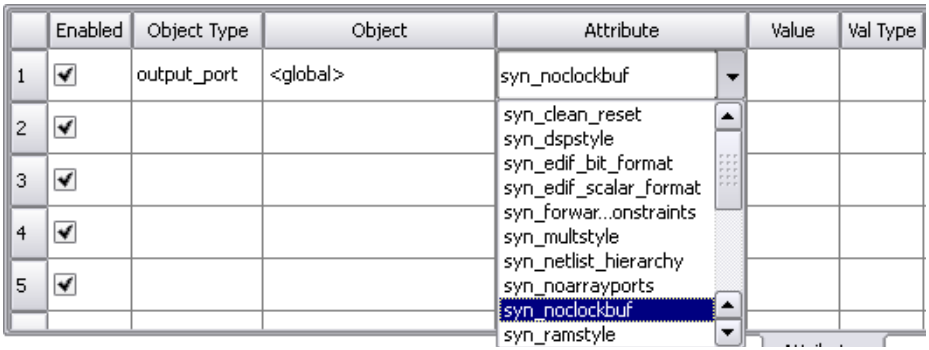
Field	Description
Setup	Specifies the setup (maximum delay) calculations used for specified path.
Start/End	Used for multicycle paths with different start and end clocks. This option determines the clock period to use for the multiplicand in the calculation for clock distance. If you do not specify a start or end clock, the end clock is the default.
Cycles	Specifies the number of cycles required for the multicycle path.

See Also

- For equivalent Tcl syntax, see:
  - set_multicycle_path, on page 290
  - set_false_path, on page 281
  - set_max_delay, on page 287
  - reset_path, on page 269
- For more information on timing exception constraints and how the tool resolves conflicts, see:
  - *Delay Path Timing Exceptions*, on page 245
  - *Conflict Resolution for Timing Exceptions*, on page 258
- For information about all SCOPE panels, see *SCOPE Tabs*, on page 217.

Attributes

You can assign attributes directly in the editor.



Here are descriptions for the Attributes columns:

Column	Description
Enabled	(Required) Turn this on to enable the constraint.
Object Type	Specifies the type of object to which the attribute is assigned. Choose from the pull-down list, to filter the available choices in the Object field.
Object	(Required) Specifies the object to which the attribute is attached. This field is synchronized with the Attribute field, so selecting an object here filters the available choices in the Attribute field.
Attribute	(Required) Specifies the attribute name. You can choose from a pull-down list that includes all available attributes for the specified technology. This field is synchronized with the Object field. If you select an object first, the attribute list is filtered. If you select an attribute first, the Synopsys FPGA synthesis tool filters the available choices in the Object field. You must select an attribute before entering a value. If a valid attribute does not appear in the pull-down list, simply type it in this field and then apply appropriate values.
Value	(Required) Specifies the attribute value. You must specify the attribute first. Clicking in the column displays the default value; a drop-down arrow lists available values where appropriate.

© 2020 Synopsys, Inc.

Val Type	Specifies the kind of value for the attribute. For example, string or boolean.
Description	Contains a one-line description of the attribute.
Comment	Lets you enter comments about the attributes.

Enter the appropriate attributes and their values, by clicking in a cell and choosing from the pull-down menu.

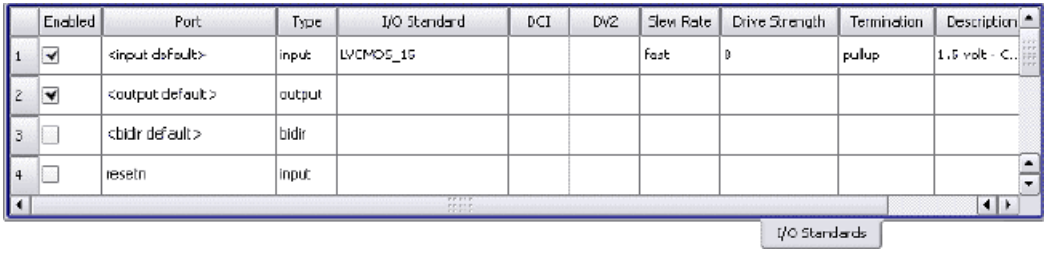
To specify an object to which you want to assign an attribute, you may also drag-and-drop it from the RTL or Technology view into a cell in the Object column. After you have entered the attributes, save the constraint file and add it to your project.

See Also

- For information about all SCOPE panels, see *SCOPE Tabs*, on page 217.

I/O Standards

You can specify a standard I/O pad type to use in the design. Define an I/O standard for any port appearing in the I/O Standards panel.



Field	Description
Enabled	(Required) Turn this on to enable the constraint, or off to disable a previous constraint.
Port	(Required) Specifies the name of the port. If you have initialized a compiled design, you can select a port name from the pull-down list. The first two entries let you specify global input and output delays, which you can then override with additional constraints on individual ports.

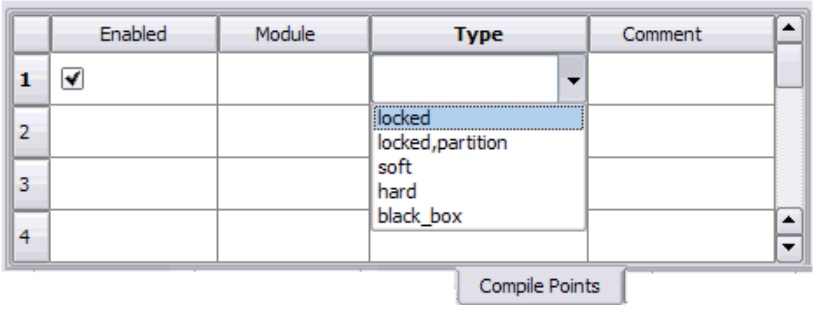
Type	(Required) Specifies whether the delay is an input or output delay.
I/O Standard	Supported I/O standards by Synopsys FPGA products. See *Industry I/O Standards* , on page 242 for a description of the standards.
Slew Rate Drive Strength Termination Power Schmitt	The values for these parameters are based on the selected I/O standard.
Description	Describes the selected I/O Standard.
Comment	Enter comments about an I/O standard.

See Also

- The Tcl equivalent of this constraint is define_io_standard.
- For information about all SCOPE panels, see *SCOPE Tabs*, on page 217.

Compile Points

Use the Compile Points panel to specify compile points in your design, and to enable/disable them. This panel, available only if the device technology supports compile points, is used to define a top-level constraint file.

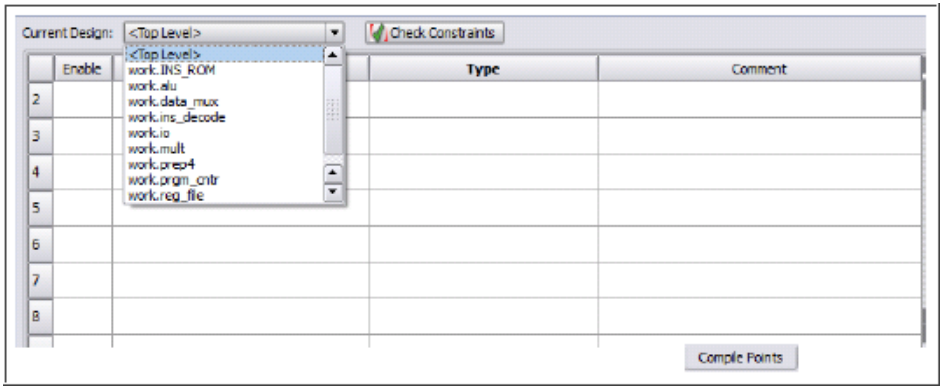


Here are the descriptions of the fields in the Compile Points panel.

Field	Description
Enabled	(Required) Turn this on to enable the constraint.
Module	(Required) Specifies the name of the compile-point module. You must specify a view module, with a v: prefix to identify the module as a view. For example: v:alu.
Type	(Required) Specifies the type of compile point: • locked (default) - no timing reoptimization is done on the compile point. The hierarchical interface is unchanged and an interface logic model is constructed for the compile point. • locked, partition - locked compile point, for which compile point information is forward annotated to the place and route tool. This mode provides place and route runtime advantages and allows for obtaining stable results for a completed design. • soft - compile point is included in the top-level synthesis, boundary optimizations can occur. • hard - compile point is included in the top-level synthesis, boundary optimizations can occur, however, the boundary remains unchanged. Although, the boundary is not modified, instances on both sides of the boundary can be modified using top-level constraints. For details, see *Compile Point Types*, on page 440 in the *User Guide*.
Comment	Lets you enter a comment about the compile point.

Constraints for Compile Points

You can set constraints at the top-level or for modules to be used as the compile points from the Current Design pull-down menu shown below. Use the Compile Points tab to select compile points and specify their types.



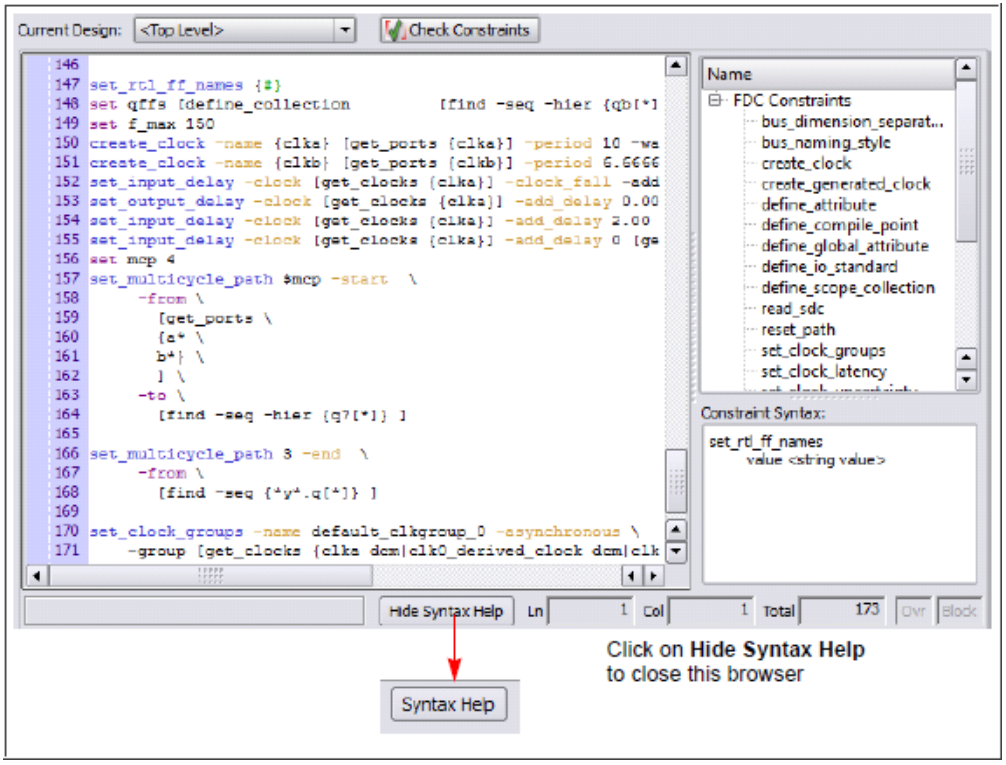
See Also

- The Tcl equivalent is define_compile_point.
- For more information on compile points and using the Compile Points panel, see *Synthesizing Compile Points*, on page 455 in the *User Guide*.
- For information about all SCOPE panels, see *SCOPE Tabs*, on page 217.

© 2020 Synopsys, Inc.

TCL View

The TCL View is an advanced text file editor for defining FPGA timing and design constraints.



This text editor provides the following capabilities:

- Uses dynamic keyword expansion and tool tips for commands that
  - Automatically completes the command from a popup list
  - Displays complete command syntax as a tool tip
  - Displays parameter options for the command from a popup list
  - Includes a keyword command syntax help
- Checks command syntax and uses color indicators that
  - Validate commands and command syntax
  - Identifies FPGA design constraints and SCOPE legacy constraints

- Allows for standard editor commands, such as copy, paste, comment/un-comment a group of lines, and highlighting of keywords

For information on how to use this Tcl text editor, see *Using the TCL View of SCOPE GUI*, on page 134.

See Also

- For Tcl timing constraint syntax, see *Timing Constraints*, on page 262.
- For Tcl design constraint syntax, see *Design Constraints*, on page 301.

Synopsys Confidential Information

Industry I/O Standards

The synthesis tool lets you specify a standard I/O pad type to use in your design. You can define an I/O standard for any port supported from the industry standard and proprietary I/O standards.

For industry I/O standards, see *Industry I/O Standards*, on page 242.

Industry I/O Standards

The following table lists industry I/O standards.

I/O Standard	Description
BLVDS_25	Bus Differential Transceiver
CTT	Center Tap Terminated - EIA/JEDEC Standard JESD8-4
DIFF_HSTL_15_Class_I	1.5 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
DIFF_HSTL_15_Class_II	1.5 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
DIFF_HSTL_18_Class_I	1.8 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-9A
DIFF_HSTL_18_Class_II	1.8 volt - Differential High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-9A
DIFF_SSTL_18_Class_II	1.8 volt - Differential Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-6
DIFF_SSTL_2_Class_I	2.5 volt - Pseudo Differential Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9A
DIFF_SSTL_2_Class_II	2.5 volt - Pseudo Differential Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9A
GTL	Gunning Transceiver Logic - EIA/JEDEC Standard JESD8-3
GTL+	Gunning Transceiver Logic Plus
GTL25	Gunning Transceiver Logic - EIA/JEDEC Standard JESD8-3
GTL+25	Gunning Transceiver Logic Plus
GTL33	Gunning Transceiver Logic - EIA/JEDEC Standard JESD8-3
GTL+33	Gunning Transceiver Logic Plus

© 2020 Synopsys, Inc.

I/O Standard	Description
HSTL_12	1.2 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_15_Class_II	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_I	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_II	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_III	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_18_Class_IV	1.8 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_I	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_II	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_III	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HSTL_Class_IV	1.5 volt - High Speed Transceiver Logic - EIA/JEDEC Standard JESD8-6
HyperTransport	2.5 volt - Hypertransport - HyperTransport Consortium

© 2020 Synopsys, Inc.
Synopsys Confidential Information

I/O Standard	Description
LVCMOS_12	1.2 volt - EIA/JEDEC Standard JESD8-16
LVCMOS_15	1.5 volt - EIA/JEDEC Standard JESD8-7
LVCMOS_18	1.8 volt - EIA/JEDEC Standard JESD8-7
LVCMOS_25	2.5 volt - EIA/JEDEC Standard JESD8-5
LVCMOS_33	3.3 volt CMOS - EIA/JEDEC Standard JESD8-B
LVCMOS_5	5.0 volt CMOS
LVDS	Differential Transceiver - ANSI/TIA/EIA-644-95
LVDSEXT_25	Differential Transceiver
LVPECL	Differential Transceiver - EIA/JEDEC Standard JESD8-2
LVTTL	3.3 volt TTL - EIA/JEDEC Standard JESD8-B
MINI_LVDS	Mini Differential Transceiver
PCI33	3.3 volt PCI 33MHz - PCI Local Bus Spec. Rev. 3.0 (PCI Special Interest Group)
PCI66	3.3 volt PCI 66MHz - PCI Local Bus Spec. Rev. 3.0 (PCI Special Interest Group)
PCI-X_133	3.3 volt PCI-X - PCI Local Bus Spec. Rev. 3.0 (PCI Special Interest Group)
PCML	3.3 volt - PCML
PCML_12	1.2 volt - PCML
PCML_14	1.4 volt - PCML
PCML_15	1.5 volt - PCML
PCML_25	2.5 volt - PCML
RSDS	Reduced Swing Differential Signalling
SSTL_18_Class_I	1.8 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-15
SSTL_18_Class_II	1.8 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-15
SSTL_2_Class_I	2.5 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9B
SSTL_2_Class_II	2.5 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-9B
SSTL_3_Class_I	3.3 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-8
SSTL_3_Class_II	3.3 volt - Stub Series Terminated Logic - EIA/JEDEC Standard JESD8-8
ULVDS_25	Differential Transceiver

© 2020 Synopsys, Inc.

Delay Path Timing Exceptions

For details about the following path types, see:

- Multicycle Paths, on page 245
- False Paths, on page 248

Multicycle Paths

Multicycle paths lets you specify paths with multiple clock cycles. The following table defines the parameters for this constraint. For the equivalent Tcl constraints, see set_multicycle_path, on page 290. This section describes the following:

- Multi-cycle Path with Different Start and End Clocks, on page 245
- Multicycle Path Examples, on page 246

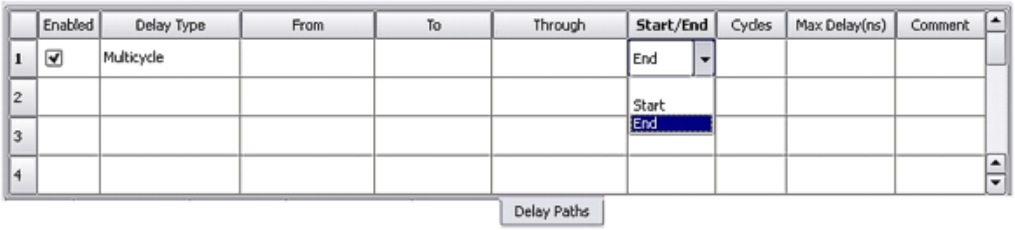
Multi-cycle Path with Different Start and End Clocks

The start/end option determines the clock period to use for the multiplicand in the calculation for required time. The following table describes the behavior of the multi-cycle path constraint using different start and end clocks. In all equations, *n* is number of clock cycles, and *clock_distance* is the default, single-cycle relationship between clocks that is calculated by the tool.

Basic required time for a multi-cycle path	clock_distance + [(n-1) * end_clock_period]
Required time with no end clock defined	clock_distance + [(n-1) * global_period]
Required time with -start option defined	clock_distance + [(n-1) * start_clock_period]
Required time with no start clock defined	clock_distance + [(n-1) * global_period]

© 2020 Synopsys, Inc.
Synopsys Confidential Information

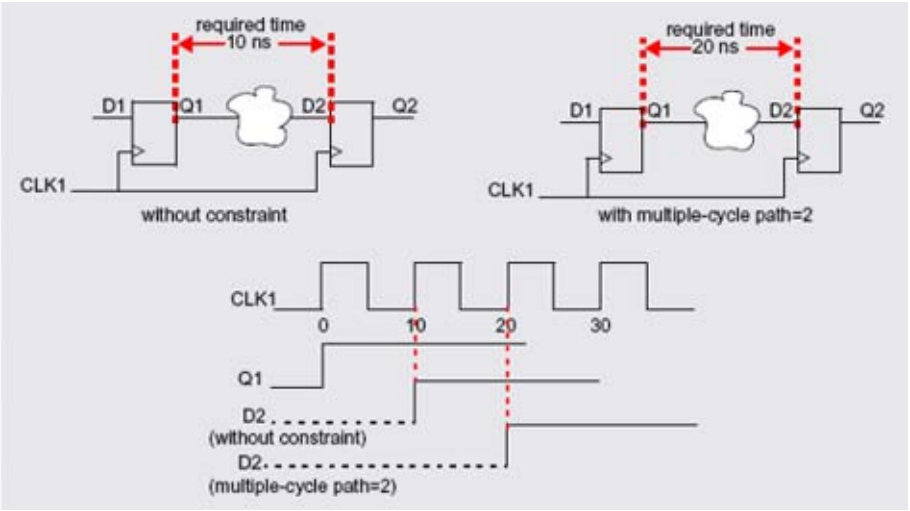
If you do not specify a start or end option, by default the end clock is used for the constraint. Here is an example:



Multicycle Path Examples

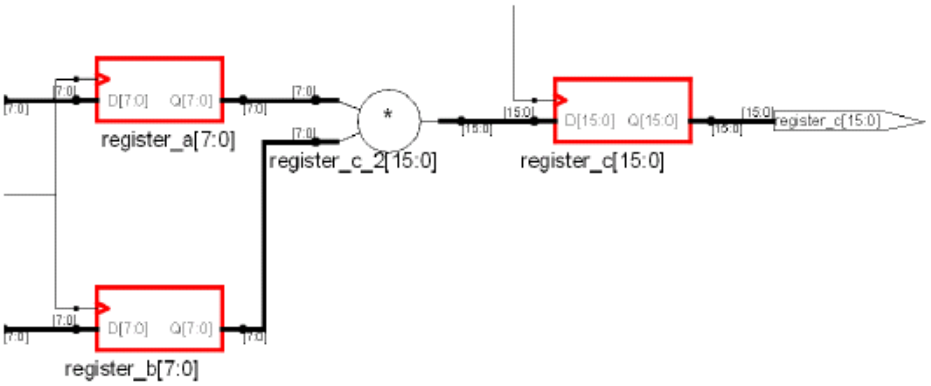
Multicycle Path Example 1

If you apply a multicycle path constraint from D1 to D2, the allowed time is #cycles x normal time between D1 and D2. In the following figure, CLK1 has a period of 10 ns. The data in this path has only one clock cycle before it must reach D2. To allow more time for the signal to complete this path, add a multiple-cycle constraint that specifies two clock cycles (10 x 2 or 20 ns) for the data to reach D2.



Multicycle Path Example 2

The design has a multiplier that multiplies signal_a with signal_b and puts the result into signal_c. Assume that signal_a and signal_b are outputs of registers register_a and register_b, respectively. The RTL view for this example is shown below. On clock cycle 1, a state machine enables an input enable signal to load signal_a into register_a and signal_b into register_b. At the beginning of clock cycle 2, the multiply begins. After two clock cycles, the state machine enables an output_enable signal on clock cycle 3 to load the result of the multiplication (signal_c) into an output register (register_c).



The design frequency goal is 50 MHz (20 ns) and the multiply function takes 35 ns, but it is given 2 clock cycles. After optimization, this 35 ns path is normally reported as a timing violation because it is more than the 20 ns clock-cycle timing goal. To avoid reporting the paths as timing violations, use the SCOPE window to set 2-cycle constraints (From column) on register_a and register_b, or include the following in the timing constraint file:

```
# Paths from register_a use 2 clock cycles
set_multicycle_path -from register_a 2

# Paths from register_b use 2 clock cycles
set_multicycle_path -from register_b 2
```

Alternatively, you can specify a 2-cycle SCOPE constraint (To column) on register_c, or add the following to the constraint file:

```
# Paths to register_c use 2 clock cycles
set_multicycle_path -to register_c 2
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

False Paths

You use the Delay Paths constraint to specify clock paths that you want the synthesis tool to ignore during timing analysis and assign low (or no) priority during optimization. The equivalent Tcl constraint is described in set_false_path, on page 281.

This section describes the following:

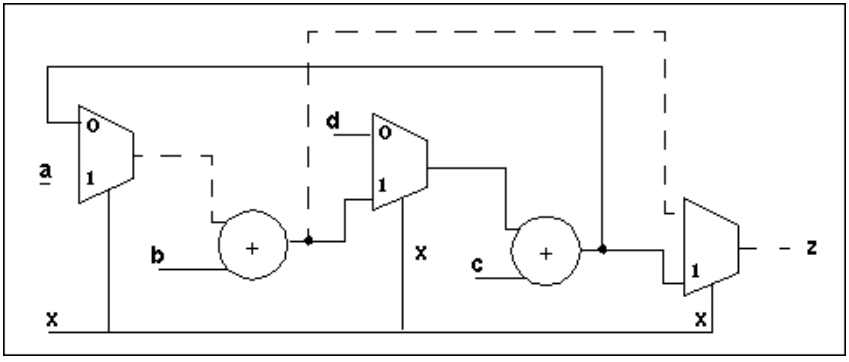
- Types of False Paths, on page 248
- False Path Constraint Examples, on page 249

Types of False Paths

A false path is a path that is not important for timing analysis. There are two types of false paths:

- Architectural false paths

  These are false paths that the designer is aware of, like an external reset signal that feeds internal registers but which is synchronized with the clock. The following example shows an architectural false path where the primary input x is always 1, but which is not optimized because the software does not optimize away primary inputs.



- Code-introduced false paths

  These are false paths that you identify after analyzing the schematic.

False Path Constraint Examples

In this example, the design frequency goal is 50 MHz (20ns) and the path from register_a to register_c is a false path with a large delay of 35 ns. After optimization, this 35 ns path is normally reported as a timing violation because it is more than the 20 ns clock-cycle timing goal. To lower the priority of this path during optimization, define it as a false path. You can do this in many ways:

- If all paths from register_a to any register or output pins are not timing-critical, then add a false path constraint to register_a in the SCOPE interface (From), or put the following line in the timing constraint file:

  ```
  #Paths from register_a are ignored
  set_false_path -from {i:register_a}
  ```

- If all paths to register_c are not timing-critical, then add a false path constraint to register_c in the SCOPE interface (To), or include the following line in the timing constraint file:

  ```
  #Paths to register_c are ignored
  set_false_path -to {i:register_c}
  ```

- If only the paths between register_a and register_c are not timing-critical, add a From/To constraint to the registers in the SCOPE interface (From and To), or include the following line in the timing constraint file:

  ```
  #Paths to register_c are ignored
  set_false_path -from {i:register_a} -to {i:register_c}
  ```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

Specifying From, To, and Through Points

The following section describes from, to, and through points for timing exceptions specified by the multicycle paths, false paths, and max delay paths constraints.

- Timing Exceptions Object Types, on page 250
- From/To Points, on page 250
- Through Points, on page 252
- Product of Sums Interface, on page 253
- Clocks as From/To Points, on page 256

Timing Exceptions Object Types

Note the following timing exceptions for object types:

- Timing exceptions must contain the type of object in the constraint specification. You must explicitly specify an object type, n: for a net, or i: for an instance, in the instance name parameter of all timing exceptions. For example:

  ```
  set_multicycle_path -from {i:inst2.lowreg_output[7]}
     -to {i:inst1.DATA0[7]} 2
  ```

  If you use the SCOPE GUI to specify timing exceptions, it automatically attaches the object type qualifier to the object name.

- When defining constraints for the fdc file from the Tcl View window in SCOPE, it is recommended you use get_* as the object type qualifier.

  For example, use get_ports, get_nets, get_pins, or get_cells instead of p:, n:, t:, or i:.

From/To Points

From specifies the starting point for the timing exception. To specifies the ending point for the timing exception. When you specify an object, use the appropriate prefix (see *syn_black_box*, on page 63) to avoid confusion. The following table lists the objects that can serve as starting and ending points:

From Points	To Points
Clocks. See *Clocks as From/To Points* , on page 256 for more information.	Clocks. See *Clocks as From/To Points* , on page 256 for more information.
Registers	Registers
Top-level input or bi-directional ports	Top-level output or bi-directional ports
Instantiated library primitive cells (gate cells)	Instantiated library primitive cells (gate cells)
Black box outputs	Black box inputs

You can specify multiple from points in a single exception. This is most common when specifying exceptions that apply to all the bits of a bus. For example, you can specify constraints From A[0:15] to B - in this case, there is an exception, starting at any of the bits of A and ending on B.

Similarly, you can specify multiple to points in a single exception. If you specify both multiple starting points and multiple ending points such as From A[0:15] to B[0:15], there is actually an exception from any start point to any end point. In this case, the exception applies to all 16 * 16 = 256 combinations of start/end points.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Through Points

Through points are limited to nets, hierarchical ports, and pins of instantiated cells. There are many ways to specify these constraints.

- Single Point
- Single List of Points
- Multiple Through Points
- Multiple Through Lists

You define these constraints in the appropriate SCOPE panels, or in the POS GUI (see *Product of Sums Interface*, on page 253). When a port and net have the same name, preface the name of the through point with n: for nets or t: for hierarchical ports. For example, you can specify n:regs_mem[2] or t:dmux.bdpol. The n: prefix must be specified to identify nets; otherwise, the associated timing constraint will not be applied for valid nets.

Single Point

You can specify a single through point. In this case, the constraint is applied to any path that passes through net regs_mem[2] as follows:

```
set_false_path -through n:regs_mem[2]
```

```
set_false_path -through [get_nets {regs_mem[2]}]
```

Single List of Points

If you specify a list of through points, the through option behaves as an OR function and applies to any path that passes through any of the points in the list. In the following example, the constraint is applied to any path through regs_mem[2] OR prgcntr.pc[7] OR dmux.alub[0] with a maximum delay value of 5 ns (-max 5):

```
set_max_delay
-through {t:regs_mem[2] t:prgcntr.pc[7] t:dmux.alub[0]} 5
```

Multiple Through Points

You can specify multiple points for the same constraint by preceding each point with the -through option. In the following example, the constraint operates as an AND function and applies to paths through regs_mem[2] AND prgcntr.pc[7] AND dmux.alub[0]:

```
set_max_delay
-through t:regs_mem[2]
-through t:prgcntr.pc[7]
-through t:dmux.alub[0] 5
```

Multiple Through Lists

If you specify multiple -through lists, the constraint is applied as an AND/OR function and is applied to the paths through all points in the lists. The following constraint applies to all paths that pass through nets {A_1 or A_2 or...A_n} AND nets {B_1 or B_2 or B_3}:

```
set_false_path -through {n:A1 n:A2...n:An} -through {n:B1 n:B2 n:B3}
```

In this example,

```
set_multicycle_path
-through {n:net1 n:net2}
-through {n:net3 n:net4} 2
```

all paths that pass through the following nets are constrained at 2 clock cycles:

```
net1 AND net3
OR net1 AND net4
OR net2 AND net3
OR net2 AND net4
```

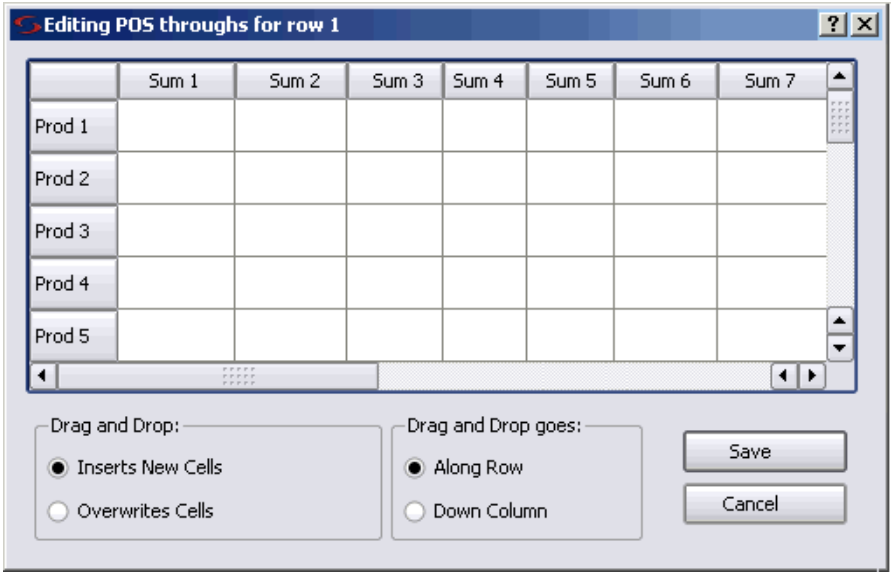
Product of Sums Interface

You can use the SCOPE GUI to format -through points for nets with multicycle path, false path, and max delay path constraints in the Product of Sums (POS) interface of the SCOPE editor. You can also manually specify constraints that use the -through option. For more information, see *Defining From/To/Through Points for Timing Exceptions*, on page 139 in the *User Guide*.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

The POS interface is accessible by clicking the arrow in a Through column cell in the following SCOPE panels:

- Multi-Cycle Paths
- False Paths
- Delay Paths



Field	Description
Prod 1, 2, etc.	Type the first net name in a cell in a Prod row, or drag the net from a HDL Analyst view into the cell. Repeat this step along the same row, adding other nets in the Sum columns. The nets in each row form an OR list.
Sum 1, 2, etc.	Type the first net name in the first cell in a Sum column, or drag the net from a HDL Analyst view into the cell. Repeat this step down the same Sum column. The nets in each column form an AND list.

Drag and Drop Goes	Along Row - places objects in multiple Sum columns, utilizing only one Prod row. Down Column - places objects in multiple Prod rows, utilizing only one Sum column.
Drag and Drop	Inserts New Cells - New cells are created when dragging and dropping nets. Overwrites Cells - Existing cells are overwritten when dragging and dropping nets.
Save/Cancel	Saves or cancels your session.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Clocks as From/To Points

You can specify clocks as from/to points in your timing exception constraints. Here is the syntax:

set_*timing_exception* **-from** | **-to {c:***clock_name* [**:***edge*]**}**

where

- *timing_exception* is one of the following constraint types: multicycle path, false path, or max delay
- **c:***clock_name***:***edge* is the name of the clock and clock edge (r or f). If you do not specify a clock edge, by default both edges are used.

See the following sections for details and examples on each timing exception.

Multicycle Path Clock Points

When you specify a clock as a from or to point, the multicycle path constraint applies to all registers clocked by the specified clock.

The following constraint allows two clock periods for all paths from the rising edge of the flip-flops clocked by clk1:

```
set_multicycle_path -from {c:clk1:r} 2
```

You cannot specify a clock as a through point. However, you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The following constraint allows two clock periods for all paths to the falling edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net:

```
set_multicycle_path -to {c:clk1:f} -through (n:MYINST.mybus2[9]} 2
```

False Path Clock Points

When you specify a clock as a from or to point, the false path constraint is set on all registers clocked by the specified clock. False paths are ignored by the timing analyzer. The following constraint disables all paths from the rising edge of the flip-flops clocked by clk1:

```
set_false_path -from {c:clk1:r}
```

You cannot specify a clock as a through point. However, you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The following constraint disables all paths to the falling edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net.

```
set_false_path -to {c:clk1:f} -through (n:MYINST.mybus2[9]}
```

Path Delay Clock Points

When you specify a clock as a from or to point for the path delay constraint, the constraint is set on all paths of the registers clocked by the specified clock. This constraint sets a max delay of 2 ns on all paths to the falling edge of the flip-flops clocked by clk1:

```
set_max_delay -to {c:clk1:f} 2
```

You cannot specify a clock as a through point, but you can set a constraint from or to a clock and through an object (net, pin, or hierarchical port). The next constraint sets a max delay of 0.2 ns on all paths from the rising edge of the flip-flops clocked by clk1 and through bit 9 of the hierarchical net:

```
set_max_delay -from {c:clk1:r} -through (n:MYINST.mybus2[9]}.2
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

Conflict Resolution for Timing Exceptions

The term *timing exceptions* refers to the false path, max path delay, and multicycle path timing constraints. When the tool encounters conflicts in the way timing exceptions are specified through the constraint file, the software uses a set priority to resolve these conflicts. Conflict resolution is categorized into four levels, meaning that there are four different tiers at which conflicting constraints can occur, with one being the highest. The table below summarizes conflict resolution for constraints. The sections following the table provide more details on how conflicts can occur and examples of how they are resolved.

Conflict Level	Constraint Conflict	Priority	For Details, see ...
1	Different timing exceptions set on the same object.	1 - False Path 2 - Path Delay 3 - Multi-cycle Path	*Conflicting Timing Exceptions* , on page 259.
2	Timing exceptions of the same constraint type, using different semantics (from/to/through).	1 - From 2 - To 3 - Through	*Same Constraint Type with Different Semantics* , on page 260.
3	Timing exceptions of the same constraint type using the same semantic, but set on different objects.	1 - Ports/Instances/Pins 2 - Clocks	*Same Constraint and Semantics with Different Objects* , on page 261.
4	Identical timing constraints, except constraint values differ.	Tightest, or most constricting constraint.	*Identical Constraints with Different Values* , on page 261.

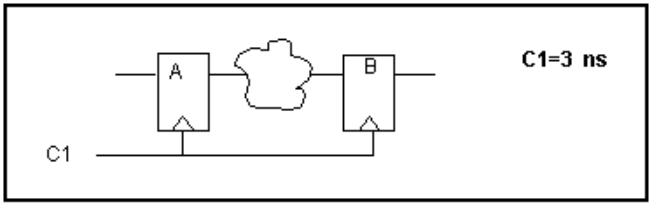
In addition to the four levels of conflict resolution for timing exceptions, there are priorities for the way the tool handles multiple I/O delays set on the same port and implicit and explicit false path constraints. For information on resolving these types of conflicts, see *Priority of Multiple I/O Constraints*, on page 230.

Conflicting Timing Exceptions

The first (and highest) level of resolution occurs when timing exceptions—false paths, max path delay, or multicycle path constraints—conflict with each other. The tool follows this priority for applying timing exceptions:

1. False Path
2. Path Delay
3. Multicycle Path

For example:




```
set_false_path -from {c:C1:r}
set_max_delay -from {i:A} -to {i:B} 10
set_multicycle_path -from {i:A} -to {i:B} 2
```

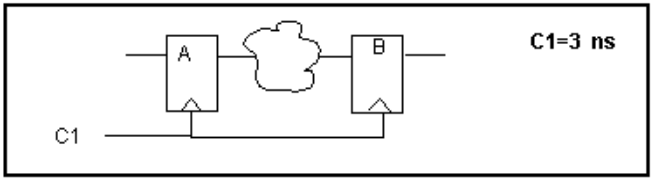
These constraints are conflicting because the path from A to B has three different constraints set on it. When the tool encounters this type of conflict, the false path constraint is honored. Because it has the highest priority of all timing exceptions, set_false_path is applied and the other timing exceptions are ignored.

Same Constraint Type with Different Semantics

The second level of resolution occurs when conflicts between timing exceptions that are of the same constraint type, use different semantics (from/to/through). The priority for these constraints is as follows:

1. From
2. To
3. Through

If there are two multicycle constraints set on the same path, one specifying a from point and the other specifying a to point, the constraint using -from takes precedence, as in the following example.




```
set_multicycle_path -from {i:A} 3
set_multicycle_path -to {i:B} 2
```

In this case, the tool uses:

```
set_multicycle_path -from {i:A} 3
```

The other constraint is ignored even though it sets a tighter constraint.

Same Constraint and Semantics with Different Objects

The third level resolves timing exceptions of the same constraint type that use the same semantic, but are set on different objects. The priority for design objects is as follows:

1. Ports/Instances/Pins
2. Clocks

If the same constraints are set on different objects, the tool ignores the constraint set on the clock for that path.

```
set_multicycle_path -from {i:mac1.datax[0]} -start 4
set_multicycle_path -from {c:clk1:r} 2
```

In the example above, the tool uses the first constraint set on the instance and ignores the constraint set on the clock from i:mac1.datax[0], even though the clock constraint is tighter.

For details on how the tool prioritizes multiple I/O delays set on the same port or implicit and explicit false path constraints, see *Priority of Multiple I/O Constraints*, on page 230.

Identical Constraints with Different Values

Where timing constraints are identical except for the constraint value, the tightest or most constricting constraint takes precedence. In the following example, the tool uses the constraint specifying two clock cycles:

```
set_multicycle_path -from {i:special_regs.trisa[7:0]} 2
set_multicycle_path -from {i:special_regs.trisa[7:0]} 3
```


© 2020 Synopsys, Inc.

Timing Constraints

The FPGA synthesis tools support FPGA timing constraints for a subset of the clock definition, I/O delay, and timing exception constraints.

The remainder of this section describes the constraint file syntax for the following FPGA timing constraints in the FPGA synthesis tools.

create_clock
create_generated_clock
reset_path
set_clock_groups
set_clock_latency
set_clock_uncertainty
set_false_path
set_input_delay
set_max_delay
set_multicycle_path
set_output_delay
set_reg_input_delay
set_reg_output_delay

Note: When adding comments for constraints, use standard Tcl syntax conventions. Otherwise, invalid specifications can cause the constraint to be ignored. The (#) comment must begin on a new line or needs to be preceded by a (;), if the comment is on the same line as the constraint. For example:

```
create_clock -period 10 [get_ports CLK]; # comment text

# comment text
set_clock_groups -asynchronous -group
MMCM_module|clk100_90_MMCM_derived_clock_CLKIN1
```

create_clock

Creates a clock object and defines its waveform in the current design.

Syntax

The supported syntax for the create_clock constraint is:

create_clock
-name *clockName* [**-add**] **{***objectList***}** |
-name *clockName* [**-add**] [**{***objectList***}**] |
[**-name** *clockName* [**-add**]] **{***objectList***}**
-period *value*
[**-waveform {***riseValue fallValue***}**]
[**-disable**]
[**-comment** *commentString*]

Arguments

-name *clockName*	Specifies the name for the clock being created, enclosed in quotation marks or curly braces. If this option is not used, the clock gets the name of the first clock source specified in the *objectList* option. If you do not specify the *objectList* option, you must use the -name option, which creates a virtual clock not associated with a port, pin, or net. You can use both the -name and *objectList* options to give the clock a more descriptive name than the first source pin, port, or net. If you specify the -add option, you must use the -name option and the clocks with the same source must have different names.
-add	Specifies whether to add this clock to the existing clock or to overwrite it. Use this option when multiple clocks must be specified on the same source for simultaneous analysis with different clock waveforms. When you specify this option, you must also use the -name option.
-period *value*	Specifies the clock period in nanoseconds. This is the minimum time over which the clock waveform repeats. The *value* type must be greater than zero.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

-waveform *riseValue fallValue*	Specifies the rise and fall edge times for the clock waveforms of the clock in nanoseconds, over an entire clock period. The first time is a rising transition, typically the first rising transition after time zero. There must be two edges, and they are assumed to be rise followed by fall. The edges must be monotonically increasing. If you do not specify this option, a default waveform is assumed, which has a rise edge of 0.0 and a fall edge of *periodValue*/2.
objectList	Clocks can be defined on the following objects: pins, ports, and nets The FPGA synthesis tools support nets and instances, where instances have only one output (for example, BUFGs).
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Examples

Refer to the following examples.

Example 1

A clock named clk_in1 is created for port clk_in1 that uses a period of 10 with rising edge of 0 and falling edge of 5.

```
create_clock -name {clk_in1} -period 10 [get_ports {clk_in1}]
```

Example 2

A clock named clk is created for port clk_in that uses a period of 10.0 with rising edge of 5.0 and falling edge of 9.5.

```
create_clock -name {clk} -period 10 -waveform {5.0 9.5}
   [get_ports {clk_in}]
```

Example 3

A virtual clock named CLK is created that uses a period of 12 with a rising edge of 0.0 and falling edge of 6.0.

```
create_clock -name {CLK} -period 12
```

create_generated_clock

Creates a generated clock object.

Syntax

The supported syntax for the create_generated_clock constraint is:

create_generated_clock
-name *clockName* [**-add**]] | **{***clockObject***}**
-source *masterPinName*
[**-master_clock** *clockName*]
[**-divide_by** *integer* | **-multiply_by** *integer* [**-duty_cycle** *value*]]
[**-invert**]
[**-edges {***edgeList***}**]
[**-edge_shift {***edgeShiftList***}**]
[**-combinational**]
[**-disable**]
[**-comment** *commentString*]

Arguments

-name *clockName*	Specifies the name of the generated clock. If this option is not used, the clock gets the name of the first clock source specified in the -source option (*clockObject*). If you specify the -add option, you must use the -name option and the clocks with the same source must have different names.
-add	Specifies whether to add this clock to the existing clock or to overwrite it. Use this option when multiple generated clocks must be specified on the same source, because multiple clocks fan into the master pin. Ideally, one generated clock must be specified for each clock that fans into the master pin. If you specify this option, you must also use the -name and -master_clock options.
clockObject	The first clock source specified in the -source option in the absence of *clockName*. Clocks can be defined on pins, ports, and nets. The FPGA synthesis tools support nets and instances, where instances have only one output (for example, BUFGs).

© 2020 Synopsys, Inc.
Synopsys Confidential Information

-source *masterPinName*	Specifies the master clock pin, which is either a master clock source pin or a fanout pin of the master clock driving the generated clock definition pin. The clock waveform at the master pin is used for deriving the generated clock waveform.
-master_clock *clockName*	Specifies the master clock to be used for this generated clock, when multiple clocks fan into the master pin.
-divide_by *integer*	Specifies the frequency division factor. If the *divideFactor* value is *2*, the generated clock period is twice as long as the master clock period.
-multiply_by *integer*	Specifies the frequency multiplication factor. If the *multiplyFactor* value is *3*, the generated clock period is one-third as long as the master clock period.
-duty_cycle *percent*	Specifies the duty cycle, as a percentage, if frequency multiplication is used. Duty cycle is the high pulse width. **Note:** This option is valid only when used with the -multiply_by option.
-invert	Inverts the generated clock signal (in the case of frequency multiplication and division).
-edges *edgeList*	Specifies a list of integers that represents edges from the source clock that are to form the edges of the generated clock. The edges are interpreted as alternating rising and falling edges and each edge must not be less than its previous edge. The number of edges must be set to 3 to make one full clock cycle of the generated clock waveform. For example, 1 represents the first source edge, 2 represents the second source edge, and so on.
-edge_shift *edgeShiftList*	Specifies a list of floating point numbers that represents the amount of shift, in nanoseconds, that the specified edges are to undergo to yield the final generated clock waveform. The number of edge shifts specified must be equal to the number of edges specified. The values can be positive or negative; positive indicating a shift later in time, while negative indicates a shift earlier in time. For example, 1 indicates that the corresponding edge is to be shifted by one library time unit.
-combinational	The source latency paths for this type of generated clock only includes the logic where the master clock propagates. The source latency paths do not flow through sequential element clock pins, transparent latch data pins, or source pins of other generated clocks.
-disable	Disables the constraint.

-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Examples

Refer to the following examples.

Example 1

A frequency of -divide_by 2 is used for the generated clock.

```
create_generated_clock -name {gen_clk} -source
   [get_pins {DCM0.CLK0}] [get_pins {BUFGMUX_inst.O}] -divide_by 2
```

Example 2

A generated clock is created whose edges are 1, 3, and 5 of the master clock source. If the master clock period is 30 and the master waveform is {24 30}, then the generated clock period becomes 60 with waveform {24 54}.

```
create_generated_clock -name {genclk} -source
   [get_ports {clk_in1}] [get_nets {dut.clk_out2}] -edges {1 3 5}
```

Example 3

This example shows the generated clock from the previous example with each derived edge shifted by 1 time unit. If the master clock period is 30 and the master waveform is {24 36}, then the generated clock period becomes 60 with waveform {25 55}.

```
create_generated_clock -name {genclk}
   -source [get_ports {clk_in1}] [get_nets {dut.clk_out2}]
   -edges {1 3 5} -edge_shift {1 1 1}
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

Example 4

This example shows the generated clock with the same edges as the master clock, where edge 2 is shifted by 0.8 time unit and edge 3 is shifted by -0.4 time unit. If the master clock period is 4 and the master waveform is {0 2}, then the generated clock period becomes 3.6 and the waveform is {0 2.8}.

```
create_generated_clock -name {genclk}
   -source [get_ports {clk_in1}] [get_nets {dut.clk_out2}]
   -edges {1 2 3} -edge_shift {0 0.8 -0.4}
```

reset_path

Resets the specified paths to single-cycle timing.

Syntax

The supported syntax for the reset_path constraint is:

reset_path [**-setup**]
[**-from {***objectList***}**]
[**-through {***objectList***}** [**-through {***objectList***} ...**]]
[**-to {***objectList***}**]
[**-disable**]
[**-comment** *commentString*]

Arguments

-setup	Specifies that setup checking (maximum delay) is reset to single-cycle behavior.
-from	Specifies the names of objects to use to find path start points. The -from *objectList* includes: • Clocks • Registers • Top-level input or bi-directional ports) • Black box outputs • Sequential cell clock pins When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points

© 2020 Synopsys, Inc.
Synopsys Confidential Information

-through	Specifies the intermediate points for the timing exception. The -through *objectList* includes: • Combinational nets • Hierarchical ports • Pins on instantiated cells By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, reset_path applies to the paths that pass through a member of each *objectList*. If you use the -through option in combination with the -from or -to options, reset_path applies only if the -from or -to and the -through conditions are satisfied.
-to	Specifies the names of objects to use to find path end points. The -to *objectList* includes: • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs • Sequential cell data input pins If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

set_clock_groups

Specifies clock groups that are mutually exclusive or asynchronous with each other in a design. Clocks created with create_clock are considered synchronous as long as no set_clock_groups constraints specify otherwise. Paths between asynchronous clocks are not considered for timing analysis.

Clock grouping in the FPGA synthesis environment is inclusionary or exclusionary. For example, clk2 and clk3 can each be related to clk1 without being related to each other.

Syntax

set_clock_groups
-asynchronous | -physically_exclusive | -logically_exclusive
[-name *clockGroupname***]**
-group {*clockList***}** [**-group {***clockList***}** …]
-derive
[**-disable**]
[**-comment** *commentString*]

Arguments

-asynchronous	Specifies that the clock groups are asynchronous to each other (the default assumes all clock groups are synchronous). Two clocks are asynchronous with respect to each other if they have no phase relationship at all.
-physically_exclusive	Specifies that the clock groups are physically exclusive to each other. An example is multiple clocks that are defined on the same source pin. Synthesis accepts this option, but treats it as -asynchronous.
-logically_exclusive	Specifies that the clock groups are logically exclusive to each other. An example is multiple clocks that are selected by a multiplexer, but might have coupling with each other in the design. Synthesis accepts this option, but treats it as -asynchronous.
-name {*clockGroupName***}**	Specifies a unique name for a clock grouping. This option allows you to easily identify specified clock groups, which are exclusive or asynchronous with all other clock groups in the design.

-group {*clockList***}**	Specifies a space-separated list of clocks in {*clockList*} that are asynchronous to all other clocks in the design, or asynchronous to the clocks specified in other -group arguments in the same command. If you specify only one group, the clocks in that group are exclusive or asynchronous with all other clocks in the design. Whenever a new clock is created, it is automatically included in the default "other" group that includes all the other clocks in the design. If you specify -group multiple times in a single command execution, the listed clocks are only asynchronous with the clocks in the other groups specified in the same command. You can include a clock in only one group in a single command execution. To include a clock in multiple groups, use multiple set_clock_groups commands. Do not use commas between clock names in the list. See *-group Option* , on page 272.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Restrictions

Be aware of the restrictions for the following set_clock_groups options:

-group Option

Do not insert commas between clock names when you use the -group option, because the tool treats the comma as part of the clock name. This is true for all constraints that contain lists. This means that if you specify the following constraint, the tool generates a warning that it cannot find clk1,:

```
set_clock_groups -asynchronous -group {clk1, clk2}
```

Examples

The following examples illustrate how to use this constraint.

Example 1

This set_clock_groups constraint specifies that clk4 is asynchronous to all other clocks in the design.

```
set_clock_groups -asynchronous -group {clk4}
```

Example 2

This set_clock_groups constraint specifies that clock clk1, clk2, and clk3 are asynchronous to all other clocks in the design. If a new clock called clkx is added to the design, clk1, clk2, and clk3 are asynchronous to it too.

```
set_clock_groups -asynchronous -group {clk1 clk2 clk3}
```

Example 3

The following set_clock_groups constraint has multiple -group arguments, and specifies that clk1 and clk2 are asynchronous to clk3 and clk4.

```
set_clock_groups -asynchronous -group {clk1 clk2}
   -group {clk3 clk4}
```

Example 4

The following set_clock_groups constraint specifies that clk1 and clk2 which were synchronous when defined with the create_clock command, are now asynchronous.

```
create_clock [get_ports {c1}] -name clk1 -period 10
create_clock [get_ports {c2}] -name clk2 -period 16
create_clock [get_ports {c3}] -name clk3 -period 5
set_clock_groups -asynchronous -group [get_clocks {clk1}]
   -group [get_clocks {clk2}]
```

The following constructs are equivalent:

```
set_clock_groups -asynchronous -group [get_clocks {clk1}]
```

```
set_clock_groups -asynchronous -group {clk1}
```

Example 5

The following constraint specifies that test|clkout0_derived_clock_CLKIN1 and test|clkout1_derived_clock_CLKIN1 are asynchronous to all other clocks in the design:

```
set_clock_groups -asynchronous -group [get_clocks {*clkout*}]
```

Example 6

This example defines the clock on the u1.clkout0 net is asynchronous to all other clocks in the design:

```
set_clock_groups -asynchronous -group [get_clocks -of_objects
{n:u1.clkout0}]
```

Examples of Asynchronous Clocks

Example 1: Multiple -group Arguments for Asynchronous Clock Definition

This method uses multiple -group arguments in one constraint:

```
set_clock_groups -asynchronous -group {clk1 clk2} -group {clk3
bclk4} -group {clk5 cclk6}
```

With this constraint, members of the same group are synchronous, but relationships between clocks from different groups defined in this constraint are asynchronous. This has the following implications:

- clk1 and clk2 are synchronous to each other, but asynchronous to clocks in all other groups defined in this constraint
- clk3 and clk4 are synchronous to each other, but asynchronous to clocks in all other groups defined in this constraint
- clk5 and clk6 are synchronous to each other, but asynchronous to clocks in all other groups defined in this constraint

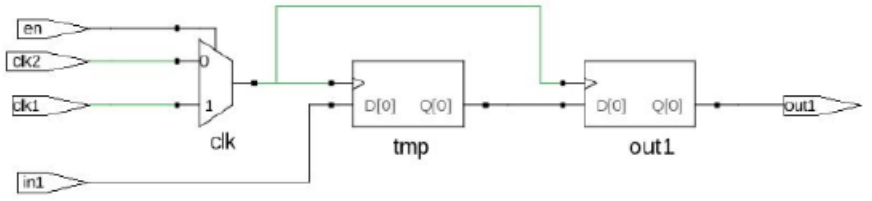
Example 2: Single -group Argument for Asynchronous Clock Definition

Asynchronous clocks defined with a single -group argument in a constraint are asynchronous to all other clocks in the design. You can specify multiple such constraints. In this example, all six clocks are asynchronous, because each individual constraint makes that clock asynchronous to all others.

```
set_clock_groups -asynchronous -group {clk1}
set_clock_groups -asynchronous -group {clk2}
set_clock_groups -asynchronous -group {clk3}
set_clock_groups -asynchronous -group {clk4}
set_clock_groups -asynchronous -group {clk5}
set_clock_groups -asynchronous -group {clk6}
```

Examples of Defining Clocks for Clock Muxes

The definition of clocks that are to be muxed together varies slightly, depending on whether the clocks have the same frequency or not. The following procedures use this example as an illustration:



Defining Muxed Clocks with Different Frequencies

If the clocks are asynchronous, separate clock paths must be defined, as described below.

1. Define the clocks with create_clock constraints.

   For the example, two clocks are defined:

   ```
   create_clock -name {clk1} [get_nets {clk1}] -period 10.0
       -waveform {0 5.0}
   create_clock -name {clk2} [get_nets {clk2}] -period 10.0 -waveform {0 5.0}
   ```

2. Use multiple set_clock_groups constraints to mark them as asynchronous to each other:

   ```
   set_clock_groups -derive -asynchronous -name {default_clkgroup_0} -group
       [get_clocks {clk1}]
   set_clock_groups -derive -asynchronous -name {default_clkgroup_1} -group
       [get_clocks {clk2}]
   ```

3. Check the timing report.

   For the example, the tool reports two separate clock paths, one for each clock.

Defining Muxed Clocks with the Same Frequency

If the clocks have the same phase and frequency, follow this procedure to define the clocks.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

1. Define the clock at the net connected to the output pin of the mux.

   For example:

   ```
   create_clock -name {clk} [get_nets {clk}] -period 10.0
       -waveform {0 5.0}
   ```

2. Define the mux output clock as asynchronous to all other clocks, using a set_clock_groups constraint:

   ```
   set_clock_groups -derive -asynchronous -name {default_clkgroup_2} -group
       [get_clocks {clk}]
   ```

3. Check the timing report.

   In this case, there should be a single clock path, instead of separate paths.

set_clock_latency

Specifies clock network latency.

Syntax

The supported syntax for the set_clock_latency constraint is:

set_clock_latency
-source
[**-clock {***clockList***}**]
delayValue
{*objectList***}**
[**-disable**]

Arguments

-source	Indicates that the specified delay is applied to the clock source latency.
-clock *clockList*	Indicates that the specified delay is applied with respect to the specified clocks. By default, the specified delay is applied to all specified objects.
delayValue	Specifies the clock latency value.
objectList	Specifies the input ports for which clock latency is to be set

Description

In the FPGA synthesis tools, the set_clock_latency constraint accepts both clock objects and clock aliases. Applying a set_clock_latency constraint on a port can be used to model the off-chip clock delays in a multi-chip environment.

In the above syntax, *objectList* references either input ports with defined clocks or clock aliases defined on the input ports. When more than one clock is defined for an input port, the -clock option can be used to apply different latency values to each alias.

Restrictions

The following limitations are present in the FPGA synthesis environment:

- Clock latency can only be applied to clocks defined on input ports.
- The set_clock_latency constraint is only used for source latency.
- The constraint only applies to port clock objects.
- Latency on clocks defined with create_generated_clock is not supported.

set_clock_uncertainty

Specifies the uncertainty (skew) of the specified clock networks.

Syntax

The supported syntax for the set_clock_uncertainty constraint is:

set_clock_uncertainty
{*objectList***}**
-from *fromClock* |**-rise_from** *riseFromClock* | **-fall_from** *fallFromClock*
-to *toClock* |**-rise_to** *riseToClock* | **-fall_to** *fallToClock*
value

Arguments

objectList	Specifies the clocks for simple uncertainty. The uncertainty is applied to the capturing latches clocked by one of the specified clocks. You must specify either this argument or a clock pair with the -from/-rise_from/-fall_from and -to/-rise_to/-fall_to options; you cannot specify both an object list and a clock pair.
-from *fromClock*	Specifies the source clocks for interclock uncertainty. You can use only one of the -from, -rise_from, and -fall_from options and you must specify a destination clock with one of the -to, -rise_to, and -fall_to options.
-rise_from *riseFromClock*	Specifies that the uncertainty applies only to the rising edge of the source clock. You can use only one of the -from, -rise_from, and -fall_from options and you must specify a destination clock with one of the -to, -rise_to, and -fall_to options.
-fall_from *fallFromClock*	Specifies that the uncertainty applies only to the falling edge of the source clock. You can use only one of the -from, -rise_from, and -fall_from options and you must specify a destination clock with one of the -to, -rise_to, and -fall_to options.
-to *toClock*	Specifies the destination clocks for interclock uncertainty. You can use only one of the -to, -rise_to, and -fall_to options and you must specify a source clock with one of the -from, -rise_from, and -fall_from options.
-rise_to *riseToClock*	Specifies that the uncertainty applies only to the rising edge of the destination clock. You can use only one of the -to, -rise_to, and -fall_to options and you must specify a source clock with one of the -from, -rise_from, and -fall_from options.

-fall_to *fallToClock*	Specifies that the uncertainty applies only to the falling edge of the destination clock. You can use only one of the -to, -rise_to, and -fall_to options and you must specify a source clock with one of the -from, -rise_from, and -fall_from options.
value	Specifies a floating-point number that indicates the uncertainty value. Only positive uncertainty numbers are acceptable.

Examples

Refer to the following examples.

Example 1

All paths to registers clocked by clk are specified with setup uncertainty of 0.4 in the following example:

```
set_clock_uncertainty 0.4 -setup [get_clocks clk]
```

Example 2

For this example, interclock uncertainties are specified between clock clk and clk2:

```
set_clock_uncertainty -from [get_clocks clk] -to
   [get_clocks clk2] 0.2

set_clock_uncertainty -from [get_clocks clk2] -to
   [get_clocks clk] 0.1
```

Example 3

For this example, interclock uncertainties are specified between clock clk and clk2 with specific edges:

```
set_clock_uncertainty -rise_from [get_clocks clk2] -to
   [get_clocks clk] 0.5

set_clock_uncertainty -rise_from [get_clocks clk2] -rise_to
   [get_clocks clk] 0.1

set_clock_uncertainty -from [get_clocks clk2] -fall_to
   [get_clocks clk] 0.1
```

set_false_path

Removes timing constraints from particular paths.

Syntax

The supported syntax for the set_false_path constraint is:

set_false_path
[**-setup**]
[**-from {***objectList***}**]
[**-through {***objectList***}** [**-through {***objectList***} ...**]]
[**-to {***objectList***}**]
[**-disable**]
[**-comment** *commentString*]

Arguments

-setup	Specifies that setup checking (maximum delay) is reset to single-cycle behavior.
-from	Specifies the names of objects to use to find path start points. The -from *objectList* includes: • Clocks • Registers • Top-level input or bi-directional ports • Black box outputs • Sequential cell clock pins • When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

-through	Specifies the intermediate points for the timing exception. The -through *objectList* includes: • Combinational nets • Hierarchical ports • Pins on instantiated cells By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_path applies to the paths that pass through a member of each *objectList*. If you use the -through option in combination with the -from or -to options, set_false_path applies only if the -from or -to and the -through conditions are satisfied.
-to	Specifies the names of objects to use to find path end points. The -to *objectList* includes: • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs • Sequential cell data input pins If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

Examples

Refer to the following examples.

Example 1

All the paths from the sequential cell output pins and clock pins in module gen_sub\[*\].u_sub with names matching out* (i.e. registers out1 and out2 in modules gen_sub\[0\].u_sub, gen_sub\[1\].u_sub, and gen_sub\[2\].u_sub) are set as false paths.

```
set_false_path -from [get_pins {gen_sub\[*\]\.u_sub.out*.*}]
```

Note: Only sequential clock pins are valid pins that can be used as start points for a timing path. Note that hierarchical module pins are not valid as starting points for a timing path.

Example 2

All the paths from the sequential cells in module gen_sub\[*\].u_sub with names matching out* (i.e. registers out1 and out2 in modules gen_sub\[0\].u_sub, gen_sub\[1\].u_sub, and gen_sub\[2\].u_sub) are set as false paths.

```
set_false_path -from [get_cells {gen_sub\[*\]\.u_sub.out*}]
```

Example 3

All paths from top-level input ports with names in* are set as false paths.

```
set_false_path -from [get_ports {in*}]
```

Note: Only top-level ports are valid port based start points for timing paths. Do not use the get_ports command to reference hierarchical module pins.

Example 4

All paths with end points clocked by clock clka are set as false paths.

```
set_false_path -to [get_clocks {clka}]
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

set_input_delay

Sets input delay on pins or input ports relative to a clock signal.

Syntax

The supported syntax for the set_input_delay constraint is:

set_input_delay
[**-clock** *clockName* [**-clock_fall**]]
[**-rise**|**-fall**]
[**-min**|**-max**]
[**-add_delay**]
delayValue
{*portPinList***}**
[**-disable**]
[**-comment** *commentString*]

Argument

-clock *clockName*	Specifies the clock to which the specified delay is related. If -clock_fall is used, -clock *clockName* must be specified. If -clock is not specified, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with the period determined by considering the sequential cells in the transitive fanout of each port.
-clock_fall	Specifies that the delay is relative to the falling edge of the clock. The default is the rising edge.
-rise	Specifies that *delayValue* refers to a rising transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed to be equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -rise option is preserved and forward annotated to the place-and-route tool.

-fall	Specifies that *delayValue* refers to a falling transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.
-min	Specifies that *delayValue* refers to the shortest path. If neither -max nor -min is specified, maximum and minimum input delays are assumed equal. **Note:** The synthesis tool does not optimize for hold time violations and only reports -min delay values in the `synlog/topLevel_fpga_mapper.srr_Min` timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.
-max	Specifies that *delayValue* refers to the longest path. If neither -max nor -min is specified, maximum and minimum input delays are assumed equal. **Note:** The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.
-add_delay	Specifies if delay information is to be added to the existing input delay or if is to be overwritten. The -add_delay option enables you to capture information about multiple paths leading to an input port that are relative to different clocks or clock edges.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
delayValue	Specifies the path delay. The *delayValue* must be in units consistent with the technology library used during optimization. The *delayValue* represents the amount of time the signal is available after a clock edge. This represents a combinational path delay from the clock pin of a register.
portPinList	Specifies a list of input port names in the current design to which *delayValue* is assigned. If more than one object is specified, the objects are enclosed in quotes ("") or in braces ({}).

Examples

Refer to the following examples.

Example 1

This example sets an input delay of 1.0 relative to the rising edge of clk.

```
set_input_delay 1.00 -clock clk [get_ports {din1 din2}]
```

Example 2

The following example sets an input delay of 1.0 relative to the rising edge of clk for all inputs in the design.

```
set_input_delay 1.00 -clock clk [all_inputs]
```

Example 3

In this scenario, there are two paths to the input port din1. The input delay for the first path is relative to the rising edge of clk. For the second path, the input delay is relative to the falling edge of clk. The -add_delay option indicates that the new input delay information does not cause old information to be removed.

```
set_input_delay 1.00 -clock clk [get_ports {din1}]

set_input_delay 2.00 -clock clk [get_ports {din1}] -add_delay
   -clock_fall
```

set_max_delay

Specifies a maximum delay target for paths in the current design.

Syntax

The supported syntax for the set_max_delay constraint is:

set_max_delay
[**-from {***objectList***}**]
[**-through {***objectList***}** [**-through {***objectList***} ...**]]
[**-to {***objectList***}**]
delayValue
[**-disable**]
[**-comment** *commentString*]

Arguments

-from Specifies the names of objects to use to find path start points. The -from *objectList* includes:

- Clocks
- Registers
- Top-level input or bi-directional ports
- Black box outputs
- Sequential cell clock pins

When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points. All paths from these start points to the end points in the -from *objectList* are constrained to *delayValue*. If a -to *objectList* is not specified, all paths from the -from *objectList* are affected. If you include more than one object, you must enclose the objects in quotation marks ("") or braces ({}).

-through	Specifies the intermediate points for the timing exception. The -through *objectList* includes: • Combinational nets • Hierarchical ports • Pins on instantiated cells By default, the through points are treated as an OR list. The constraint is applied if the path crosses any points in *objectList*. The max delay value applies only to paths that pass through one of the points in the -through *objectList*. If more than one object is included, the objects must be enclosed either in quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_max_delay applies to the paths that pass through a member of each *objectList*. If you use the -through option in combination with the -from or -to options, set_max_delay applies only if the -from or -to and the -through conditions are satisfied.
-to	Specifies the names of objects to use to find path end points. The -to *objectList* includes: • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs • Sequential cell data input pins If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points. All paths to the end points in the -to *objectList* are constrained to *delayValue*. If a -from *objectList* is not specified, all paths to the -to *objectList* are affected. If you include more than one object, you must enclose the objects in quotation marks ("") or braces ({}).
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.

delayValue	Specifies the value of the desired maximum delay for paths between start and end points. You must express *delayValue* in the same units as the technology library used during optimization. If a path start point is on a sequential device, clock skew is included in the computed delay. If a path start point has an input delay specified, that delay value is added to the path delay. If a path end point is on a sequential device, clock skew and library setup time are included in the computed delay. If the end point has an output delay specified, that delay is added into the path delay.

Examples

Refer to the following example.

Example 1

This example shows how to specify that all paths from cell temp1 to cell temp2 must be less than 4.0 units.

```
set_max_delay -from [get_cells {temp1}] -to [get_cells {temp2}] 4
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

set_multicycle_path

Modifies the single-cycle timing relationship of a constrained path.

Syntax

The supported syntax for the set_multicycle_path constraint is:

set_multicycle_path
[**-setup** |**-hold**]
[**-start** |**-end**]
[**-from {***objectList***}**]
[**-through {***objectList***}** [**-through {***objectList***}** **...**]]
[**-to {***objectList***}**]
pathMultiplier
[**-disable**]
[**-comment** *commentString*]

Arguments

-setup |**-hold**	The option -setup specifies the pathMultiplier to be used for the setup (maximum delay) calculations. The option -hold enables you to over-ride the default hold multiplier—{pathMultiplier – 1}— that is forward annotated to the vendor constraint file. If you use this option, you must specify the hold value for each of the defined multicycle constraints. If you do not provide -setup or -hold, the pathMultiplier is used for setup.
-start | **-end**	Specifies if the multi-cycle information is relative to the period of either the start clock or the end clock. These options are only needed for multi-frequency designs; otherwise start and end are equivalent. The start clock is the clock source related to the register or primary input at the path start point. The end clock is the clock source related to the register or primary output at the path endpoint. The default is to move the setup check relative to the end clock, and the hold check relative to the start clock. A setup multiplier of 2 with -end moves the relation forward one cycle of the end clock. A setup multiplier of 2 with -start moves the relation back one cycle of the start clock. A hold multiplier of 1 with -start moves the relation forward one cycle of the start clock. A hold multiplier of 1 with -end moves the relation back one cycle of the end clock. If you do not provide -start or -end, -end is assumed.

© 2020 Synopsys, Inc.

-from	Specifies the names of objects to use to find path start points. The -from *objectList* includes: • Clocks • Registers • Top-level input or bi-directional ports • Black box outputs • Sequential cell clock pins When the specified object is a clock, all flip-flops, latches, and primary inputs related to that clock are used as path start points. If a -to *objectList* is not specified, all paths from the -from *objectList* are affected. If you include more than one object, you must enclose the objects in quotation marks ("") or braces ({}).
-through	Specifies the intermediate points for the timing exception. The -through *objectList* includes: • Combinational nets • Hierarchical ports • Pins on instantiated cells The multi-cycle values apply only to paths that pass through one of the points in the -through *objectList*. If more than one object is included, the objects must be enclosed either in double quotation marks ("") or in braces ({}). If you specify the -through option multiple times, set_multicycle_delay applies to the paths that pass through a member of each *objectList*. If the -through option is used in combination with the -from or -to options, the multi-cycle values apply only if the -from or -to conditions and the -through conditions are satisfied.
-to	Specifies the names of objects to use to find path end points. The -to *objectList* includes: • Clocks • Registers • Top-level output or bi-directional ports • Black box inputs • Sequential cell data input pins If a specified object is a clock, all flip-flops, latches, and primary outputs related to that clock are used as path end points. If a -from *objectList* is not specified, all paths to the -to *objectList* are affected. If you include more than one object, you must enclose the objects in quotation marks ("") or braces ({})..
-disable	Disables the constraint.

-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
pathMultiplier	Specifies the number of cycles that the data path must have for setup or hold relative to the start point or end point clock before data is required at the end point.

Examples

Refer to the following examples.

Example 1

All the paths from the sequential cell output pins and clock pins in module gen_sub\[*\].u_sub with names matching out* (i.e. registers out1 and out2 in modules gen_sub\[0\].u_sub, gen_sub\[1\].u_sub, and gen_sub\[2\].u_sub) provide 2 timing cycles before the data is required at the end point.

```
set_multicycle_path -from [get_pins{gen_sub\[*\]\.u_sub.out*.*}] 2
```

Note: Only sequential clock pins are pins that can be used as valid start points for a timing path. Note that hierarchical module pins cannot be used as starting points for a timing path.

Example 2

All the paths from the sequential cells in module gen_sub\[*\].u_sub with names matching out* (i.e. registers out1 and out2 in modules gen_sub\[0\].u_sub, gen_sub\[1\].u_sub, and gen_sub\[2\].u_sub) support the timing cycle set to 2.

```
set_multicycle_path -from [get_cells {gen_sub\[*\]\.u_sub.out*}] 2
```

Example 3

All paths from top-level input ports with names in* provide 2 timing cycles before the data is required at the end point.

```
set_multicycle_path -from [get_ports {in*}] 2
```

Note: Only top-level ports are valid port based start points for timing paths. Do not use the get_ports command to reference hierarchical module pins.

Example 4

All paths with end points clocked by clock clka provide 2 timing cycles before the data is required at the end point.

```
set_multicycle_path -to [get_clocks {clka}] 2
```


© 2020 Synopsys, Inc.

set_output_delay

Sets output delay on pins or output ports relative to a clock signal.

Syntax

The supported syntax for the set_output_delay constraint is:

set_output_delay
[**-clock** *clockName* [**-clock_fall**]]
[**-rise**|[**-fall**]
[**-min**|**-max**]
[**-add_delay**]
delayValue
{*portPinList***}**
[**-disable**]
[**-comment** *commentString*]

Arguments

-clock *clockName*	Specifies the clock to which the specified delay is related. If -clock_fall is used, -clock *clockName* must be specified. If -clock is not specified, the delay is relative to time zero for combinational designs. For sequential designs, the delay is considered relative to a new clock with the period determined by considering the sequential cells in the transitive fanout of each port.
-clock_fall	Specifies that the delay is relative to the falling edge of the clock. If -clock is specified, the default is the rising edge.
-rise	Specifies that *delayValue* refers to a rising transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed to be equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -rise option is preserved and forward annotated to the place-and-route tool.

-fall	Specifies that *delayValue* refers to a falling transition on the specified ports of the current design. If neither -rise nor -fall is specified, rising and falling delays are assumed equal. Currently, the synthesis tool does not differentiate between the rising and falling edges for the data transition arcs on the specified ports. The worst case path delay is used instead. However, the -fall option is preserved and forward annotated to the place-and-route tool.
-min	Specifies that *delayValue* refers to the shortest path. If neither -max nor -min is specified, maximum and minimum output delays are assumed equal. **Note:** The synthesis tool does not optimize for hold time violations and only reports -min delay values in the `synlog/topLevel_fpga_mapper.srr_Min` timing report section of the log file. The -min delay values are forward annotated to the place-and-route tool.
-max	Specifies that *delayValue* refers to the longest path. If neither -max nor -min is specified, maximum and minimum output delays are assumed equal. **Note:** The -max delay values are reported in the top-level log file and are forward annotated to the place-and-route tool.
-add_delay	Specifies whether to add delay information to the existing output delay or to overwrite. The -add_delay option enables you to capture information about multiple paths leading to an output port that are relative to different clocks or clock edges.
-disable	Disables the constraint.
-comment *textString*	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
delayValue	Specifies the path delay. The *delayValue* must be in units consistent with the technology library used during optimization. The *delayValue* represents the amount of time that the signal is required before a clock edge. For maximum output delay, this usually represents a combinational path delay to a register plus the library setup time of that register. For minimum output delay, this value is usually the shortest path delay to a register minus the library hold time

portPinList	A list of output port names in the current design to which *delayValue* is assigned. If more than one object is specified, the objects are enclosed in double quotation marks ("") or in braces ({}).

Examples

Refer to the following examples.

Example 1

This example sets an output delay of 1.00 relative to the rising edge of clk for all the output ports in the design.

```
set_output_delay 1.00 -clock clk [all_outputs]
```

Example 2

The following example sets an output delay of 2.00 relative to the falling edge of clk for the output port dout1. The -add_delay option indicates that this delay value is to be added to any existing output delays defined on this port.

```
set_output_delay 2.0 -clock clk [get_ports {dout1}] -add_delay
   -clock_fall
```

set_reg_input_delay

Speeds up paths feeding a register by a given number of nanoseconds.

Syntax

set_reg_input_delay {*registerName***}** [**-route** *ns*] [**-disable**] [**-comment** *textString*]

Arguments

registerName	A single bit, an entire bus, or a slice of a bus.
-route	Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths to the register.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
-disable	Disables the constraint.

Description

The set_reg_input_delay timing constraint speeds up paths feeding a register by a given number of nanoseconds. The Synopsys FPGA synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with create_clock). Use this constraint to speed up the paths feeding a register. For information about the equivalent SCOPE spreadsheet interface, see *Registers*, on page 231.

Use this constraint instead of the legacy constraint, define_reg_input_delay.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

set_reg_output_delay

Speeds up paths coming from a register by a given number of nanoseconds.

Syntax

set_reg_output_delay {*registerName***}** [**-route** *ns*] [**-disable**] [**-comment** *textString*]

Arguments

registerName	A single bit, an entire bus, or a slice of a bus.
-route	Advanced user option that you use to tighten constraints during resynthesis, when the place-and-route timing report shows the timing goal is not met because of long paths from the register.
-comment	Allows the command to accept a comment string. The tool honors the annotation and preserves it with the object so that the exact string is written out when the constraint is written out. The comment remains intact through the synthesis, place-and-route, and timing-analysis flows.
-disable	Disables the constraint.

Description

The set_reg_output_delay constraint speeds up paths coming from a register by a given number of nanoseconds. The synthesis tool attempts to meet the global clock frequency goals for a design as well as the individual clock frequency goals (set with create_clock). Use this constraint to speed up the paths coming from a register. For information about the equivalent SCOPE spreadsheet interface, see *Registers*, on page 231.

Use this constraint instead of the legacy constraint, define_reg_output_delay.

Naming Rule Syntax Commands

The FPGA synthesis environment uses a set of naming conventions for design objects in the RTL when your project contains constraint files. The following naming rule commands are added to the constraint file to change the expected default values. These commands must appear at the beginning of

the constraint file before any other constraints. Similarly, when multiple constraint files are included in the project, the naming rule commands must be in the first constraint file read.

set_hierarchy_separator Command

The set_hierarchy_separator command redefines the hierarchy separator character (the default separator character is the period in the FPGA synthesis environment). For example, the following command changes the separator character to a forward slash:

```
set_hierarchy_separator {/}
```

Embedded Tcl commands, such as get_pins must be enclosed in brackets **[]** for the software to execute the command. Also, the curly brackets **{}** are required when object names include the escape (\) character or square brackets. For example, the following syntax is honored by the tool:

```
set_hierarchy_separator {/}
create_clock -name {clk1} [get_pins
{pdp_c/ib_phy_c/port_g\.1\.phy_c/c7_g\.gtxe2_common_0_i/GTREFCLK[0]}]
-period {10}
```

set_rtl_ff_names Command

The set_rtl_ff_names command controls the stripping of register suffixes in the object strings of delay-path constraints (for example, set_false_path, set_multicycle_path). Generally, it is only necessary to change this value from its default when constraints that target ASIC designs are being imported from the Design Compiler (in the Design Compiler, inferred registers are given a _reg suffix during the elaboration phase; constraints targeting these registers must include this suffix). When importing constraints from the Design Compiler, include the following command to change the value of this naming rule to {_reg} to automatically recognize the added suffix.

```
set_rtl_ff_names {_reg}
```

For example, using the above value allows the DC exception

```
set_false_path -to [get_cells {register_bus_reg[0]}]
```

to apply to the following object without having to manually modify the constraint:

```
[get_cells {register_bus[0]}]
```


© 2020 Synopsys, Inc.
Synopsys Confidential Information

bus_naming_style Command

The bus_naming_style command redefines the format for identifying bits of a bus (by default, individual bits of a bus are identified by the bus name followed by the bus bit enclosed in square brackets). For example, the following command changes the bus-bit identification from the default *busName*[*busBit*] format to the *busName_busBit* format:

```
bus_naming_style {%s_%d}
```

bus_dimension_separator_style Command

The bus_dimension_separator_style command redefines the format for identifying multi-dimensional arrays (by default, multidimensional arrays such as row 2, bit 3 of array ABC[*n* x *m*] are identified as ABC[2][3]). For example, the following command changes the bus-dimension separator from individual square bracket sets to an underscore:

```
bus_dimension_separator_style {_}
```

The resulting format for the above example is:

```
ABC[2_3]
```

read_sdc Command

Reads in a script in Synopsys FPGA constraint format. The supported syntax for the read_sdc constraint is:

read_sdc *fileName*

Design Constraints

This section describes the constraint file syntax for the following non-timing design constraints:

- define_compile_point, on page 302
- define_current_design, on page 303
- define_io_standard, on page 304

© 2020 Synopsys, Inc.
Synopsys Confidential Information

define_compile_point

The define_compile_point command defines a compile point in a top-level constraint file. You use one define_compile_point command for each compile point you define. For the equivalent SCOPE spreadsheet interface, see *Compile Points*, on page 237. (Compile points are only available for certain technologies.)

This is the syntax:

define_compile_point [**-disable**] **{***moduleName***}**
-type {soft|hard|locked|locked, partition} [**-comment** *textString*]

-disable	Disables a previous compile point definition.
-type	Specifies the type of compile point. This can be soft, hard, locked, or locked, partition. See *Compile Point Types* , on page 440 for more information.

Refer to *Guidelines for Entering and Editing Constraints*, on page 136 for details about the syntax and prefixes for naming objects.

Here is a syntax example:

```
define_compile_point {v:work.prgm_cntr} -type {locked}
```

define_current_design

The define_current_design command specifies the module to which the constraints that follow it apply. It must be the first command in a block-level or compile-point constraint file. The specified module becomes the top level for objects defined in this hierarchy and the constraints applied in the respective block-level or compile-point constraint file.

This is the syntax:

define_current_design {*regionName* | *libraryName***.***moduleName***}**

Refer to *Guidelines for Entering and Editing Constraints*, on page 136 for details about the syntax and prefixes for naming objects.

Here is an example:

```
define_current_design {lib1.prgm_cntr}
```

Objects in all constraints that follow this command relate to prgm_cntr.

© 2020 Synopsys, Inc.

define_io_standard

Specifies a standard I/O pad type to use for various vendor-specific families. See *I/O Standards, on page 236* for details of the SCOPE equivalent.

define_io_standard [**-disable**] **{p:***portName***} -delay_type input|output|bidir syn_pad_type {***IO_standard***}** [*parameter* **{***value***}**...]

In the above syntax:

portName
is the name of the input, output, or bidirectional port.

-delay_type
identifies the port direction which must be input, output, or bidir.

syn_pad_type
is the I/O pad type (I/O standard) to be assigned to *portName*.

parameter
is one or more of the parameters defined in the following table. Note that these parameters are device-family dependent.

Parameter	Function
syn_io_termination	The termination type; typical values are pullup and pulldown.
syn_io_drive	The output drive strength; values include low and high or numerical values in mA.
syn_io_dv2	Switch to use a 2x impedance value (DV2).
syn_io_dci	Switch for digitally-controlled impedance (DCI).
syn_io_slew	The slew rate for single-ended output buffers; values include slow and fast or low and high.

Examples:

```
define_io_standard {p:DATA1[7:0]} -delay_type input
   syn_pad_type {LVCMOS_33} syn_io_slew {high}
   syn_io_drive {12} syn_io_termination {pulldown}

define_io_standard {p:en} -delay_type input
   syn_pad_type {LVCMOS_18} syn_io_dci {DCI}
   syn_io_dv2 {DV2}
```

© 2020 Synopsys, Inc.

SYNOPSYS®
Silicon to Software™

CHAPTER 5

User Interface Commands

The following describe the graphical user interface (GUI) commands available from the menus:

- File Menu, on page 306
- Edit Menu, on page 311
- View Menu, on page 325
- Project Menu, on page 333
- Implementation Options Command, on page 346
- Run Menu, on page 381
- Analysis Menu, on page 400
- HDL Analyst Menu, on page 412
- Options Menu, on page 424
- Web Menu, on page 454
- Help Menu, on page 455

For information about context-sensitive commands accessed from right-click popup menus, see Chapter 6, *GUI Popup Menu Commands*.

© 2020 Synopsys, Inc.

File Menu

Use the File menu for opening, creating, saving, and closing projects and files. The following table describes the File menu commands.

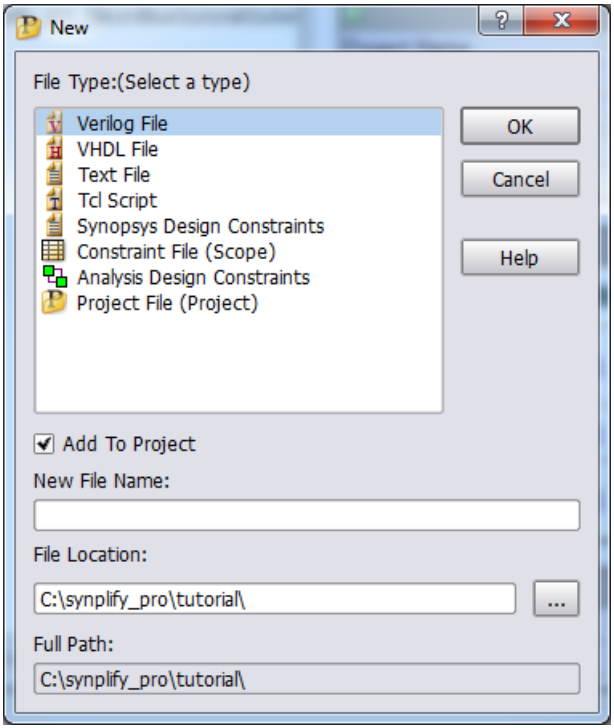
Command	Description
New	Can create any of the following types of files: Text, Tcl Script, VHDL, Verilog, P&R Options file, Design Constraints, Analysis Design Constraints, or Project file. See *New Command* , on page 307.
 Open	Opens a project or file.
Close	Closes a project file.
 Save	Saves a project or a file.
Save As	Saves a project or a file to a specified name.
 Save All	Saves all projects or files.
Print	Prints a file. For more information about printing, see the operating system documentation.
Print Setup	Specify print options.
Create Image	This command is available in the following views: • HDL Analyst Views • FSM Viewer A camera pointer () appears. Drag a selection rectangle around the region for which you want to create an image, then release the mouse button. You can also simply click in the current view, then the Create Image dialog appears. See *Create Image Command* , on page 308.
Build Project	Creates a new project based on the file open in the Text Editor (if active), or lets you choose files to add to a new project. See *Build Project Command* , on page 310.
 Open Project	Opens a project. See *Open Project Command* , on page 311.
New Project	Creates a new project. If a project is already open, it prompts you to save it before creating a new one. If you want to open multiple projects, select Allow multiple projects to be opened in the Project View dialog box. See *Project View Options Command* , on page 429.

Command	Description
Close Project	Closes the current project.
Recent Projects	Lists recently accessed projects. Choose a project listed in the submenu to open it.
Recent files (listed as separate menu items)	Lists the last files you recently opened as separate menu items. Choose a file to open it.
Exit	Exits the session.

New Command

Select File->New to display the New dialog box, where you can select a file type to be created (for example, Verilog, VHDL, text, Tcl script, P&R options, design constraints, analysis design constraints, or project). For most file types, a text editor window opens to allow you to define the file contents. You must provide a file name. You can automatically add the new file to your project by enabling the Add To Project checkbox before clicking OK.

© 2020 Synopsys, Inc.
Synopsys Confidential Information



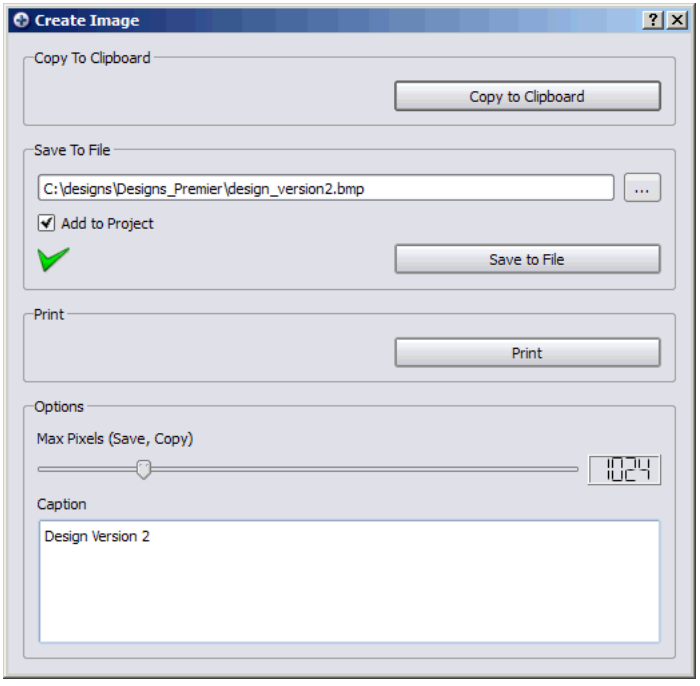
File Type	Opens Window	Directory Name	Extension
Verilog	Text Editor	Verilog	.v
VHDL	Text Editor	VHDL	.vhd
Text	Text Editor	Other	.txt
Tcl Script	Text Editor	Tcl Script	.tcl
FPGA Design Constraints	SCOPE	Constraint	.fdc
Analysis Design Constraints	SCOPE	Analysis Design Constraint	.adc
Project	None	None	.prj

Create Image Command

Select File->Create Image to create a capture image from any of the following views:

- HDL Analyst Views
- FSM Viewer

Drag the camera cursor to define the area for the image. When you release the cursor, the Create Image dialog box appears. Use the dialog box to copy the image, save the image to a file, or to print the image.



Field/Option	Description
Copy to Clipboard	Copies the image to the clipboard so you can paste it into a selected application (for example, a Microsoft Word file). When you copy an image to the clipboard, a green check mark appears in the Copy To Clipboard field.
Save to File	Saves the image to the specified file. You can save the file in a number of formats (platform dependent) including bmp, jpg, png, ppm, tif, xbm, and xpm.
Add to Project	Adds the saved image file to the Images folder in the Project view. This option is enabled by default.

Synopsys Confidential Information

Field/Option	Description
Save to File button	You must click this button to save an image to the specified file. When you save the image, a green check mark appears in the Save To File field.
Print	Prints the image. When you print the image, a green check mark appears in the Print field.
Options	Allows you to select the resolution of the image saved to a file or copied to the clipboard. Use the Max Pixels slider to change the image resolution.
Caption	Allows you to enter a caption for a saved or copied image. The overlay for the caption is at the top-left corner of the image.

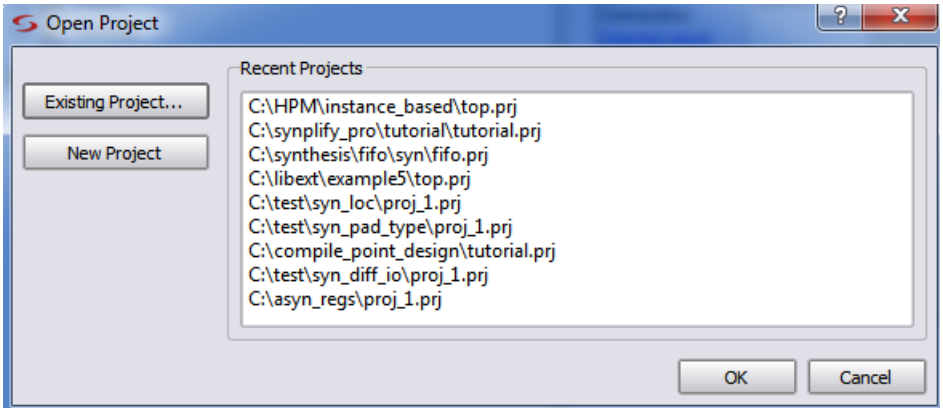
Build Project Command

Select File->Build Project to build a new project. This command behaves differently if an HDL file is open in the Text Editor.

- When an active Text Editor window with an HDL file is open, File->Build Project creates a project with the same name as the open file.
- If no file is open, File->Build Project prompts you to add files to the project using the Select Files to Add to New Project dialog box. The name of the new project is the name of the first HDL file added. See *Add Source File Command*, on page 334.

Open Project Command

Select File->Open Project to open an existing project or to create a new project.



Field/Option	Description
Existing Project	Displays the Open Project dialog box for opening an existing project.
New Project	Creates a new project and places it in the Project view.

Edit Menu

You use the Edit menu to edit text files (such as HDL source files) in your project. This includes cutting, copying, pasting, finding, and replacing text; manipulating bookmarks; and commenting-out code lines. The Edit menu commands available at any time depend on the active window or view (Project, Text Editor, SCOPE spreadsheet, RTL, or Technology views).

The available Edit menu commands vary, depending on your current view. The following table describes all of the Edit menu commands:

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Command	Description
Basic Edit Menu Commands	
 Undo	Cancels the last action.
 Redo	Performs the action undone by Undo.
 Cut	Removes the selected text and makes it available to Paste.
 Copy	Duplicates the selected text and makes it available to Paste.
 Paste	Pastes text that was cut (Cut) or copied (Copy).
Delete	Deletes the selected text.
 Find	Searches the file for text matching a given search string; see *Find Command (Text)* , on page 314. In the RTL view, opens the Object Query dialog box, which lets you search your design for instances, symbols, nets, and ports, by name; see *Find Command (HDL Analyst)* , on page 317. In the project view, searches files for text strings; see *Find Command (In Project)* , on page 315.
Find Next	Continues the search initiated by the last Find.
Find in Files	Performs a string search of the target files. See *Find in Files Command* , on page 321.
Edit Menu Commands for the Text Editor	
Select All	Selects all text in the file.
Replace	Finds and replaces text. See *Replace Command* , on page 323.
Goto	Goes to a specific line number. See *Goto Command* , on page 324.
 Toggle bookmark	Toggles between inserting and removing a bookmark on the line that contains the text cursor.
 Next bookmark	Takes you to the next bookmark.
 Previous bookmark	Takes you to the previous bookmark.
 Delete all bookmarks	Removes all bookmarks from the Text Editor window.

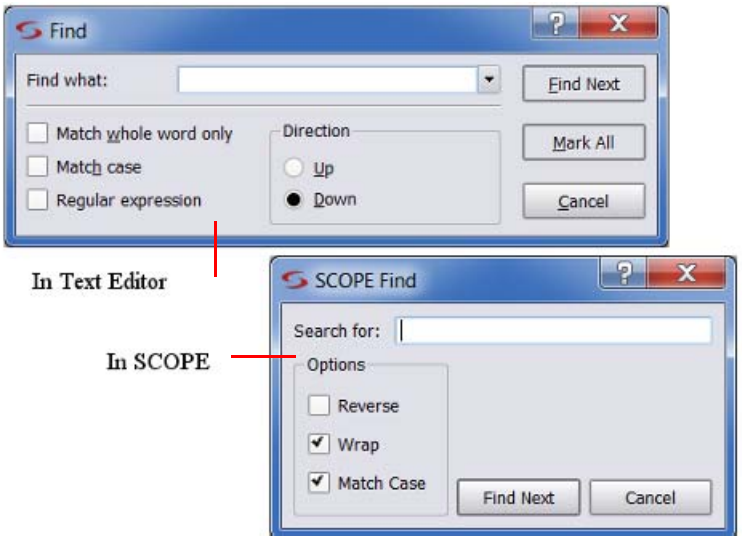
Command	Description
Advanced->Comment Code	Inserts the appropriate comment prefix at the current text cursor location.
Advanced-> Uncomment Code	Removes comment prefix at the current text cursor location.
Advanced->Uppercase	Makes the selected string all upper case.
Advanced->Lowercase	Makes the selected string all lower case.
Select->All	Selects all text in the file (same as All).

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Find Command (Text)

Select Edit->Find to display the Find dialog box. In the SCOPE window, the FSM Viewer, and the Text Editor window, the command has basic text-based search capabilities. Some search features, like regular expressions and line-number highlighting, are available only in the Text Editor. See *Find Command (In Project)*, on page 315, to search for files in the Project.

The HDL Analyst Find command is different; see *Find Command (HDL Analyst)*, on page 317 for details.



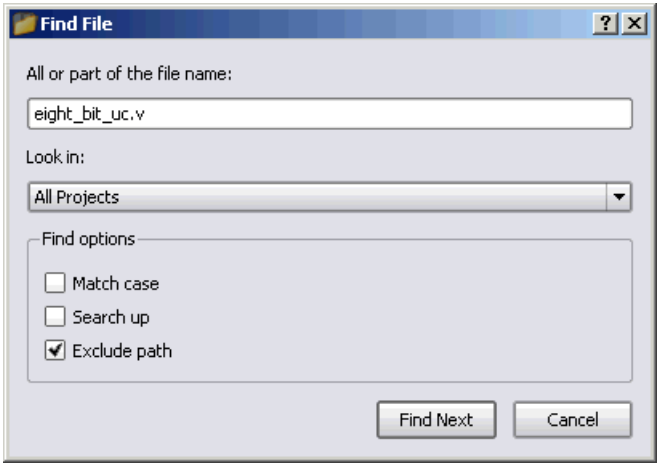
Field/Option	Description
Find What/Search for	Search string matching the text to find. In the text editor, you can use the pull-down list to view and reuse search strings used previously in the current session.
Match whole word only (text editor only)	When enabled, matches the entire word rather than a portion of the word.
Match Case	When enabled, searching is case sensitive.
Regular expression (text editor only)	When enabled, wildcard characters (* and ?) can be used in the search string: ? matches any single character; * matches any string of characters, including an empty string.

© 2020 Synopsys, Inc.

Field/Option	Description
Direction/Reverse	Changes search direction. In the text editor, buttons select the search direction (Up or Down).
Find Next	Initiates a search for the search string (see Find What/Search for). In the text editor, searching starts again after reaching the end (Down) or beginning (Up) of the file.
Wrap (SCOPE only)	When enabled, searching starts again after reaching the end or beginning (Reverse) of the spread sheet.
Mark All (Text editor only)	Highlights the line numbers of the text matching the search string and closes the Find dialog box.

Find Command (In Project)

Select Edit->Find to display the Find File dialog box. In the Project view, the command has basic text-based search capabilities to locate files in the project.

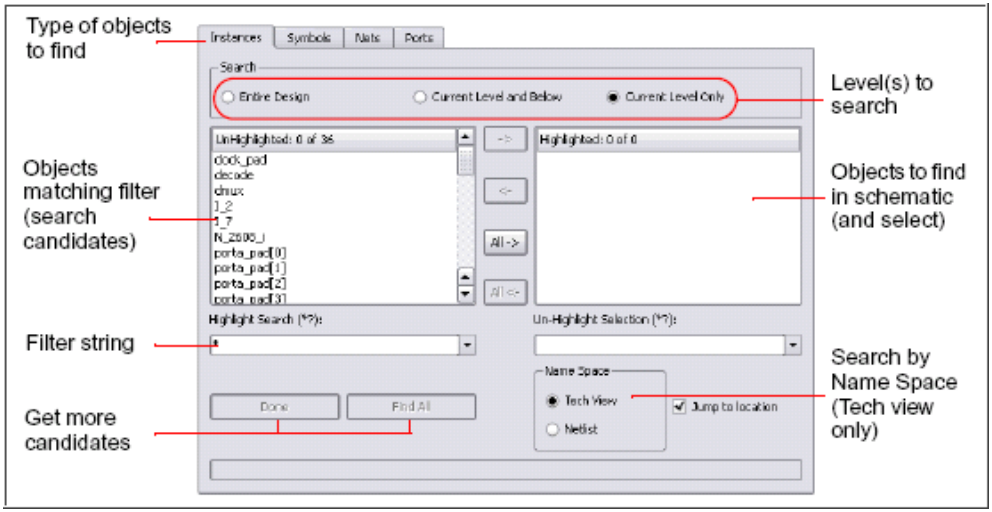


© 2020 Synopsys, Inc.

Field/Option	Description
All or part of the file name	Search string matching the file to find. You can specify all or part of the file name.
Look in	Search for files in all projects or limit the search to files only in the specified project.
Match Case	When enabled, searching is case sensitive.
Search up	Searches in the up direction (search terminates when an end of tree is reached in either direction).
Exclude path	Excludes the path name during the search.
Find Next	Initiates a search for the file name string.

Find Command (HDL Analyst)

In the RTL or Technology view, use Edit->Find to display the Object Query dialog box. For a detailed procedure about using this command, see Using Find for Hierarchical and Restricted Searches, on page 318 of the *User Guide*.



The available Find menu commands vary, depending on your current view. The following table describes all of the Find menu commands:

Field/Option	Description
Instances, Symbols, Nets, Ports	Tabbed panels for finding different kinds of objects. Choose a panel for a given object type by clicking its tab. In terms of memory consumption, searching for Instances is most efficient, and searching for Nets is least efficient.
Search	Where to search: Entire Design, Current Level & Below, or Current Level Only. See Using Find for Hierarchical and Restricted Searches, on page 318 of the *User Guide*.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Field/Option	Description
UnHighlighted	Names of all objects of the current panel type, in the level(s) chosen to Search, that match the Highlight Search (*?) filter. This list is populated by the Find 200 and Find All buttons. To select an object as a candidate for highlighting, click its name in this list. The complete name of the selected object appears near the bottom of the dialog box. You can select part or all of this complete name, then use the Ctrl-C keyboard shortcut to copy it for pasting. You can select multiple objects by pressing the Ctrl or Shift key while clicking; press Ctrl and click a selection to deselect it. The number of objects selected, and the total number listed, are displayed above the list, after the UnHighlighted: label: # selected of # total. To confirm a selection for highlighting and move the selected objects to the Highlighted list, click the -> button.
Highlight Search (*?)	Determines which object names appear in the UnHighlighted area, based on the case-sensitive filter string that you enter. For tips about using this field, see Using Wildcards with the Find Command, on page 321 of the *User Guide.* The filter string can contain the following wildcard characters: • * (asterisk) - matches any sequence of characters • ? (question mark) - matches any single character • . (period) - does not match any characters, but indicates a change in hierarchical level. Wildcards * and ? only match characters within the current hierarchy level; a*b*, for example, will not cross levels to match alpha.beta (where the period indicates a change in hierarchy). If you must match a period character occurring in a name, use \. (backslash period) in the filter string. The backslash prevents interpreting the period as a wildcard. The filter string is matched at each searched level of the hierarchy (the Search levels are described above). Use filter strings that are as specific as possible to limit the number of unwanted matches. Unnecessarily extensive search can be costly in terms of memory performance.
->	Moves the selected names from the UnHighlighted area to the Highlighted area, and highlights their objects in the RTL and Technology views.
<-	Moves the selected names from the Highlighted area to the UnHighlighted area, and unhighlights their objects in the RTL and Technology views.

Field/Option	Description
All ->	Moves all names from the UnHighlighted to the Highlighted area, and highlights their objects in the RTL and Technology views.
<- All	Moves all names from the Highlighted to the UnHighlighted area, and unhighlights their objects in the RTL and Technology views.
Highlighted	Complementary and analogous to the UnHighlighted area. You select object names here as candidates for moving to the UnHighlighted list. (You move names to the UnHighlighted list by clicking the <- button which unselects and unhighlights the corresponding objects.) When you select a name in the Highlighted list, the view is changed to show the (original, unfiltered) schematic sheet containing the object.
Un-Highlight Selection (*?)	Complementary and analogous to the Highlight Search area: selects names in the Highlighted area, based on the filter string you input here.
Jump to location	When enabled, jumps to another sheet if necessary to show target objects.
Name Space: Tech View	Searches for the specified name using the mapped (.srm) database. For more information, see Using Find for Hierarchical and Restricted Searches, on page 318 of the *User Guide*.
Name Space: Netlist	Searches for the specified name using the output netlist file. For more information, see Using Find for Hierarchical and Restricted Searches, on page 318 of the *User Guide*.

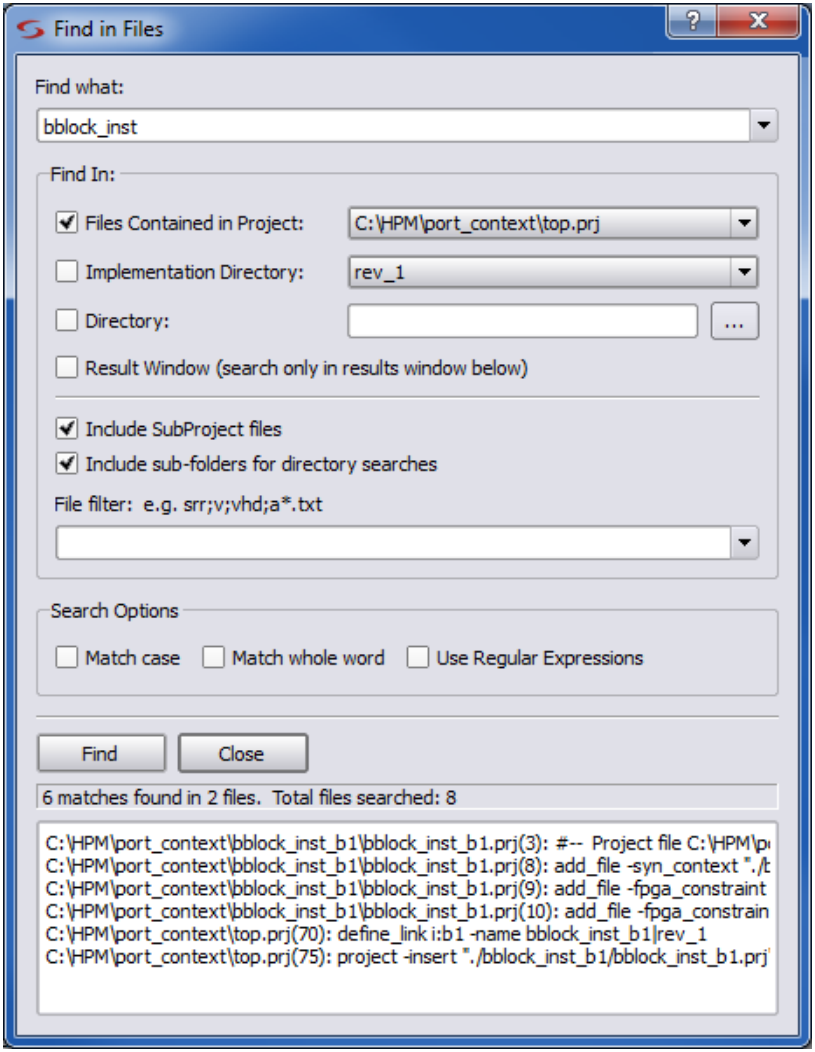
© 2020 Synopsys, Inc.
Synopsys Confidential Information

Field/Option	Description
Find 200	Adds up to 200 more objects that match the filter string to the UnHighlighted list. This button becomes available after you enter a Highlight Search (*?) filter string. This button does not find objects in HDL Analyst views. It matches names of design objects against the Highlight Search (*?) filter and provides the candidates listed in the UnHighlighted list, from which you select the objects to find. Using the Enter (Return) key when the cursor is in the Highlight Search (*?) field is equivalent to clicking the Find 200 button. *Usage note:* Click Find 200 before Find All to prevent unwanted matches in case the Highlight Search (*?) string is less selective than you expect.
Find All	Places all objects that match the Highlight Search (*?) filter string in the UnHighlighted list. This button does not find objects in HDL Analyst views. It matches names of design objects against the Highlight Search (*?) filter and provides the candidates listed in the UnHighlighted list, from which you select the objects to find. (Enter a filter string before clicking this button.) See *Usage Note* for Find 200, above.

For more information on using the Object Query dialog box, see Using Find for Hierarchical and Restricted Searches, on page 318 of the *User Guide*.

Find in Files Command

The Find in Files command searches the defined target for the occurrence of a specified search string. The list of files containing the string is reported in the display area at the bottom of the dialog box. For information on using this feature, see *Searching Files*, on page 102 of the *User Guide*.

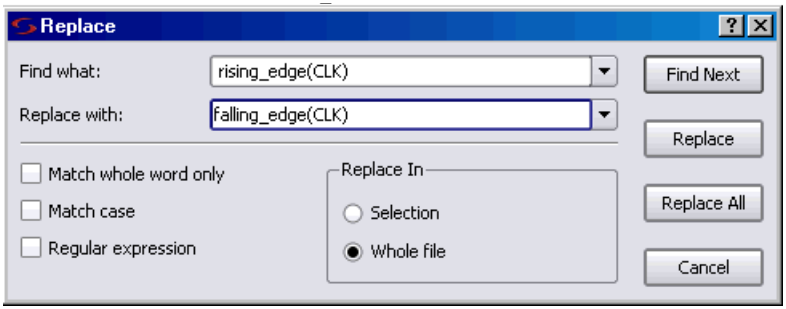


Synopsys Confidential Information
© 2020 Synopsys, Inc.

Field/Option	Description
Find what	Text string object of search.
Files Contained in Project	Drop-down menu identifying the source project of the files to be searched.
Implementation Directory	Drop-down menu restricting project search to a specific implementation or all implementations.
Directory	Identifies directory for files to be searched.
Result Window	Allows a secondary search string (Find what) to be applied to the targets reported from the initial search.
Include sub-folders for directory searches	When checked, extends the search to sub-directories of the target directory.
File filter	Excludes files from the search by filename extension.
Search Options	Standard string search options; check to enable.
Find	Initiates search.
Result Display	List of files containing search string. Status line lists the number of matches in each file and the number of files searched.

Replace Command

Use Edit->Replace to find and optionally replace text in the Text Editor.

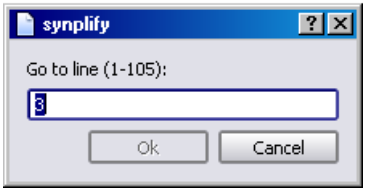


Feature	Description
Find what	Search string matching the text to find. You can use the pull-down list to view and reuse search strings used previously in the current session.
Replace with	The text that replaces the found text. You can use the pull-down list to view and reuse replacement text used previously in the current session.
Match whole word only	Finds only occurrences of the exact string (strings longer than the Find what string are not recognized).
Match case	When enabled, searching is case sensitive.
Regular expression	When enabled, wildcard characters (* and ?) can be used in the search string: ? matches any single character; * matches any string of characters, including the empty string.
Selection	Replace All replaces only the matched occurrence.
Whole file	Replace All replaces all matching occurrences.
Find Next	Initiates a search for the search string (see Find What).
Replace	Replaces the found text with the replacement text, and locates the next match.
Replace All	Replaces all text that matches the search string.

Synopsys Confidential Information

Goto Command

Use Edit->Goto to go to a specified line number in the Text Editor.



View Menu

Use the View menu to set the display and viewing options, choose toolbars, and display result files. The commands in the View menu vary with the active view. The following tables describe the View menu commands in various views.

- View Menu Commands: All Views, on page 325
- View Menu: Zoom Commands, on page 326
- View Menu: RTL and Technology Views Commands, on page 326
- View Menu: FSM Viewer Commands, on page 327

View Menu Commands: All Views

Command	Description
Font Size	Changes the font size in the Project UI of the synthesis tools. You can select one of the following options: • Increase Font Size • Decrease Font Size • Reset Font Size (default size)
Toolbars	Displays the Toolbars dialog box, where you specify the toolbars to display. See *Toolbar Command* , on page 328.
Status Bar	When enabled, displays context-sensitive information in the lower-left corner of the main window as you move the mouse pointer over design elements. This information includes element identification.
Refresh	Updates the UI display of project files and folders.
Output Windows	Displays or removes the Tcl Script/Messages and Watch windows simultaneously in the Project view. Refer to the Tcl Window and Watch Window options for more information.
Tcl Window	When enabled, displays the Tcl Script and Messages windows. All commands you execute in the Project view appear in the Tcl window. You can enter or paste Tcl commands and scripts in the Tcl window. Check for notes, warning, and errors in the Messages window.

© 2020 Synopsys, Inc.

Command	Description
Watch Window	When enabled, displays selected information from the log file in the Watch window.
View Log File	Displays a log file report that includes compiler, mapper, and timing information on your design. See View Log File Command, on page 330.
View Result File	Displays a detailed netlist report.

View Menu: Zoom Commands

Command	Description
 Zoom In  Zoom Out	Lets you Zoom in or out. When selected, a Z-shaped mouse pointer ( ) appears. Zoom in or out on the view by clicking or dragging a box around (lassoing) the region. Clicking zooms in or out on the center of the view; lassoing zooms in or out on the lassoed region. Right-click to exit zooming mode. In the SCOPE spreadsheet, selecting these commands increases or decreases the view in small increments.
Pan	Lets you pan (scroll) a schematic or FSM view using the mouse. If your mouse has a wheel feature, use the wheel to pan up and down. To pan left and right, use the Shift key with the wheel.
 Full View	Zooms the active view so that it shows the entire design.
 Normal View	Zooms the active view to normal size and centers it where you click. If the view is already normal size, clicking centers the view.

View Menu: RTL and Technology Views Commands

These commands are available when the RTL view or Technology view is active. These commands are available in addition to the commands described in *View Menu Commands: All Views*, on page 325 and *View Menu: Zoom Commands*, on page 326.

Command	Description
 Push/Pop Hierarchy	Traverses design hierarchy using the push/pop mode - see *Exploring Design Hierarchy (Standard)* , on page 308 of the *User Guide*.
 Previous Sheet	Displays the previous sheet of a multiple-sheet schematic.
 Next Sheet	Displays the next sheet of a multiple-sheet schematic.
View Sheets	Displays the Goto Sheet dialog box where you can select a sheet to display from a list of all sheets. See *View Sheets Command* , on page 329.
Visual Properties	Toggles the display of information for nets, instances, pins, and ports in the HDL Analyst view. To customize the information that displays, set the values with Options->HDL Analyst Options->Visual Properties. See *Visual Properties Panel* , on page 452.
 Back	Goes backward in the history of displayed sheets for the current HDL Analyst view.
 Forward	Goes forward in the history of displayed sheets for the current HDL Analyst view.
Filter	Filters the RTL/Technology view to display only the selected objects.

View Menu: FSM Viewer Commands

The following commands are available when the FSM viewer is active. These commands are in addition to the common commands described in *View Menu Commands: All Views*, on page 325 and *View Menu: Zoom Commands*, on page 326.

Command	Description
 Filter->Selected	Hides all but the selected state(s).
 Filter->By output transitions	Hides all but the selected state(s), their output transitions, and the destination states of those transitions.
Filter->By input transitions	Hides all but the selected state(s), their input transitions, and the origin states of those transitions.

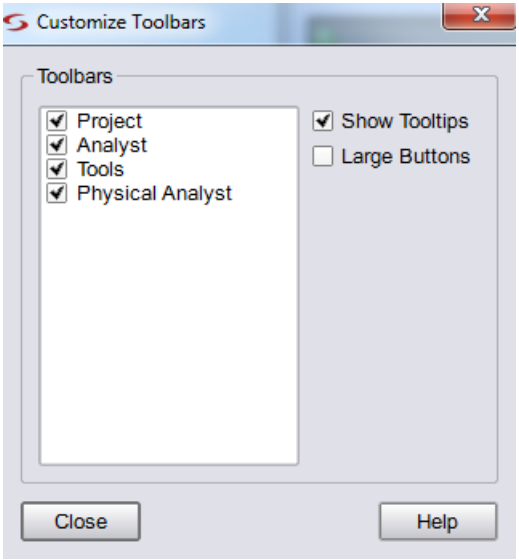
© 2020 Synopsys, Inc.

Command	Description
Filter->By any transition	Hides all but the selected state(s), their input and output transitions, and their predecessor and successor states.
 Unfilter	Restores a filtered FSM diagram so that all the states and transitions are showing.
Cross Probing	Enables cross probing between FSM nodes and RTL view schematic.
Select All States	Selects all the states.
 FSM Table	Toggles display of the transition table.
FSM Graph	Toggles FSM state diagram on or off.
Annotate Transitions	Toggles display of state transitions on or off on FSM state diagram.
Selection Transcription	
Tool Tips	Toggles state diagram tool tips on or off.
FSM Properties	Displays FSM Properties dialog box.
Unselect All	Unselects all states and transitions.

Toolbar Command

Select View->Toolbars to display the Toolbars dialog box, where you can:

- Choose the toolbars to display
- Customize their appearance

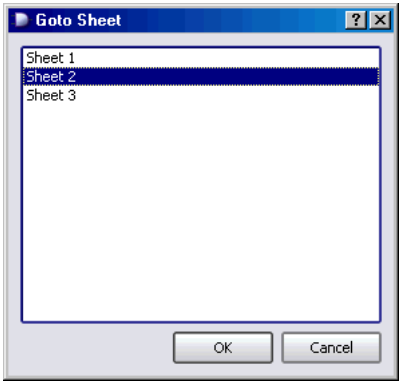


Feature	Description
Toolbars	Lists the available toolbars. Select the toolbars that you want to display.
Show Tooltips	When selected, a descriptive tooltip appears whenever you position the pointer over an icon.
Large Buttons	When selected, large icons are used.

View Sheets Command

Select View->View Sheets to display the Goto Sheet dialog box and select a sheet to display. The Goto Sheet dialog box is only available in an RTL or Technology view, and only when a multiple-sheet design is present.

© 2020 Synopsys, Inc.

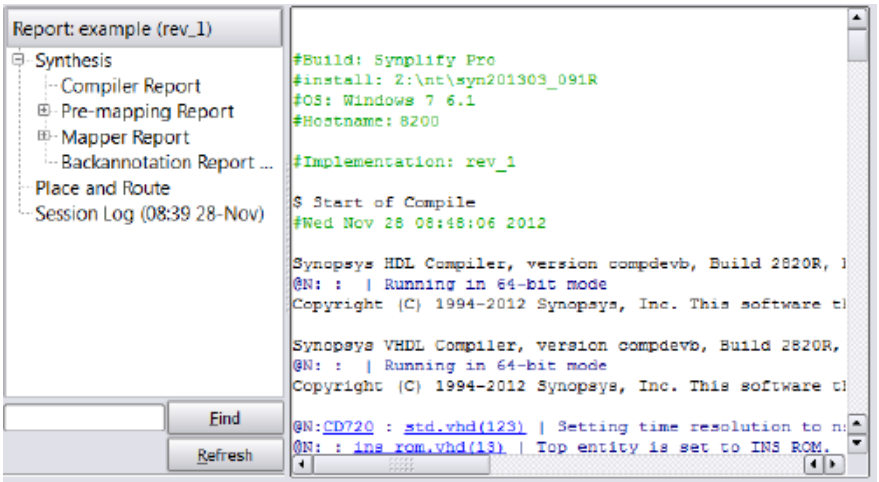


To see if your design has multiple sheets, check the sheet count display at the top of the schematic window.

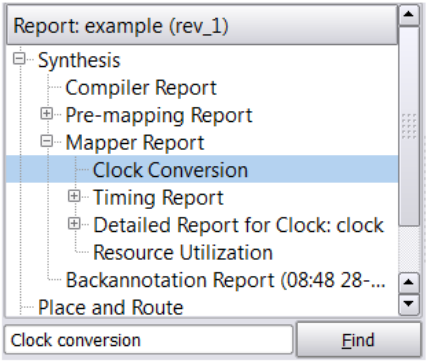
View Log File Command

View->View Log File displays the log file report for your project. The log file is available in either text (*project_name*.srr) or HTML (*project_name*_srr.htm) format. To enable or disable the HTML file format for the log file, select the View log file in HTML option in the Options->Project View Options dialog box.

When opening the log file, a table of contents appears. Selecting an item from the table of contents takes you to the corresponding HTML page. To go back, right-click the HTML page and select Back from the menu.

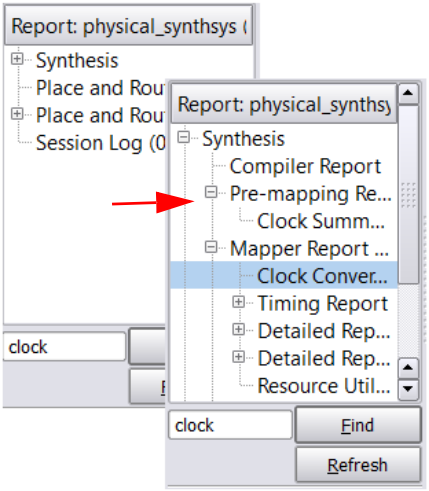


You can use the search field to find an item in the table of contents. Enter all or part of the header name in the search field, then click Find. The log file displays the resulting section.

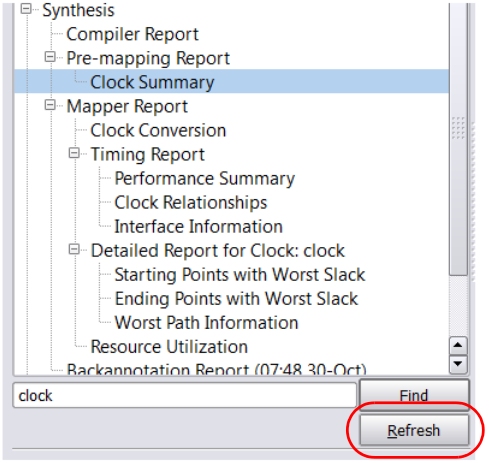


Find searches within collapsed tables. It expands the tables to show your results.

© 2020 Synopsys, Inc.



If the file changes while the search window is open, click the Refresh button to update the table of contents.



Project Menu

You use the Project menu to set implementation options, add or remove files from a project, change project filenames, create new implementations, and archive or copy the project. The Project menu commands change, depending on the view you are in. For example, the HDL Analyst RTL and Technology views only include a subset of the Project menu commands.

The following table describes the Project menu commands.

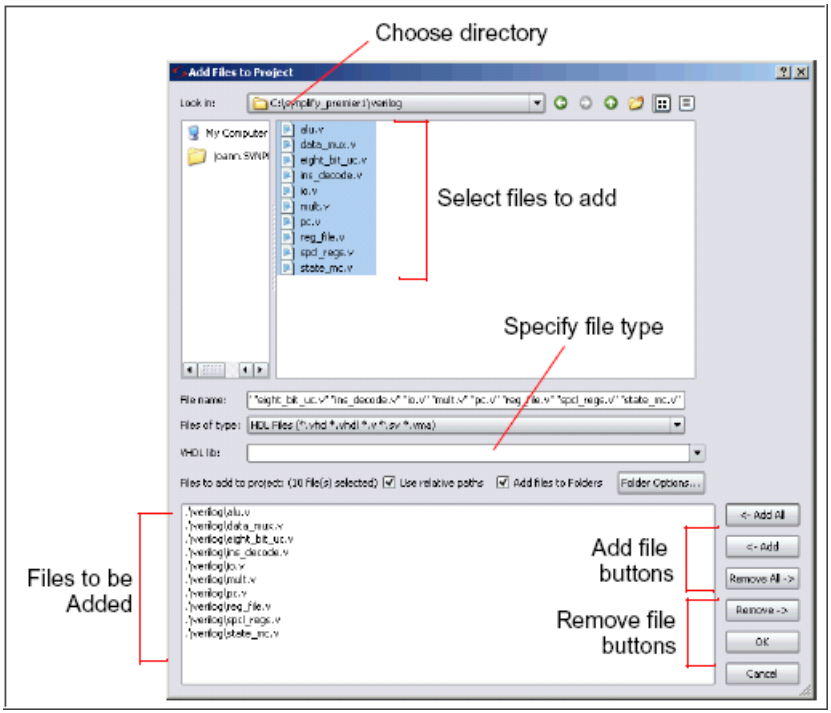
Command	Description
Implementation Options	Displays the Implementation Options dialog box, where you set options for implementing your design. See *Implementation Options Command* , on page 346.
Add Source File	Displays the Select Files to Add to Project dialog box. See *Add Source File Command* , on page 334. Tcl equivalent: **add_file -fileType** *filename*
Remove Implementation	Displays the Remove Implementation dialog box that allows you to remove the selected implementation. See *Remove Implementation* , on page 336. Tcl equivalent: **impl -remove** *implementationName*
Remove File From Project	Removes selected files from your project. Tcl equivalent: **project_file -remove** *filename*
Change File	Replaces the selected file in your project with another that you choose. See *Change File Command* , on page 337. Tcl equivalent: **project_file -name "***originalFile***" "***newFile***"**
Set VHDL Library	Displays the File Options dialog box, where you choose the library (Library Name) for synthesizing VHDL files. The default library is called work. See *Set VHDL Library Command* , on page 337.
Add Implementation	Creates a new implementation for a current design. Each implementation pertains to the same design, but it can have different options settings and/or constraints for synthesis runs. See *Add Implementation Command* , on page 338). Tcl equivalent: **impl -add** *implementation_1 implementation* **-type** *implementationType*

© 2020 Synopsys, Inc.

Command	Description
New Identify Implementation	Creates a new Identify implementation for a current design. To launch the Identify toolset, see the *Identify Instrumentor Command* , on page 387 and *Launch Identify Debugger Command* , on page 389. Tcl equivalent: **impl -add** *implementation_1 implementation*
Archive Project	Archives a design project. Use this command to archive a full or partial project, or to add files to or remove files from an archived project. See *Archive Project Command* , on page 339 for a description of the utility wizard options.
Un-Archive Project	Loads an archived project file to the specified directory. See *Un-Archive Project Command* , on page 340 for a description of the utility wizard options.
Copy Project	Creates a copy of a full or partial design project. See *Copy Project Command* , on page 343 for a description of the utility wizard options.

Add Source File Command

Select Project->Add Source File to add files, such as HDL source files, to your project. This selection displays the Select Files to Add to Project dialog box.



Feature	Description
Look in	The directory of the file to add. You can use the pull-down directory list or the Up One Level button to choose the directory.
File name	The name of a file to add to the project. If you enter a name using the keyboard, then you must include the file-type extension.
Files of type	The type (extension) of files to be added to the project. Only files in the active directory that match the file type selected from the drop-down menu are displayed in the list of files. Use All Files to list all files in the directory.

© 2020 Synopsys, Inc.

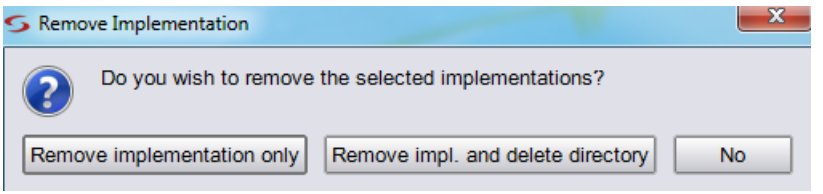
Feature	Description
Files To Add To Project	The files to add to the project. You add files to this list with the <-Add and <-Add All buttons. You remove files from this list with the Remove -> and Remove All -> buttons. For information about adding files to custom folders, see *Creating Custom Folders* , on page 70. Tcl equivalent: **add_file -type** *filename*
Use relative paths	When you add files to the project, you can specify either to use the relative path or full path names for the files.
Add files to Folders	When you add files to the project, you can specify whether or not to automatically add the files to folders. See the Folder Options described below.
Folder Options	When you add files to folders, you can specify the folder name as either the: • Operating System (OS) folder name • Parent path name from a list provided in the display

Remove Implementation

Displays the Remove Implementation dialog box that allows you to remove the selected implementation. You can select any of the following:

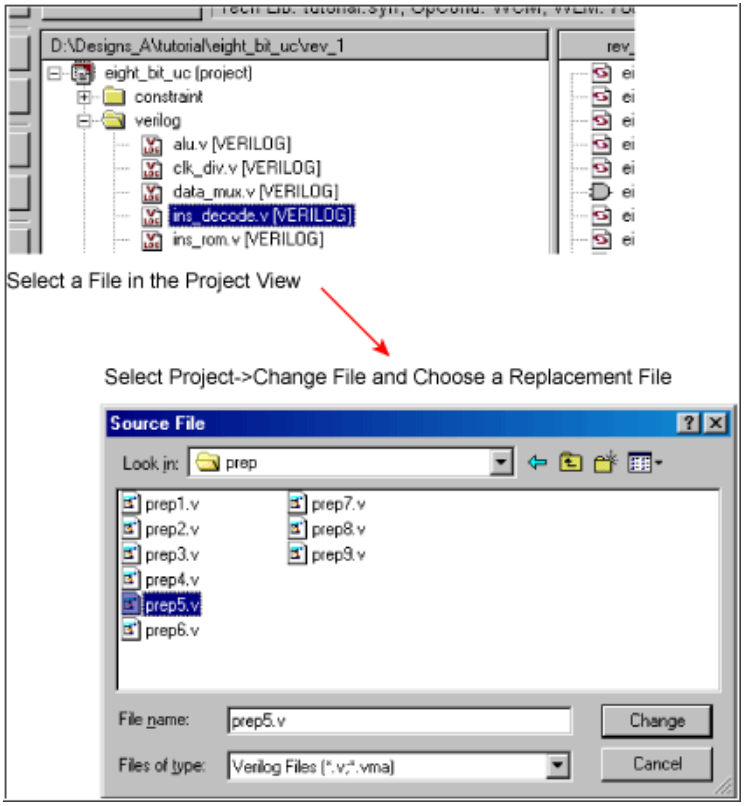
- Remove implementation only - Removes the implementation from the project only.
- Remove impl. and delete directory - Removes the implementation from the project and deletes the directory on the disk.
- No - Do not remove the implementation.

Choose the appropriate option shown in the dialog box below.



Change File Command

Select Project->Change File to replace a file in the project files list with another of the same type. This displays the Source File dialog box, where you specify the replacement file. You must first select the file to replace, in the Project view, before you can use this command.



Set VHDL Library Command

Select Project->Set VHDL Library to display the File Options dialog box, where you view VHDL file properties and specify the VHDL library name. See *File Options Popup Menu Command*, on page 473. This is the same dialog box as that displayed by right-clicking a VHDL filename in the Project view and choosing File Options.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

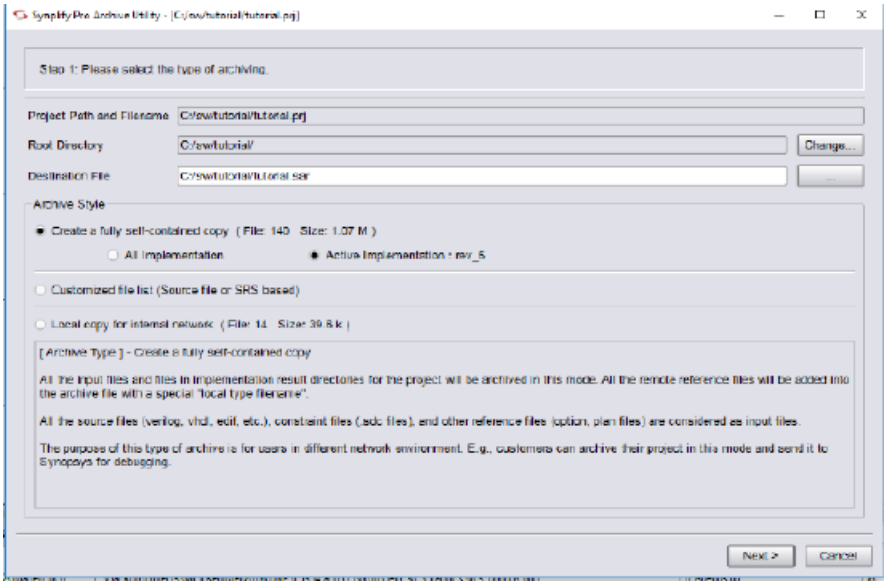
Add Implementation Command

Select Project->Add Implementation to create a new implementation for the selected project. This selection displays the Implementation Options dialog box, where you define the implementation options for the project - see *Implementation Options Command*, on page 346. This is the same dialog box as that displayed by Project->Implementation Options, except that there is no list of Implementations to the right of the tabbed panels.

Archive Project Command

Use the Project->Archive Project command to store files for a design project into a single archive file in Synopsys Proprietary Format (.sar). You can archive an entire project or selected files from the project.

The Archive Project command displays the Synopsys Archive Utility wizard consisting of either two (all files archived) or three (custom file selection) tabs.



Option	Description
Project Path and Filename	Path and filename of the .prj file.
Root Directory	Top-level directory that contains the project files.
Destination Directory	Pathname of the directory to store the archive .sar file.

© 2020 Synopsys, Inc.

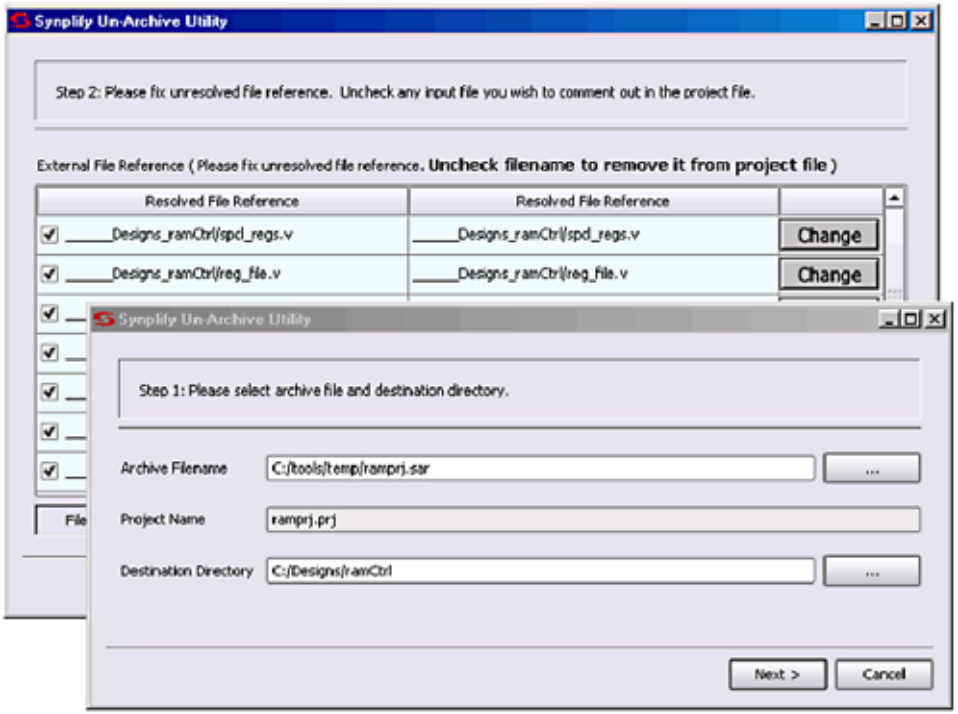
Option	Description
Archive Style	The type of archive: • Create a fully self-contained copy - all project files are archived; includes project input files and result files. • If the project contains more than one implementation: - All Implementation includes all implementations in the project. - Active Implementation includes only the active implementation. • Customized file list - only project files that you select are included in the archive. • Local copy for internal network - only project input files are archived, no result files will be included.
Create Project using	If you select the Customized file list option in the wizard, you can choose one the following options on the second tab: • Source Files - Includes all design files in the archive. You cannot enable the SRS option if this option is enabled. • SRS - Includes all .srs files (RTL schematics) in the archive. You cannot enable the Source Files option when this option is enabled.
Add Extra Files	If you select the Customized file list option in the wizard, you can use this button on the second tab to add additional files to the archive.

For step-by-step details on how to use the archive utility, see *Archive Project Command*, on page 339.

Un-Archive Project Command

Use the Project->Un-Archive Project command to extract the files from an archived design project.

This command displays a Synplify Un-Archive Utility wizard.



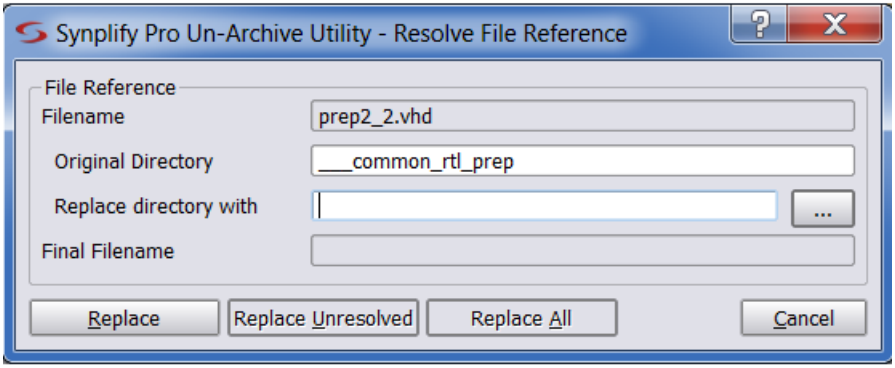
Option	Description
Archive Filename	Path and filename of the .prj file.
Project Name	Top-level directory that contains the project files.
Destination Directory	Pathname of the directory to store the archive .sar file.
Original File Reference/ Resolved File Reference	Displays the files in the archive that will be extracted. You can exclude files from the .sar by unchecking the file in the Original File Reference list. Any unchecked files are commented out in the .prj file. If there are unresolved reference files in the .sar file, you must fix (Resolve button) or uncheck them. Or, if there are files that you want to change when project files are extracted, use the Change button and select files, as appropriate. See *Resolve File Reference*, next for more details.

© 2020 Synopsys, Inc.

For step-by-step details on how to use the un-archive utility, see *Un-Archive Project Command*, on page 340.

Resolve File Reference

When you use the Un-Archive Utility wizard to extract a project, if there are unresolved file references, use the Resolve button next to the file to point to a new file location. You can also optionally replace project files in the destination directory by clicking the Change button next to the file you want to replace. The Change and Resolve buttons bring up the following dialog box:



Option	Description
Filename	Specifies the path and name of the file you want to change or resolve.
Original Directory	Specifies the location of the project at the time it was archived.

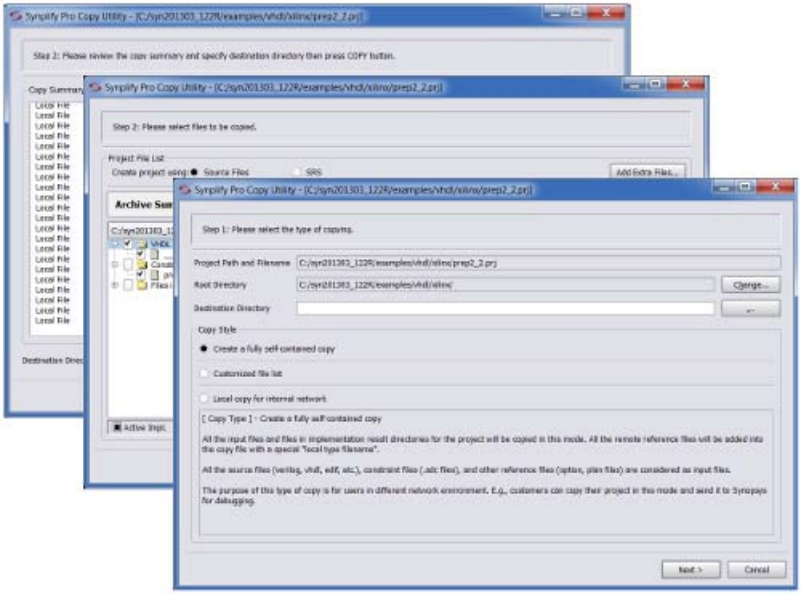
Option	Description
Replace directory with	Specifies the new location of the project files you want to use to replace files.
Final Filename	Specifies the path name of the directory and the file name of the replace file.
Replace buttons	• Replace - replaces only the file specified in the Filename field when the project is extracted. • Replace Unresolved - replaces any unresolved files in the project, with files of the same name from the Replace directory. • Replace All - replaces all files in the archived project with files of the same name from the Replace directory. • To undo any replace-file references, clear the Replace directory with field, then click Replace. This causes the utility to point back to the Original Directory and filenames.

Copy Project Command

Use the Project->Copy Project command to create a copy of a design project. You can copy an entire project or selected files from the project.

The Copy Project command displays the Synopsys Copy Utility wizard consisting of either two (all files copied) or three (custom file selection) tabs.

© 2020 Synopsys, Inc.
Synopsys Confidential Information



Option	Description
Project Path and Filename	Path and filename of the .prj file.
Root Directory	Top-level directory that contains the project files.
Destination Directory	Pathname of the directory to store the archive .sar file.

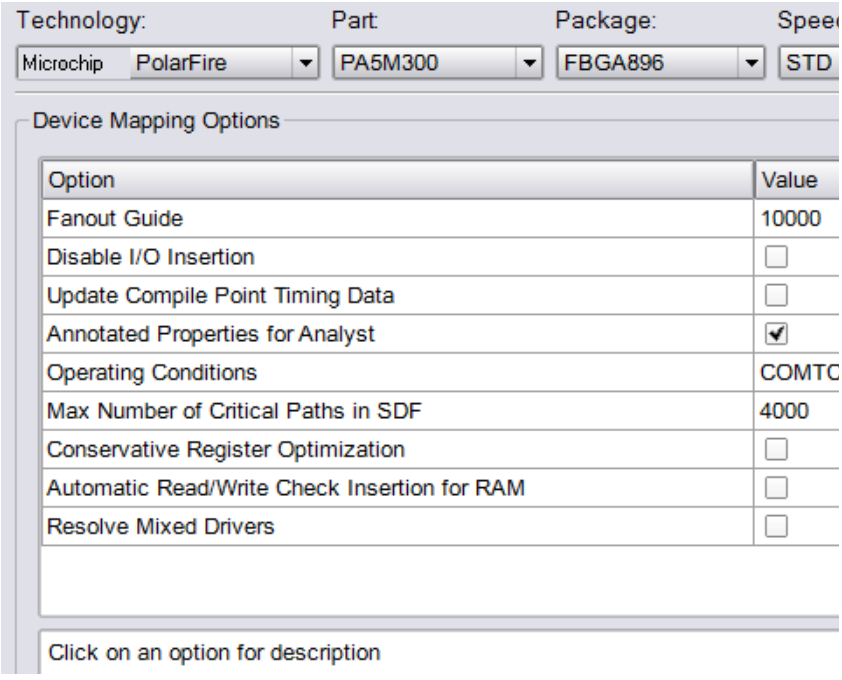
Option	Description
Copy Style	The type of archive: • Create a fully self-contained copy - all project files are archived; includes project input files and result files. • If the project contains more than one implementation: - All Implementation includes all implementations in the project. - Active Implementation includes only the active implementation. • Customized file list - only project files that you select are included in the archive. • Local copy for internal network - only project input files are archived, no result files will be included.
Create Project using	If you select the Customized file list option in the wizard, you can choose one the following options on the second tab: • Source Files - Includes all design files in the archive. You cannot enable the SRS option if this option is enabled. • SRS - Includes all .srs files (RTL schematics) in the archive. You cannot enable the Source Files option if this option is enabled.
Add Extra Files	If you select the Customized file list option in the wizard, you can use this button on the second tab to add additional files to the archive.

For step-by-step details on how to use the copy utility, see *Copy Project Command*, on page 343.

Synopsys Confidential Information

Implementation Options Command

You use the Implementation Options dialog box to define the implementation options for the current project. You can access this dialog box from Project->Implementation Options, by clicking the button in the Project view, or by clicking the text in the Project view that lists the current technology options.



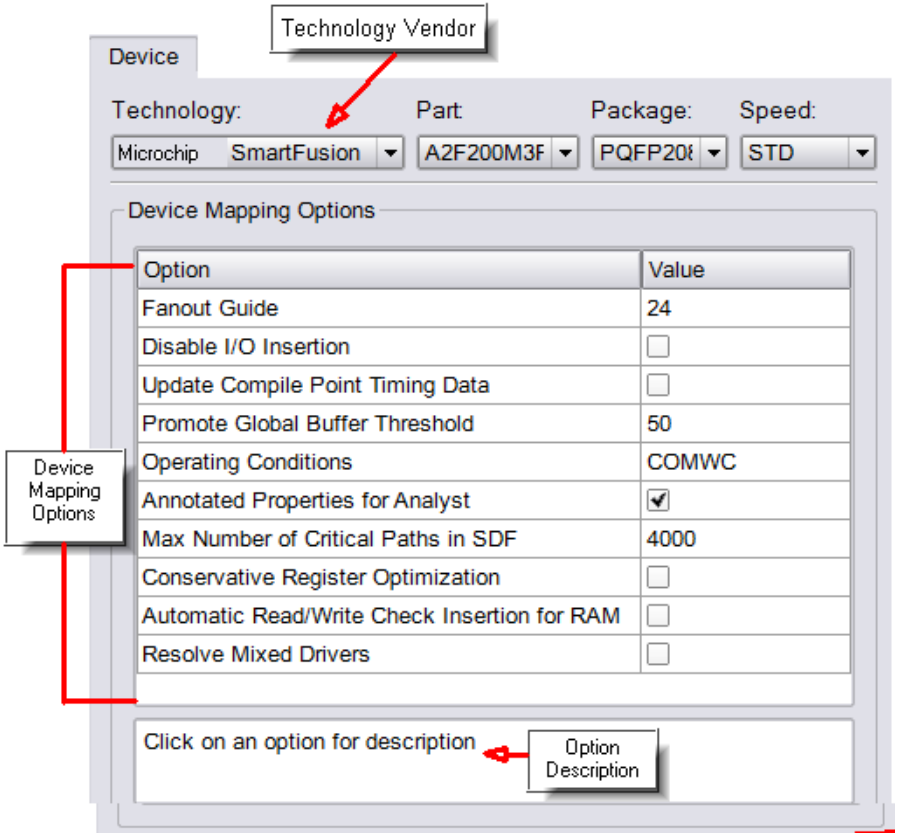
This section describes the following:

- Device Panel, on page 347. For device-specific details of the options, refer to the appropriate vendor chapter.
- Options Panel, on page 349
- Constraints Panel, on page 351
- Implementation Results Panel, on page 353
- Timing Report Panel, on page 355
- High Reliability Panel, on page 357

- VHDL Panel, on page 358
- Verilog Panel, on page 362
- Place and Route Panel, on page 379
- Place and Route Panel, on page 379

Device Panel

You use the Device panel to set mapping options for the selected technology.



© 2020 Synopsys, Inc.

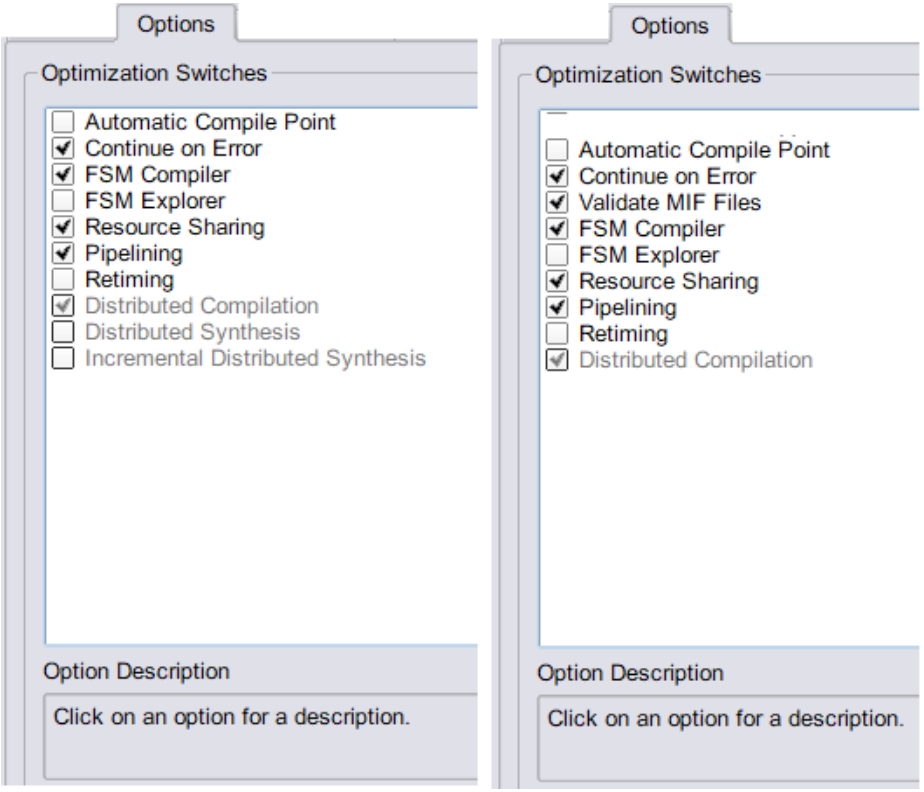
The mapping options vary, depending on the technology. See *Setting Device Options*, on page 78 in the *User Guide* for a procedure, and the relevant vendor sections in this reference manual for technology-specific descriptions of the options.

The table below lists the following category of options. Not all options are available for all tools and technologies.

Option	Description
Technology Vendor	Specify the device technology you want to synthesize. You can also select the part, package, and speed grade to use. For more information, see the appropriate vendor appendix in the *Reference* manual.
Device Mapping Options	The device mapping options vary depending on the device technology you select. For more information, see the appropriate vendor appendix in the *Reference* manual.
Option Description	Click on a device mapping option to display its description in this field. Refer to the relevant vendor sections for technology-specific descriptions of the options.

Options Panel

You use the Options panel of the Implementation Options dialog box to define general options for synthesis optimization. See *Setting Optimization Options*, on page 81 of the *User Guide* for details.



© 2020 Synopsys, Inc.

The following table lists the options alphabetically. Not all options are available for all technologies.

Option	Description
Auto Compile Point	Enables the automatic compile point flow, which can analyze a design and identify modules that can automatically be defined as compile points and mapped in parallel using multiprocessing. See *The Automatic Compile Point Flow* , on page 456 in the *User Guide*. Tcl equivalent: **set_option -automatic_compile_point 0|1**
Continue on Error	When enabled for Synplify Pro synthesis, it only affects compile-point synthesis, allowing the operation to continue and synthesize other compile points. See *Using Continue on Error* , on page 219 in the *User Guide*. Tcl equivalent: **set_option -continue_on_error 0|1**
Enable 64-bit Synthesis	Enables/disables the 64-bit mapping switch. When enabled, this switch allows you to run client programs in 64-bit mode, if available on your system. This option is supported on the Windows and Linux platforms. Tcl equivalent: **set_option -enable64bit 0|1**
FSM Compiler	Determines whether the FSM Compiler is run. See *Running the FSM Compiler* , on page 425 in the *User Guide.* Tcl equivalent: **set_option -symbolic_fsm_compiler 0|1**

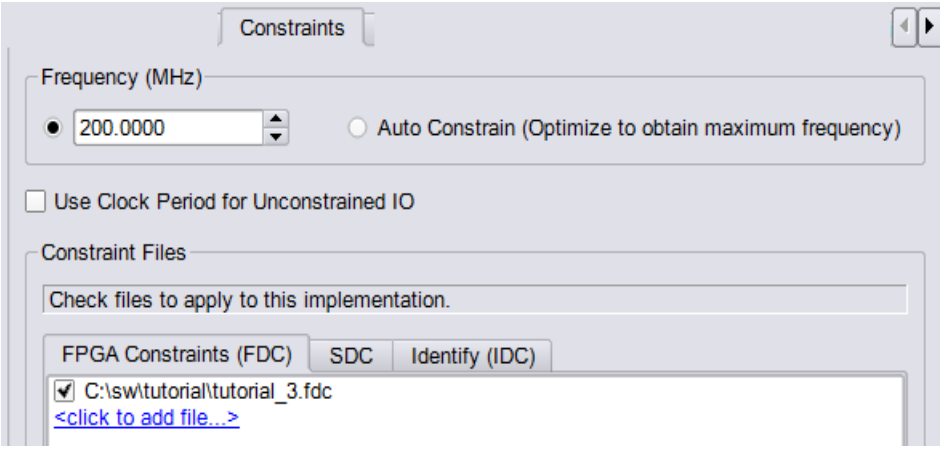
Option	Description
FSM Explorer	Determines whether the FSM Explorer is run. See *Running the FSM Explorer , on page 429* in the *User Guide*. Tcl equivalent: **set_option -use_fsm_explorer 0|1**
Resource Sharing	Determines whether you optimize area by sharing resources. When enabled, this optimization technique runs during the compilation stage of synthesis. Even if it is disabled, the mapper can still flatten the netlist and re-optimize adders, multipliers as needed to improve timing, because this setting does not affect the mapper. See *Sharing Resources , on page 422* for information for how to use this option in the *User Guide*. Enabling this option generates the resource sharing report in the log file (see *Resource Usage Report , on page 160*). Tcl equivalent: **set_option -resource_sharing 0|1**
Retiming	Determines whether the tool moves storage devices across computational elements to improve timing performance in sequential circuits. Note that the tool might retime registers associated with RAMs, DSPs, and generated clocks, regardless of the Retiming setting. See Retiming, on page 406 in the *User Guide*. Tcl equivalent: **set_option -retiming 0|1**

Constraints Panel

You use the Constraints panel of the Implementation Options dialog box to specify target frequency and timing constraint files for design synthesis. Depending on the synthesis tool you are using and the device you specify, the types of constraint files you can apply for the implementation may vary. See the table below for a complete list of option types you can apply.

See *Specifying Global Frequency and Constraint Files, on page 83*, in the *User Guide* for details.

© 2020 Synopsys, Inc.
Synopsys Confidential Information



Option	Description
Frequency	Sets the default global frequency. You can either set the global frequency here or in the Project view. To override the default you set here, set individual clock constraints from the SCOPE interface. Tcl equivalent: **set_option -frequency** *frequency*
Auto Constrain	When enabled and no clocks are defined, the software automatically constrains the design to achieve the best possible timing. It does this by reducing periods of each individual clock and the timing of any timed I/O paths in successive steps. See *Using Auto Constraints* , on page 376 in the *User Guide* for information about using this option. You can also set this option in the Project view. Tcl equivalent: **set_option -frequency auto**

Option	Description
Use clock period for unconstrained IO	Determines whether default constraints are used for I/O ports that do not have user-defined constraints. When disabled, only set_input_delay or set_output_delay constraints are considered during synthesis or forward-annotated after synthesis. When enabled, the software considers any explicit set_input_delay or set_output_delay constraints. In addition, for all ports without explicit constraints, it uses constraints based on the clock period of the attached registers. Both the explicit and implicit constraints are used for synthesis and forward-annotation. The default is off (disabled) for new designs. Tcl equivalent: **set_option -auto_constrain_io 0|1**
Constraint Files FDC	Specifies which timing constraints (FDC) files to use for the implementation. Select the check box to choose a file. For block-level files in the compile-point flows, the Module column shows the name of the module or compile point.
Constraint Files SDC	Specifies which timing constraints (SDC) files to use for the implementation. Select the check box to choose a file. For block-level files in the compile-point flows, the Module column shows the name of the module or compile point.
Identify (IDC)	Specifies the instrumentation design constraints (IDC) files that add compiler pragmas in these files to the design RTL for the instrumented signals and break points. Enable the check box to select a file.

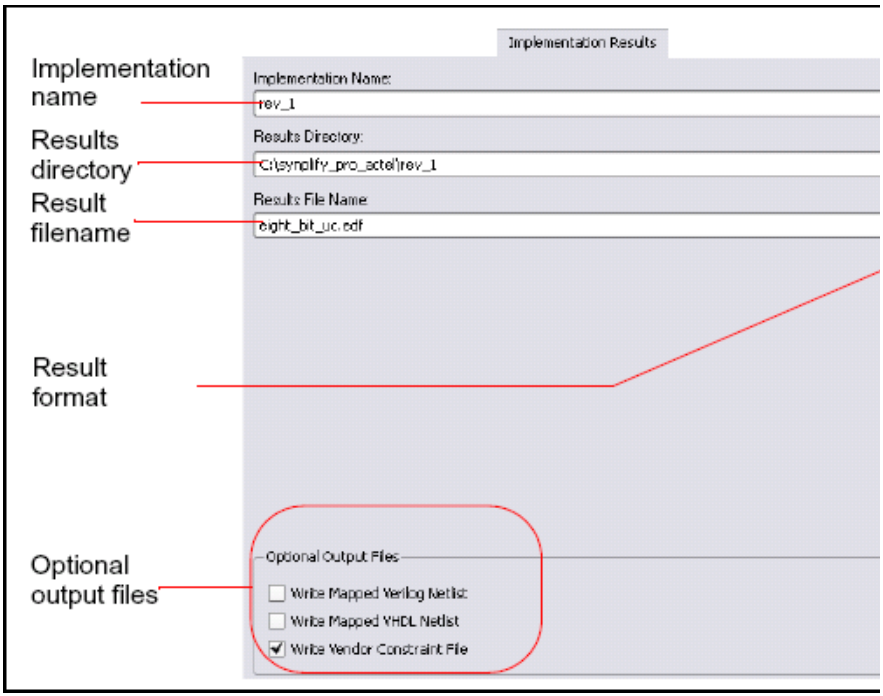
Implementation Results Panel

You use the Implementation Results panel to specify the implementation name (default: rev_1), the results directory, and the name and format of the top-level output netlist file (Result File). You can also specify output constraint and netlist files. See *Specifying Result Options*, on page 85 of the *User Guide* for details.

The results directory is a subdirectory of the project file directory. Clicking the Browse button brings up the Select Run Directory dialog box to allow you to browse for the results directory. You can change the location of the results directory, but its name must be identical to the implementation name.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Select optional output file check boxes to generate the corresponding Verilog netlist, VHDL netlist, or vendor constraint files.

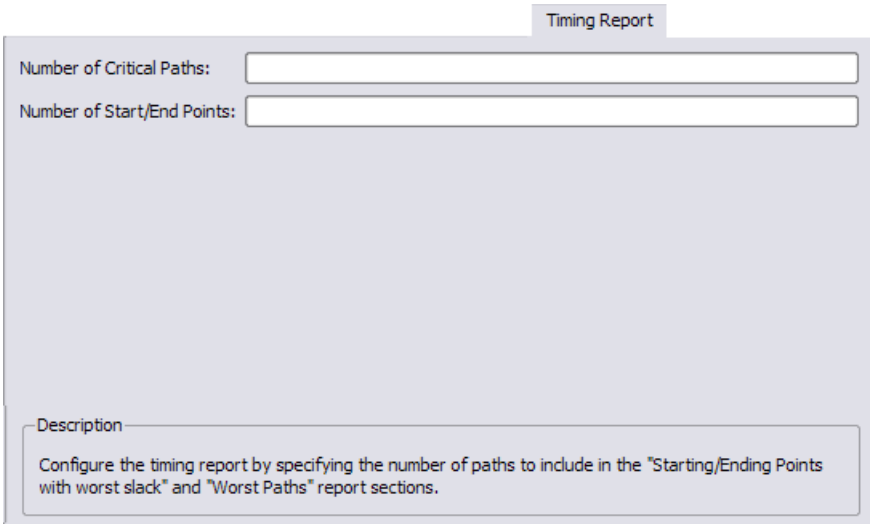


Option	Description
Implementation Name Results Directory Result Base Name	Displays implementation name, directory path for results, and the base name for the result files. Tcl equivalent: **set_option -result_file** *pathtoResultFile*
Result Format	Select the output that corresponds to the technology you are using. See the *Appendices* of the *Reference* manual for a vendor-specific in the *User Guide*list of netlist formats. Tcl equivalent: **set_option -result_format** *format*
Write Mapped Verilog Netlist Write Mapped VHDL Netlist	Generates mapped Verilog or VHDL netlist files. Tcl equivalent: **set_option -write_verilog 0|1** Tcl equivalent: **set_option -write_vhdl 0|1**
Write Vendor Constraint File	Generates a vendor-specific constraint file for forward annotation. Tcl equivalent: **set_option -write_apr_constrain 0|1**

Timing Report Panel

Use the Timing Report panel (Implementation Options dialog box) to set criteria for the (default) output timing report. Specify the number of critical paths and the number of start and end points to appear in the timing report. See *Specifying Timing Report Output*, on page 87 in the *User Guide* for details. For a description of the report, see *Timing Reports*, on page 162.

© 2020 Synopsys, Inc.
Synopsys Confidential Information



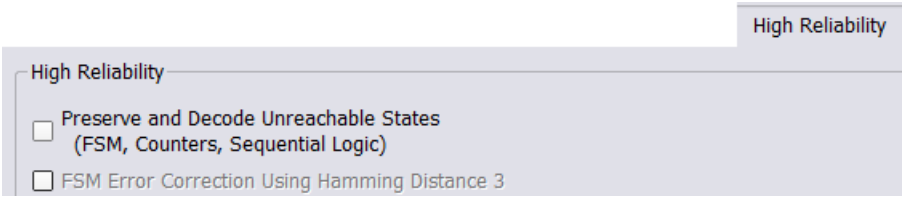
Option	Description
Number of Critical Paths	Set the number of critical paths for the software to report. Tcl equivalent: **set_option -num_critical_paths** *numberOfPaths*
Number of Start/End Points	Specify the number of start and end points to see reported in the critical path sections. Tcl equivalent: **set_option -num_startend_points** *numberOfPoints*

See also:

- Timing Reports, on page 162, for more information on the default timing report, which is affected by the Timing Report panel settings.
- Analysis Menu, on page 400, information on creating additional custom timing reports for certain device technologies.

High Reliability Panel

Use the High Reliability panel (Implementation Options dialog box) to implement safe logic for the design. For more information about high reliability support, see Chapter 15, *Handling High-Reliability Designs*.

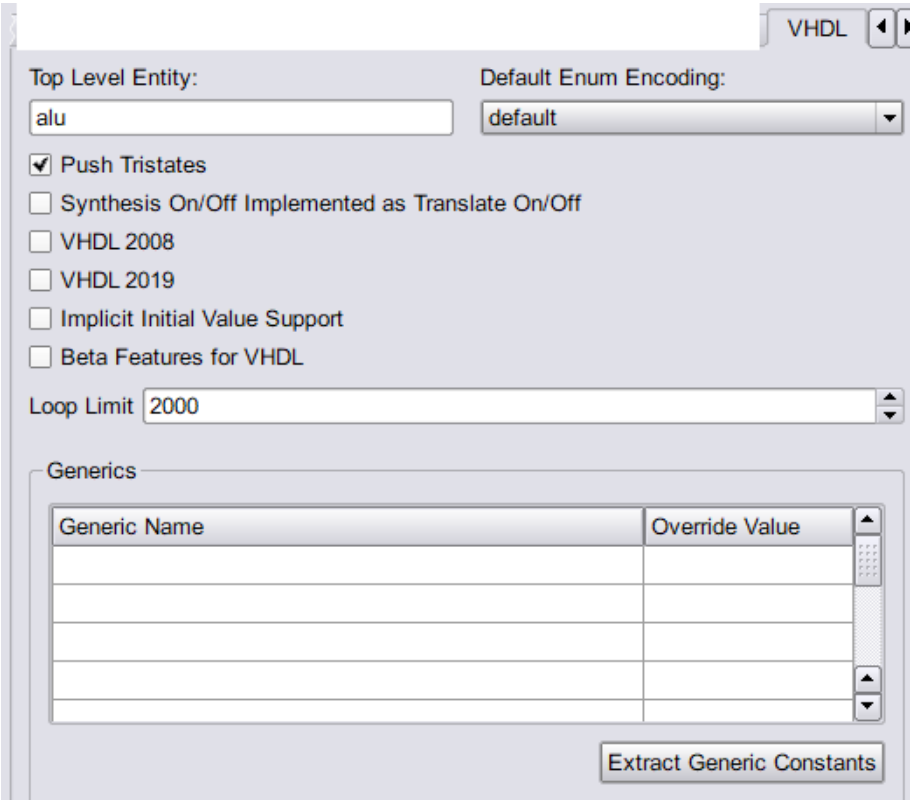


Option	Description
Preserve and Decode Unreachable States (FSM, Counters, Sequential Logic)	When enabled, this option turns off sequential optimizations on all counters, FSMs, and sequential logic, to increase the reliability of the circuit. If you do not want to implement this globally, use the syn_safe_case directive (*syn_safe_case* , on page 225) on individual FSMs. Tcl equivalent: **set_option -safe_case 0|1**

Synopsys Confidential Information

VHDL Panel

You use the VHDL panel in the Implementation Options dialog box to specify various language-related options. With mixed HDL designs, the VHDL and Verilog panels are both available. See *Setting Verilog and VHDL Options*, on page 87, of the *User Guide* for details.



The following table describes the options available.

Feature	Description
Top Level Entity	The name of the top-level entity of your design. If the top-level entity does not use the default work library to compile the VHDL files, you must specify the library file where the top-level entity can be found. To do this, the top-level entity name must be preceded by the VHDL library followed by a period (.). To specify VHDL library files, see *Project Menu* , on page 333 for the Set VHDL Library command, or the *File Options Popup Menu Command* , on page 473. Tcl equivalent: **set_option -top_module** *topLevelName*
Default Enum Encoding	The default enumeration encoding to use. This is only for enumerated types; the FSM compiler automatically determines the state-machine encoding, or you can specify the encoding using the syn_encoding attribute. Tcl equivalent: **set_option -default_enum_encoding** *encodingType* Note: Enable FSM Compiler before attempting to change the encoding style.
Push Tristates	When enabled (default), tristates are pushed across process/block boundaries. For more information, see *Push Tristates Option* , on page 366. Tcl equivalent: **set_option -compiler_compatible 0|1**
Synthesis On/Off Implemented as Translate On/Off	When enabled, the software ignores any VHDL code between synthesis_on and synthesis_off directives. It treats these third-party directives like translate_on/translate_off directives (see *translate_off/translate_on* , on page 283 for details). Tcl equivalent: **set_option -synthesis_onoff_pragma 0|1**
VHDL 2008	When enabled, allows you to use VHDL 2008 language standards. Tcl equivalent: **set_option -vhdl2008 0|1**
VHDL 2019	When enabled, allows you to implement VHDL 2019 conditional analysis. See VHDL 2019 Conditional Analysis, on page 360.

© 2020 Synopsys, Inc.

Feature	Description
Implicit Initial Value Support	When enabled, the compiler passes init values through a syn_init property to the mapper. For more information, see *VHDL Implicit Data-type Defaults* , on page 336. Tcl equivalent: **set_option -supporttypedflt 0|1**
Beta Features for VHDL	Enables use of any VHDL beta features included in the release. Enabling this checkbox is equivalent to including a set_option -hdl_define -set _BETA_FEATURES_ON_ directive in the project file. Tcl equivalent: **set_option -beta_vhfeatures 0|1**
Loop Limit	Overrides the default compiler loop limit value of 2000 in the RTL and sets a new global default. You can apply limits on a per-loop basis using the Verilog loop_limit or the VHDL syn_looplimit directive for individual loops. For details about these directives, see *loop_limit* , on page 47 and *syn_looplimit* , on page 121 in the *Attribute Reference*. Tcl equivalent: **set_option -looplimit** *loopLimitValue*
Generics	Shows generics extracted with Extract Generic Constants. You can override the default and set a new value for the generic constant. The value is valid for the current implementation.
Extract Generic Constants	Extracts generics from the top-level entity and displays them in the table.

VHDL 2019 Conditional Analysis

This section describes how to implement VHDL 2019 conditional analysis identifiers and values:

1. Open the Implementation Options dialog box and select the VHDL tab.
2. Click the VHDL 2019 option to specify VHDL analysis.
3. Add a file, condparams.dat, into the synwork folder of your implementation directory. The file should have one identifier name and one string value pair, per line of the file.

   For example:

```
VHDL_VERSION "2019"
TOOL_TYPE "SYNTHESIS"
DEBUG_LEVEL "2"
```

The VHDL 2019 LRM includes predefined identifiers with their default values:

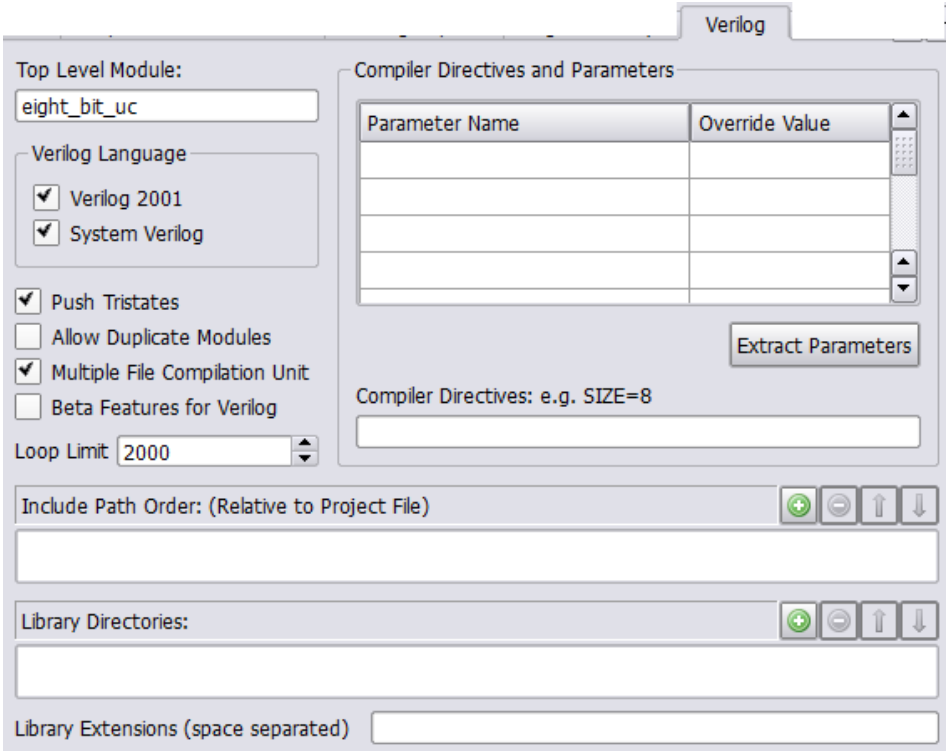
```
VHDL_VERSION "2019"
TOOL_TYPE "SYNTHESIS"
TOOL_VENDOR "SYNOPSYS"
TOOL_NAME "FPGA_COMPILER"
TOOL_EDITION "SYNPLIFY"
TOOL_VERSION "P-2019.03"
```

4. You can edit the condparams.dat file to override these defaults.

© 2020 Synopsys, Inc.

Verilog Panel

You use the Verilog panel in the Implementation Options dialog box to specify various language-related options. With mixed HDL designs, the VHDL and Verilog panels are both available. See *Setting Verilog and VHDL Options*, on page 87 of the *User Guide* for details.


Verilog
Top Level Module:
eight_bit_uc
Verilog Language
Verilog 2001
System Verilog
Push Tristates
Allow Duplicate Modules
Multiple File Compilation Unit
Beta Features for Verilog
Loop Limit
2000
Compiler Directives and Parameters
Parameter Name
Override Value
Extract Parameters
Compiler Directives: e.g. SIZE=8
Include Path Order: (Relative to Project File)
Library Directories:
Library Extensions (space separated)

Feature	Description
Top Level Module	The name of the top-level module of your design. Tcl equivalent: **set_option -top_module** *moduleName*
Compiler Directives and Parameters	Shows design parameters extracted with Extract Parameters. You can override the default and set a new value for the parameter. The value is valid for the current implementation.
Extract Parameters	Extracts design parameters from the top-level module and displays them in the table. See *Compiler Directives and Design Parameters* , on page 369.
Compiler Directives	Provides an interface where you can enter compiler directives that you would normally enter in the code with ‘ifdef and ‘define statements. See *Compiler Directives and Design Parameters* , on page 369.
Verilog Language - Verilog 2001	When enabled, the default Verilog standard for the project is Verilog 2001. When both Verilog 2001 and SystemVerilog are disabled, the default standard is Verilog 95. For information about Verilog 2001, see *Verilog 2001 Support* , on page 33. You can override the default project standard on a per file basis by selecting the file, right-clicking, and selecting the File Options command (see *File Options Popup Menu Command* , on page 473). Tcl equivalent: **set_option -vlog_std v2001**
Verilog Language - SystemVerilog	When enabled, the default Verilog standard for the project is SystemVerilog which is the default standard for all new projects. Enabling SystemVerilog automatically enables Verilog 2001. Tcl equivalent: **set_option -vlog_std sysv**
Push Tristates	When enabled (default), tristates are pushed across process/block boundaries. For details, see *Push Tristates Option* , on page 366. Tcl equivalent: **set_option -compiler_compatible 0|1**

Feature	Description
Allow Duplicate Modules CRM 9001291474	Allows the use of duplicate modules in your design. When enabled, the last definition of the module is used by the software and any previous definitions are ignored. You should not use duplicate module names in your Verilog design, therefore this option is disabled by default. However, if you need to, you can allow for duplicate modules by enabling this option. Tcl equivalent: **set_option -allow_duplicate_modules 0|1**
Multiple File Compilation Unit	When enabled (the default), the Verilog compiler uses the compilation unit for modules defined in multiple files. See SystemVerilog *Compilation Units* , on page 232 for additional information. Tcl equivalent: **set_option -multi_file_compilation_unit 0|1**
Beta Features for Verilog	Enables use of any Verilog beta features included in the release. Enabling this checkbox is equivalent to including a set_option -hdl_define -set _BETA_FEATURES_ON_ directive in the project file. Tcl equivalent: **set_option -beta_vfeatures 0|1**
Loop Limit	Overrides the default compiler loop limit value of 2000 in the RTL and sets a new global default. You can apply limits on a per-loop basis using the Verilog loop_limit or the VHDL syn_looplimit directive for individual loops. For details about these directives, see *loop_limit* , on page 47 and *syn_looplimit* , on page 121 in the *Attribute Reference*. Tcl equivalent: **set_option -looplimit** *loopLimitValue*
Include Path Order (Relative to Project File)	Specifies the search paths for the include commands in the Verilog design files of your project. Use the buttons in the upper right corner of the box to add, delete, or reorder the paths. The include paths are relative. See *Updating Verilog Include Paths in Older Project Files* , on page 68 in the *User Guide* for additional information.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Feature	Description
Library Directories or Files	Specifies all the paths to the directories which contain the Verilog library files to be included in your design for the project. You can also add custom library files with module definitions for the design in a single file. See *Verilog Single Library File Support* , on page 368. The names of files read from the library path must match module names. Mismatches result in error messages. Tcl equivalent: **set_option -library_path** *./libraryPath* or *libraryFile*
Library Extensions (space separated)	Adds library extensions to Verilog library files included in your design for the project and searches the directory paths you specified that contain these Verilog library files. To use library extensions, see *Using Library Extensions for Verilog Library Files* , on page 45 in the *User Guide*. Tcl equivalent: **set_option -libext** *.libextName* Enter a space between each unique library extension.

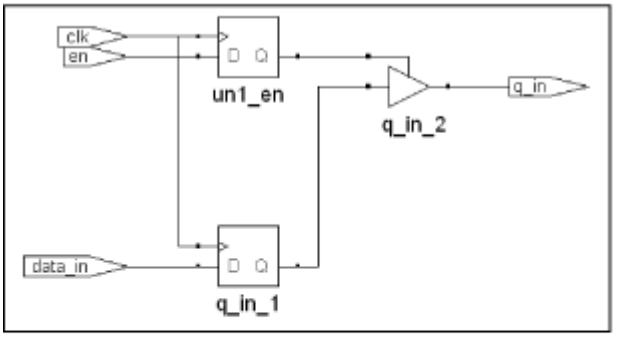
Push Tristates Option

Pushing tristates is a synthesis optimization option you set with Project->Implementation Options->Verilog or VHDL.

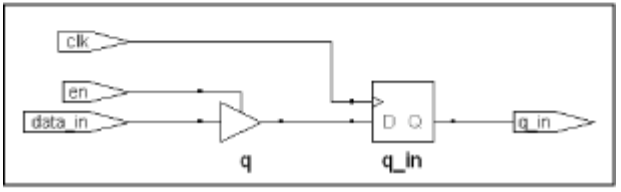
Description

When the Push Tristates option is enabled, the Synopsys FPGA compiler pushes tristates through objects such as muxes, registers, latches, buffers, nets, and tristate buffers, and propagates the high-impedance state. The high-impedance states are not pushed through combinational gates such as ANDs or ORs.

Push Tristates on: tristate is pushed through the flip-flop so that the result matches RTL simulation



Push Tristates off: tristate is not pushed through the flip-flop



If there are multiple tristates, the software muxes them into one tristate and pushes it through. The software pushes tristates through loops and stores the high impedance across multiple cycles in the register.

Advantages and Disadvantages

The advantage to pushing tristates to the periphery of the design is that you get better timing results because the software uses tristate output buffers.

The Synopsys FPGA software approach to tristate inference matches the simulation approach. Simulation languages are defined to store and propagate 0, 1, and Z (high impedance) states. Like the simulation tools, the Synopsys FPGA synthesis tool propagates the high-impedance states instead of producing tristate drivers at the outputs of process (VHDL) or always (Verilog) blocks.

The disadvantage to pushing tristates is that you might use more design resources.

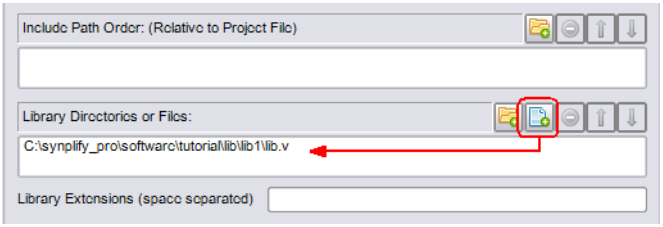
Effect on Other Synthesis Options

Tristate pushing has no effect on the syn_tristatetomux attribute. This is because tristate pushing is a compiler directive, while the syn_tristatetomux attribute is used during mapping.

Verilog Single Library File Support

You can add a single library file to your project for easier migration from a VCS environment and to ensure their behaviors are consistent for the design. To do this, either:

- Select the Add a file icon from the Verilog tab of the Implementation Options panel. Then, specify the library file to be added to the project from the Library Directories and Files option.



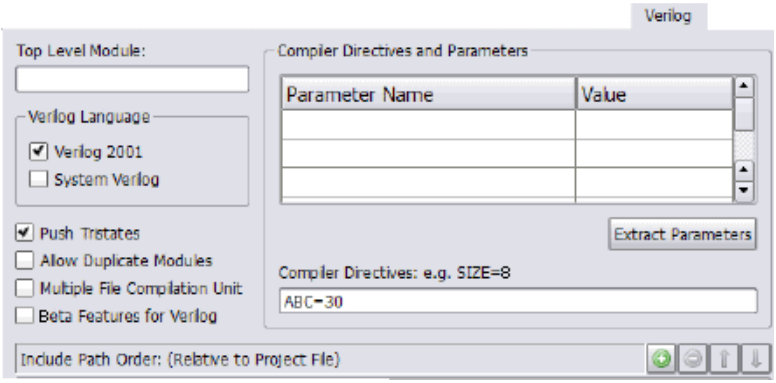
- Add the following Tcl command to your project file:

  set_option -library_path {./*libPath*/*libFile*.v}

Compiler Directives and Design Parameters

When you click the Extract Parameters button in the Verilog panel (Implementation Options dialog box), parameter values from the top-level module are displayed in the table. You can also override the default by setting a new value for the parameter. The value is valid for the current implementation only.

The Compiler Directives field provides an interface where you can enter compiler directives that you would normally enter in the code using ‘ifdef and ‘define statements. Use spaces to separate the statements. The directives you enter are stored in the project file. For example, if you enter the directive shown below to the Compiler Directives field of the Verilog panel:

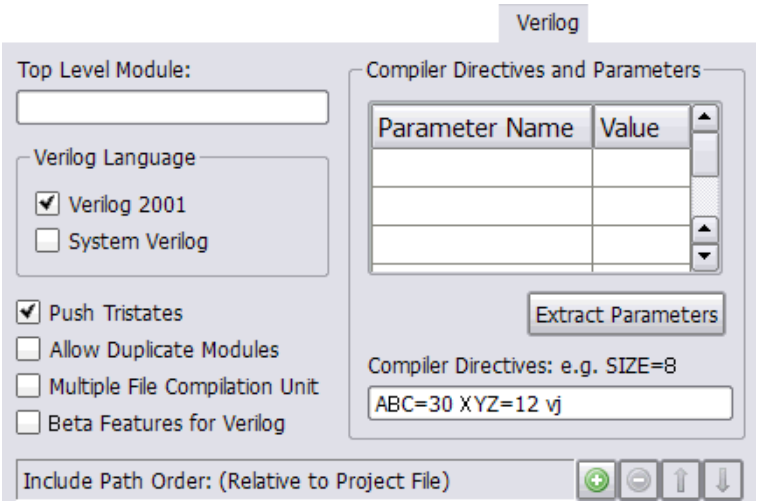


The software writes the following statement to the project file:

```
set_option -hdl_define -set "ABC=30"
```

To define multiple variables in the GUI, use a space delimiter. For example:

Synopsys Confidential Information



The software writes the following statement to the prj file:

```
set_option hdl_define -set "ABC=30 XYZ=12 vj"
```

More information is provided for the following Verilog compiler directives:

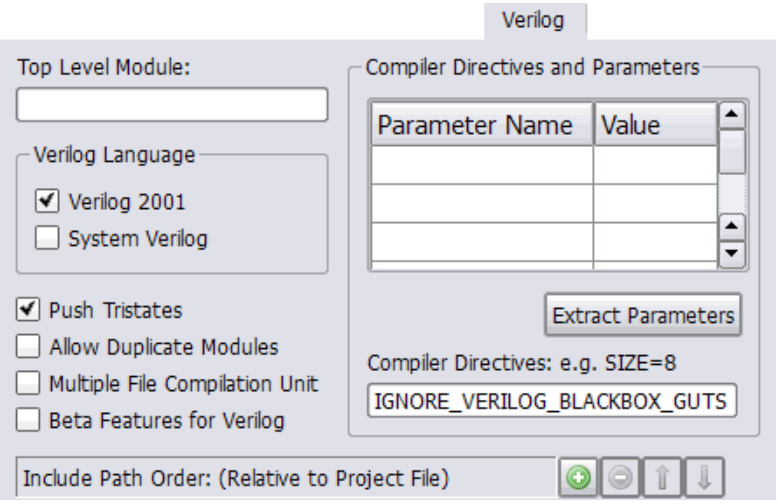
Compiler Directive	Description
__ALLOWNESTEDBLOCKCOMMENTSTART__	Allows for nested comment blocks.
_BETA_FEATURES_ON_	Explicitly enables beta HDL language features.
IGNORE_VERILOG_BLACKBOX_GUTS	Ignores the contents of a black box.
__SEARCHFILENAMEONLY__	Provides workarounds for archive utility limitations.
SYN_COMPATIBLE	Ensure compatibility between different Synopsys tools.
__SYN_COMPATIBLE_INCLUDEPATH__	Specifies that the search path order for includes to be the same as the one used by the simulation tool (VCS).
__SYN_STRICT_MODPORTS__	Requires that modports defined strictly access the associated interface ports specified in the instantiation.

IGNORE_VERILOG_BLACKBOX_GUTS

When you use the syn_black_box directive, the compiler parses the contents of the black box and can determine whether illegal syntax or incorrect code is defined within it. Whenever this occurs, an error message is generated. You can specify the IGNORE_VERILOG_BLACKBOX_GUTS compiler directive to ignore the contents of the black box. However, make sure that the black box is syntactically correct.

If you want the tool to ignore the contents of your black box, set the:

- Built-in compiler directive IGNORE_VERILOG_BLACKBOX_GUTS in the Compiler Directives field of the Verilog panel on the Implementation Options dialog box.



  The software writes the following command to the project file:

  ```
  set_option -hdl_define -set "IGNORE_VERILOG_BLACKBOX_GUTS"
  ```

- `` `define IGNORE_VERILOG_BLACKBOX_GUTS `` directive in the Verilog file.

This option is implemented globally for the project file.

Synopsys Confidential Information

Example of the IGNORE_VERILOG_BLACKBOX_GUTS Directive

Note that the IGNORE_VERILOG_BLACKBOX_GUTS directive ignores the contents of the black box. However, whenever you use this directive, you must first define the ports for the black box correctly. Otherwise, the IGNORE_VERILOG_BLACKBOX_GUTS directive generates an error. See the following valid Verilog example:

```
`define IGNORE_VERILOG_BLACKBOX_GUTS
module b1_fpga1 (A,B,C,D) /* synthesis syn_black_box */;
input B;
output A;
input [2:0] D;
output [2:0] C;
temp;
assign A = B;
assign C = D;

endmodule

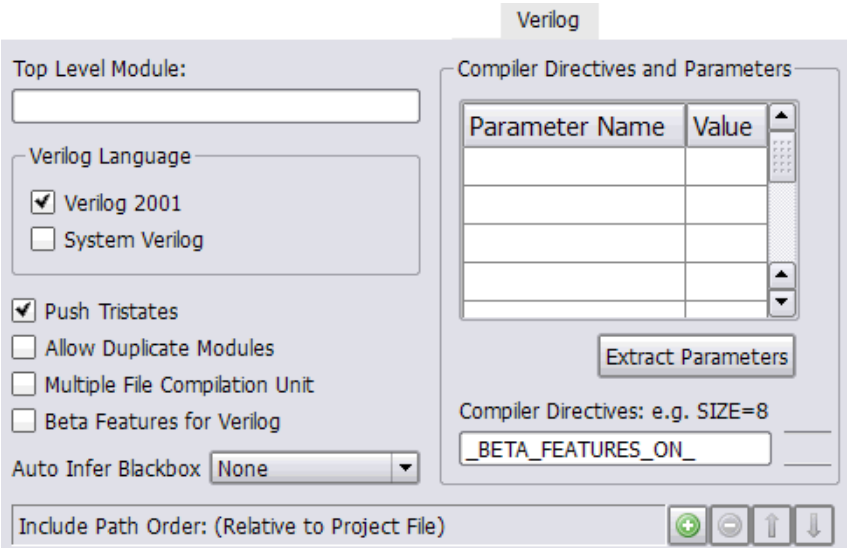
module b1_fpga1_top (inout A, B, inout [2:0] C, D);
b1_fpga1 b1_fpga1_inst(A,B,C,D);
endmodule
```

_BETA_FEATURES_ON_

Beta features for the Verilog, SystemVerilog, or VHDL language must be explicitly enabled. In the UI, a Beta Features checkbox is included on the VHDL or Verilog tab of the Implementations Options dialog box. A _BETA_FEATURES_ON_ compiler directive is also available. This directive is specified with a set_option -hdl_define command added to the project file as shown below:

```
set_option -hdl_define -set _BETA_FEATURES_ON_
```

The directive can also be added to the Compiler Directives field of the Verilog panel.



Current beta features that must be explicitly enabled for the compiler include the following:

HDL Language Constructs	Descriptions
Aggregates	• Multi-assignments for array aggregates • Assignments for the union of variable types

__SEARCHFILENAMEONLY__

This directive provides a workaround for some known limitations of the archive utility.

If Verilog 'include files belong in any of the following categories, you may encounter problems when compiling a design after un-archiving:

1. The include paths have relative paths to the project file.

   ```
   `include "../../../defines.h"
   ```

2. The include paths have absolute paths to the project file.

   ```
   `include "c:/temp/params.h"
   ```


© 2020 Synopsys, Inc.

```
`include "/remote/sbg_home/user/params.h"
```

3. The include paths have the same file names, but are located in different directories relative to the project file.

```
`include "../myflop.v"
…
`include "../../myflop.v"
```

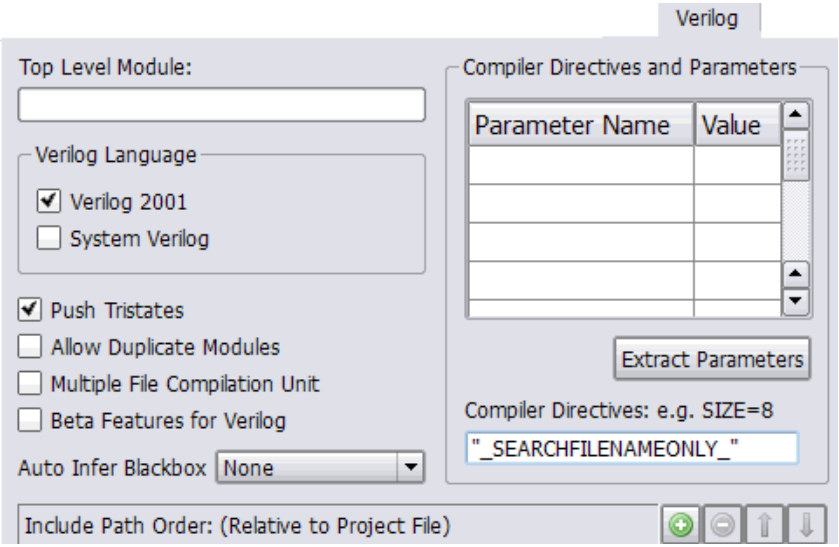
Use the _SEARCHFILENAMEONLY_ directive to resolve categories 1 and 2 above. Category 3 is a known limitation; in this case it is recommended that you adopt standard coding practices to avoid files with the same name and different content.

When you un-archive a sar file that contains relative or absolute include paths for the files in the project, you can add the _SEARCHFILENAMEONLY_ compiler directive to the unarchived project. This has the compiler remove the relative/absolute paths from the `include and search only for the file names.

This directive is specified with a set_option -hdl_define command added to an implementation within the project file as shown below:

```
set_option -hdl_define -set "__SEARCHFILENAMEONLY__"
```

The directive can also be added to the Compiler Directives field of the Verilog panel as shown below.



The compiler generates the following warning message whenever it extracts include files using this directive:

```
@W: | Macro _SEARCHFILENAMEONLY_ is set: fileName not found
attempting to search for base file name fileName
```

__ALLOWNESTEDBLOCKCOMMENTSTART__

Verilog/SystemVerilog comments can be included in RTL code as a

- One-line comment starting with the characters // and ending with a new line
- Block comment starting with /* and ending with */

However, nested block comments are not supported according to the LRM, so most tools generate a warning and ignore these comments in the RTL code. To match this behavior, the synthesis tools must also ignore various nested block comments, such as:

```
/*....../*....*/
(An incorrect pair)

/*...../*...../*.....*/
```

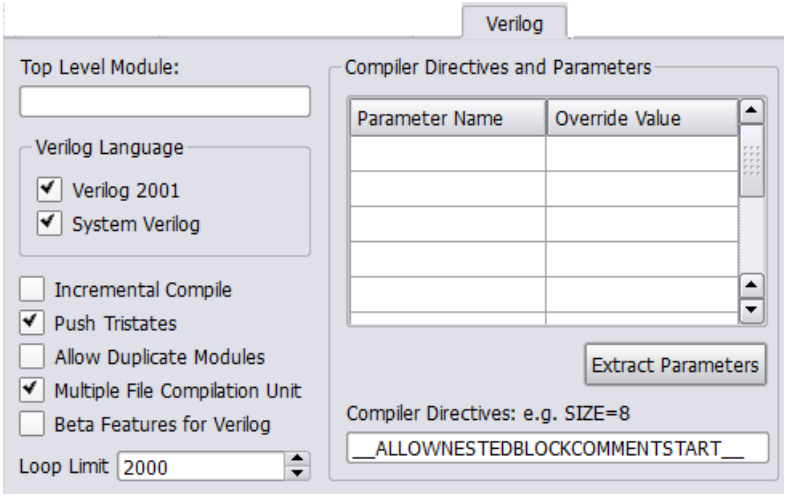

© 2020 Synopsys, Inc.
Synopsys Confidential Information

To do this, the compiler uses the __ALLOWNESTEDBLOCKCOMMENTSTART__ directive and parses the first /* until it encounters the associated pair */, then ignores all content between and including any number of these lines and the block comments.

You can specify the __ALLOWNESTEDBLOCKCOMMENTSTART__ directive with a set_option -hdl_define command added to the project file as shown below:

```
set_option -hdl_define -set __ALLOWNESTEDBLOCKCOMMENTSTART__
```

The directive can also be added to the Compiler Directives field of the Verilog panel.



__SYN_COMPATIBLE_INCLUDEPATH__

Specifies that the search path order for includes to be the same as the one used by the simulation tool (VCS), instead of the following default search order, which searches the current logical library of the file where the module is instantiated first, then the library path for the current logical library, and lastly other logical libraries.

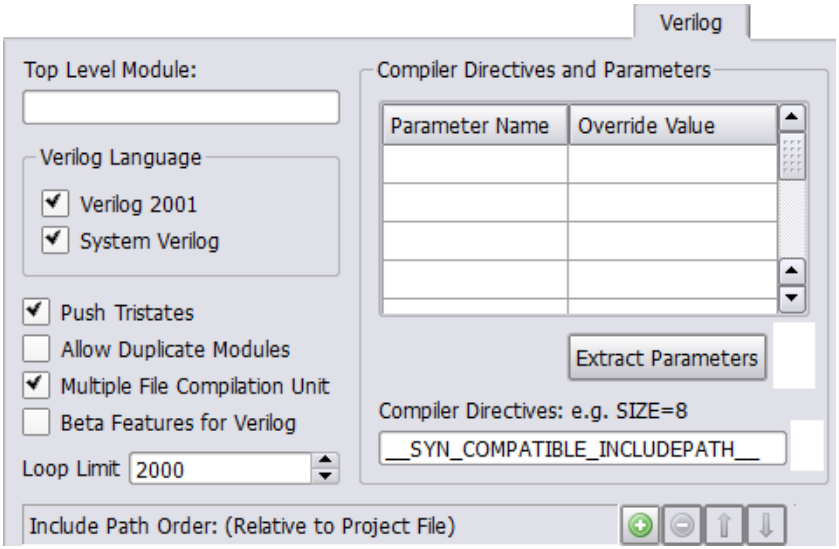
With __SYN_COMPATIBLE_INCLUDEPATH__, this is the search path order:

- Current working directory
- Include file directories, with first priority to RTL includes and then include paths

You can specify the __SYN_COMPATIBLE_INCLUDEPATH__ directive with a set_option -hdl_define command added to the project file as shown below:

```
set_option -hdl_define -set __SYN_COMPATIBLE_INCLUDEPATH__
```

The directive can also be added to the Compiler Directives field of the Verilog panel.



SYN_COMPATIBLE

Use the SYN_COMPATIBLE macro to ensure compatibility between different Synopsys tools. Some Synopsys tools, such as Design Compiler (DC), ignore dynamic initialization assignments, unlike the synthesis tool. In the following example, note the line logic a=b; DC leaves the output unconnected, because DC does not handle inline assignments. By contrast, the synthesis tool handles inline assignments, so input b drives the output q.

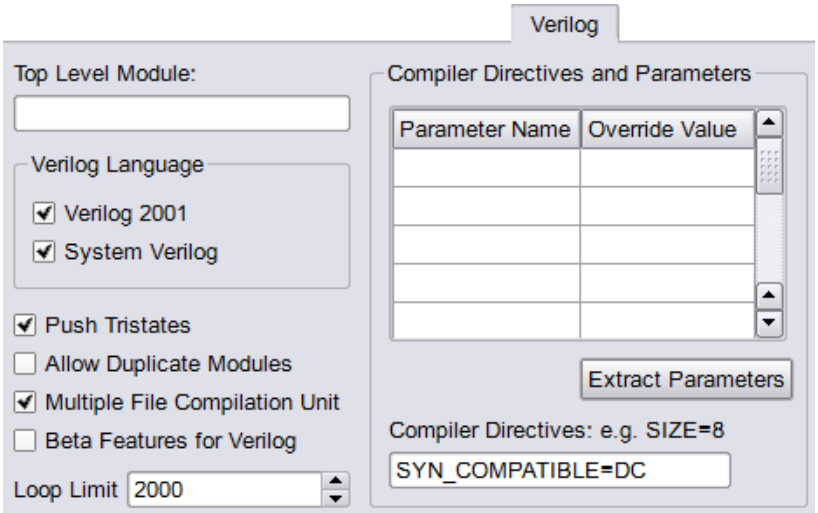
```
module test (b, q);
input b;
output q;
logic a = b;
assign q = a;
endmodule
```

The tool warns you of the difference in handling (warning message @W: CG879), and treats the assignment as a regular assign. If you are using modules from DC and need it to be compatible with the synthesis tool, there are two ways to match the behavior and ignore the non-constant initial values:

1. Specify a macro with a Tcl command.

   set_option -hdl_define -set SYN_COMPATIBLE=DC

2. Specify a macro through the GUI.
   - To specify it from the GUI, go to the Verilog tab of the Implementation Options dialog box.
   - In the Compiler Directives field, set SYN_COMPATIBLE=DC.



__SYN_STRICT_MODPORTS__

Use the __SYN_STRICT_MODPORTS__ macro to require that modports defined strictly access the associated interface ports specified for the instantiation. You might encounter the following error:

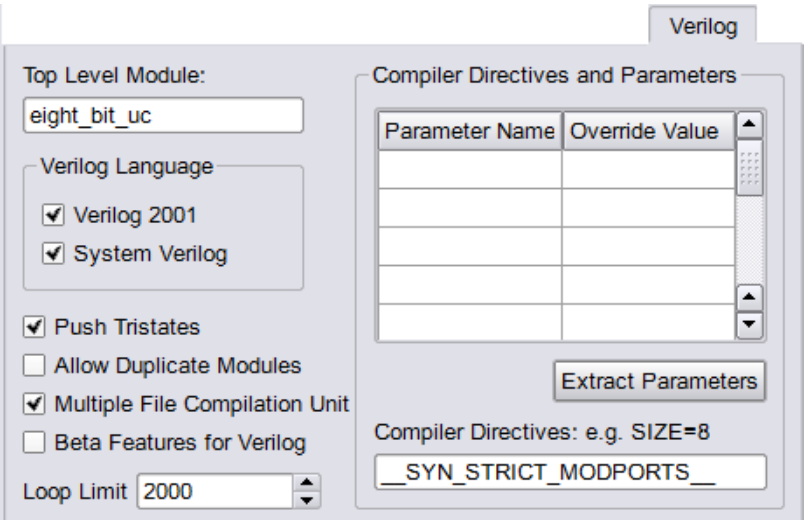
> @E: CS172 The number of ports in the instantiation does not match the number of ports in the module definition for instance *instanceName*

This can occur if you have specified a global interface instantiation for modports using the ".*" syntax to access the ports. Set the __SYN_STRICT_-MODPORTS__ macro to ensure mapping completes successfully.

1. To specify the macro with a Tcl command.

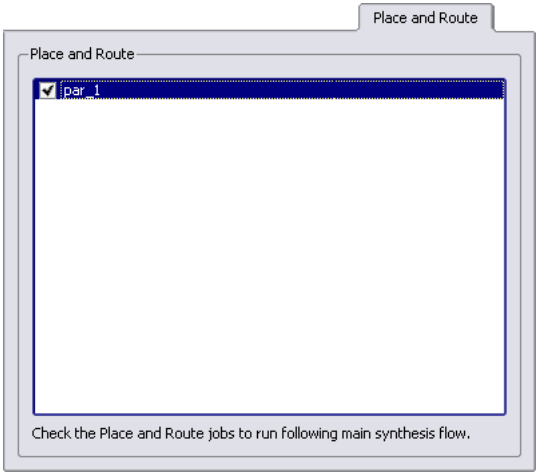
   set_option -hdl_define -set __SYN_STRICT_MODPORTS__

2. To specify the macro through the GUI.
   - To specify it from the GUI, go to the Verilog tab of the Implementation Options dialog box.
   - In the Compiler Directives field, set __SYN_STRICT_MODPORTS__.



Place and Route Panel

The Place and Route panel allows you to run selected place-and-route jobs after design synthesis. To create a place-and-route job, see *Add P&R Implementation Popup Menu Command*, on page 478 or *Options for Place & Route Jobs Popup Menu Command*, on page 480 for details.


Place and Route
Place and Route
par_1
Check the Place and Route jobs to run following main synthesis flow.

Run Menu

You use the Run menu to perform the following tasks:

- Compile a design, without mapping it.
- Synthesize (compile and map) or resynthesize a design.
- Check design syntax and synthesis code, and check source code errors.
- Check constraint syntax and how/if constraints are applied to the design.
- Run Tcl scripts.
- Run all implementations at once.
- Check the status of the current job.

The following table describes the Run menu commands.

Command	Description
Run	Synthesizes (compiles and maps) the top-level design. For the compile point flow, this command also synthesizes any compile points whose constraints, implementation options, or source code changed since the last synthesis run. You can view the result of design synthesis in the RTL and Technology views. Same as clicking the Run button in the Project view. Tcl equivalent: **project -run**
Resynthesize All	Resynthesizes (compiles and maps) the entire design, including the top level and *all compile points*, whether or not their constraints, implementation options, or source code changed since the last synthesis. If you do *not* want to force a *recompilation of all compile points*, then use Run->Run instead. Tcl equivalent: **project -run synthesis -clean**
Compile Only	Compiles the design into technology-independent high-level structures. You can view the result in the RTL view. Tcl equivalent: **project -run compile**

Command	Description
Write Output Netlist Only	Generates an output netlist after synthesis has been run. This command generates the netlists you specify on the Implementation Results tab of the Implementation Options dialog box. You can also use this command in an incremental timing analysis flow. See Generating Custom Timing Reports with STA, on page 366 for details. Tcl equivalent: **project -run write_netlist**
FSM Explorer	Analyzes finite state machines contained in a design, and selects the optimum encoding style. This menu command is not available in some views. Tcl equivalent: **project -run fsm_explorer**
Syntax Check	Runs a syntax check on design code. The status bar at the bottom of the window displays any error messages. If the active window shows an HDL file, then the command checks only that file; otherwise, it checks all project source code files. Tcl equivalent: **project -run syntax_check**
Synthesis Check	Runs a synthesis check on your design code. This includes a syntax check and a check to see if the synthesis tool could map the design to the hardware. No optimizations are carried out. The status bar at the bottom of the window displays any error messages. If the active window shows an HDL file, then the command checks only that file; otherwise, it checks all project source code files. Tcl equivalent: **project -run synthesis_check**
Constraint Check	Checks the syntax and applicability of the timing constraints in the .fdc/.sdc file for your project and generates a report (*projectName*_cck.rpt). The report contains information on the constraints that can be applied, cannot be applied because objects do not exist, and wildcard expansion on the constraints. See *Constraint Checking Report* , on page 173. Tcl equivalent: **project -run constraint_check**
Arrange VHDL files	Reorders the VHDL source files for synthesis. Tcl equivalent: **project -run hdl_info_gen fileorder**

Command	Description
Launch Identify Instrumentor	Launches the Identify Instrumentor within the FPGA synthesis tools; See *Working with the Identify Tools* , on page 556 to launch the debugger tools.
Launch Identify Debugger	Launches the Identify debugger tool. For more information, see: *Working with the Identify Tools* , on page 556 of the *User Guide*. To launch the Identify debugger in batch mode, use the set_option -identify_debug_mode 1 Tcl command to create an Identify implementation.
Launch SYNCore	Opens the Synopsys FPGA IP Core Wizard. This tool helps you build IP blocks such as memory or FIFO models for your design. See the *Launch SYNCore Command* , on page 389 for details.
Configure and Launch VCS Simulator	Allows you to configure and launch the VCS simulator. See *Configure and Launch VCS Simulator Command* , on page 389.
Run Tcl Script	Displays the Open dialog box, where you choose a Tcl script to run. See *Run Tcl Script Command* , on page 384.
Run Implementations Setup	Runs all selected implementations for one project at the same time. See *Run Implementations Setup Command* , on page 385. Tcl equivalent: **run -impl "***implementation1 implementation2...***" -parallel**
Job Status	During compilation, tells you the name of the current job, and gives you the runtime and directory location of your design. This option is enabled during synthesis. See *Job Status Command* , on page 387. Clicking in the status area of the Project view is a shortcut for this command.

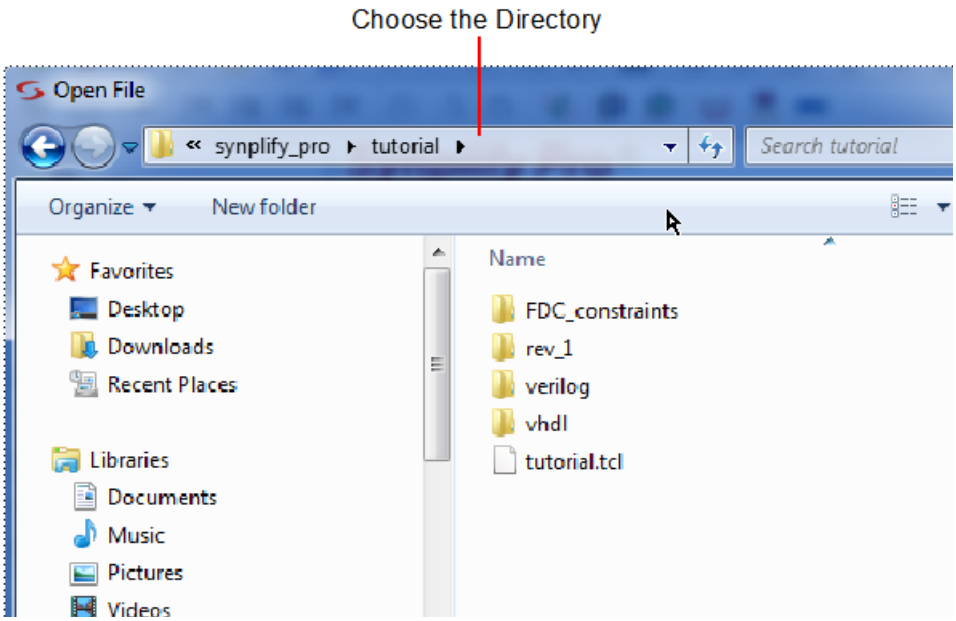
© 2020 Synopsys, Inc.
Synopsys Confidential Information

Command	Description
Next Error/Warning	Shows the next error or warning in your source code file.
Previous Error/Warning	Shows the previous error or warning in your source code file.
Log File Message Filter	Allows messages in the current session to be elevated in severity (for example, promoted to an error from a warning), lowered in severity (for example, demoting a warning to a note), or suppressed from the log file after the next run through the Log File Filter dialog box. For more information, see *Log File Message Controls* , on page 213.

Run Tcl Script Command

Select Run->Run Tcl Script to display the Open dialog box, where you specify the Tcl script file to execute. The File name area is filled automatically with the wildcard string "*.tcl", corresponding to Tcl files.

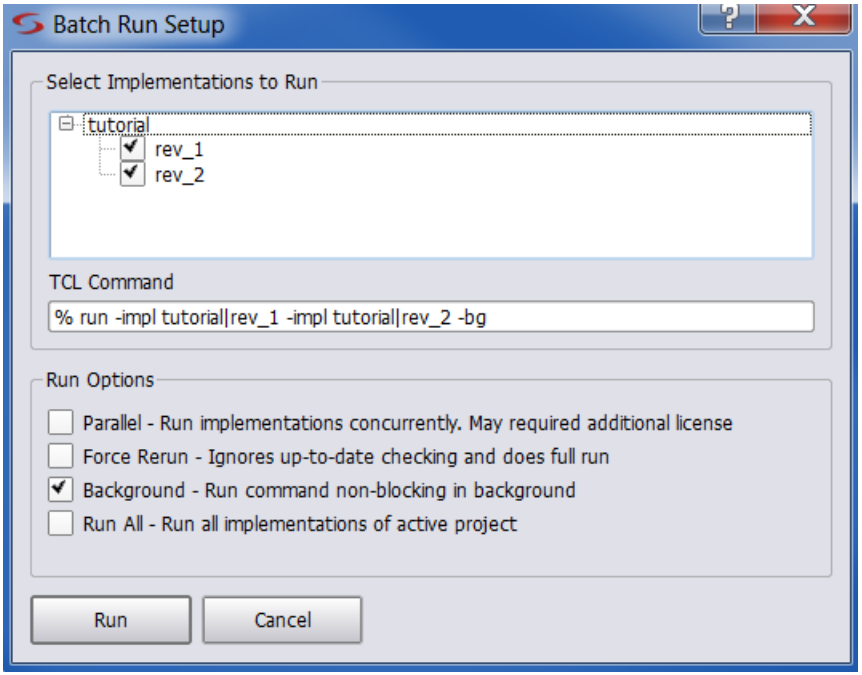
This dialog box is the same as that displayed with File->Open, except that no Open as read-only check box is present. See *Open Project Command*, on page 311, for an explanation of the features in the Open dialog box.



Run Implementations Setup Command

Select Run->Run Implementations Setup to run selected implementations of a Project file in batch mode. To use the Batch Run Setup dialog box, check one or more implementations from the list displayed and click the Run button.

© 2020 Synopsys, Inc.



You can also choose to run the selected implementations with one or more of the following options:

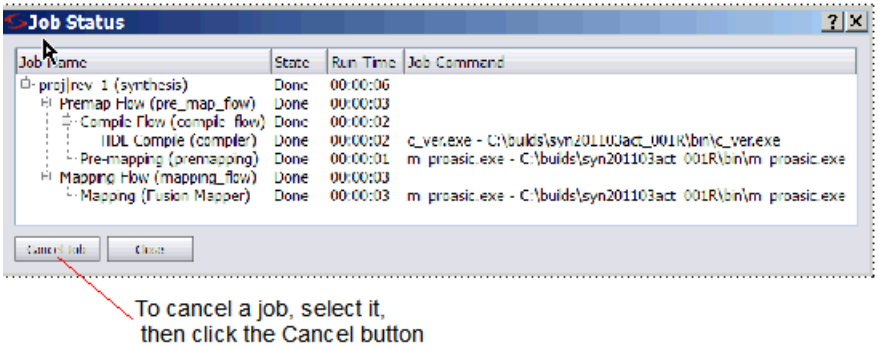
Command	Description
Parallel	Runs specified implementations concurrently. This may require additional licenses. Tcl equivalent: **run -impl** *implName* **-parallel**
Force Rerun	Runs specified implementations, while ignoring up-to-date checking. This option clears all previous results and forces a complete rerun. Tcl equivalent: **run -impl** *implName* **-clean**
Background	Runs specified implementations in non-blocking background mode. Tcl equivalent: **run -impl** *implName* **-bg**
Run All	Runs all implementations of the active project. Tcl equivalent: **run -all**

© 2020 Synopsys, Inc.

Job Status Command

Select Run->Job Status to monitor the synthesis jobs that are running, their run times, and their associated commands. This information appears in the Job Status dialog box. This dialog box is also displayed when you click in the status area of the Project view (see The Project View, on page 22).

You can cancel a displayed job by selecting it in the dialog box and clicking Cancel Job.



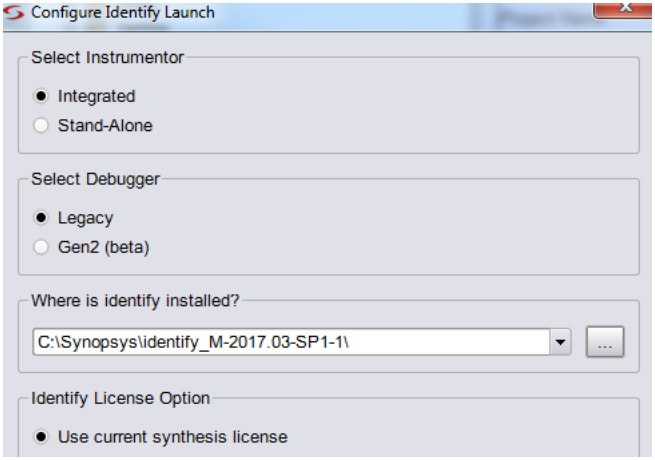
Identify Instrumentor Command

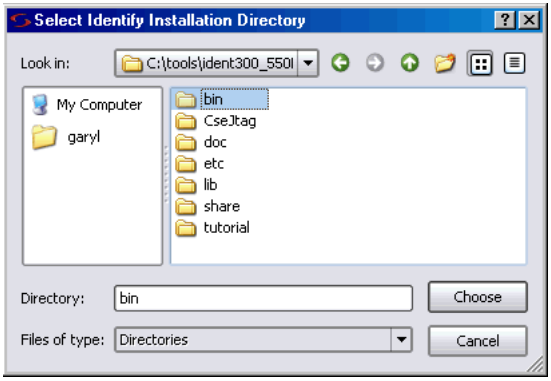
The Identify Instrumentor command lets you start the integrated or stand-alone Identify instrumentor from within the synthesis interface. Before you can use this command, you must define an Identify implementation in the project view. For a description of the work flow using the Identify instrumentor, see *Working with the Identify Tools*, on page 556 in the *User Guide*.

Configure Identify Launch Dialog Box

Select Options->Configure Identify Launch to display this dialog box or which is automatically displayed when the location of the Identify executable has not been previously defined.

© 2020 Synopsys, Inc.
Synopsys Confidential Information



Command	Description
Select Instrumentor	Click the radio button to select which version of the instrumentor to use. You can choose either the integrated or stand-alone Identify Instrumentor.
Locate Identify Installation (for the Identify Debugger)	A pointer to the Identify install directory. Use the (...) button to navigate to the directory location. 
Identify License Option	Radio buttons to select the Identify license option. Select Use current synthesis license when only a single TSL license is available; select Use separate Identify Instrumentor license when multiple licenses are available. With a single TSL license, you are prohibited from compiling or mapping in the synthesis tool while the Identify instrumentor is open.

Launch Identify Debugger Command

The Launch Identify Debugger command launches a stand-alone version the Identify Debugger software from the synthesis interface. Before you can use this command, you must have an active Identify implementation and an instrumented design. For a description of the work flow using the Identify/Identify RTL Debugger software, see *Working with the Identify Tools*, on page 556 in the *User Guide*.

Launch SYNCore Command

The SYNCore wizard helps you build IP cores. The wizard can compile RAM and ROM memories including a byte-enable RAM, a FIFO, an adder/subtractor, and a counter. The resulting Verilog models can be synthesized and simulated. For details about using the wizard to build these models, see the following topics:

- *SYNCore FIFO Compiler*, on page 250
- *SYNCore RAM Compiler*, on page 281
- *SYNCore Byte-Enable RAM Compiler*, on page 303
- *SYNCore ROM Compiler*, on page 319
- *SYNCore Adder/Subtractor Compiler*, on page 334
- *SYNCore Counter Compiler*, on page 358

Configure and Launch VCS Simulator Command

The Configure and Launch VCS Simulator command enables you to launch VCS simulation from within the Synopsys FPGA synthesis tools. Additionally, configuration information, such as libraries and options can be specified on the Run VCS Simulator dialog box before running VCS simulation. You can launch this simulation tool from the synthesis tools on Linux platforms only.

For a step-by-step procedure on setting up and launching this tool, see Simulating with the VCS Tool, on page 565 in the *User Guide*.

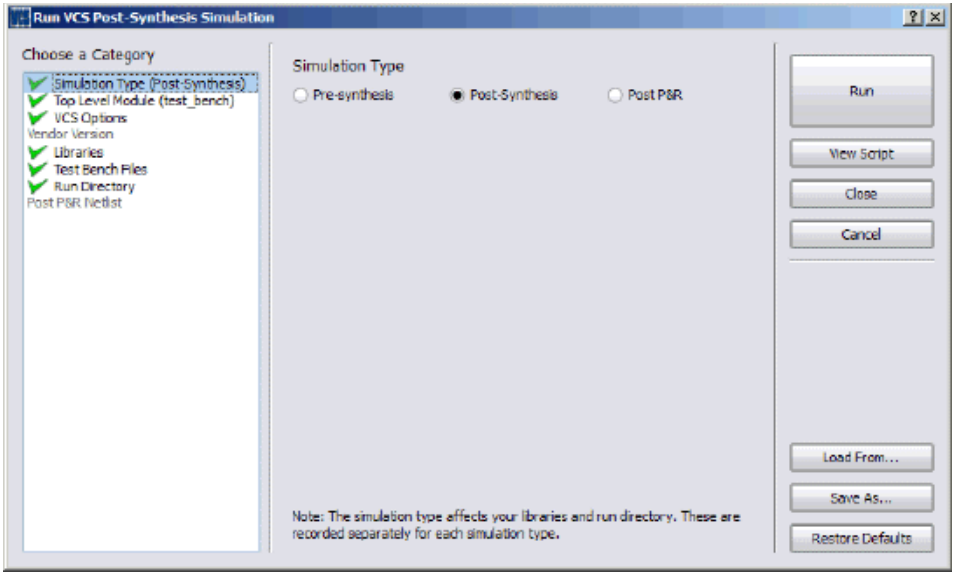
The Run VCS *SimulationType* Simulation dialog box contains unique pages for specific tasks, such as specifying simulation type, VCS options, and libraries or test bench files. From this dialog box:

© 2020 Synopsys, Inc.
Synopsys Confidential Information

- Choose a category, which simplifies the data input for each task.
- A task marked with (✔) means that data has automatically been filled in; however, an (⊗) requires that data must be filled in.
- You are prompted to save, after canceling changes made in the dialog box.

Simulation Type

The following dialog box displays the Simulation Type task.



The Run VCS Simulator dialog box contains the following options:

Command	Description
Choose a Category Simulation Type	Select Simulation Type and choose the type of simulation to run: • Pre-synthesis - RTL simulation • Post-synthesis - Post-synthesis netlist simulation • Post-P&R - Post-P&R netlist simulation See *Simulation Type* , on page 390 to view the dialog box.
Choose a Category Top Level Module	Select Top Level Module and specify the top-level VCS module or modules for simulation. You can use any combination of the semi-colon (;), comma (,), or a space to separate multiple top-level modules. See *Top Level Module* , on page 393 to view the dialog box.
Choose a Category VCS Options	Select VCS Options and specify options for each VCS step: • Verilog compiler - VLOGAN command options for compiling and analyzing Verilog, such as the -q option. • VHDL compiler - VHDLAN options for compiling and analyzing VHDL. • Elaboration - VCS command options. The default setting is -debug_all. • Simulation - SIMV command options. The default setting is -gui. The default settings use the FPGA version of VCS and open the VCS GUI for the debugger (DBE) and the waveform viewer. See *VCS Options* , on page 394 to view the dialog box.
Choose a Category Libraries	Select Libraries and specify library files typically used for Post-synthesis or Post-P&R simulation. These library files are automatically populated in the display window. You can choose to: • Add a library • Edit the selected library • Remove the selected library See *Libraries* , on page 395 and *Changing Library and Test Bench Files* , on page 397 for more information.

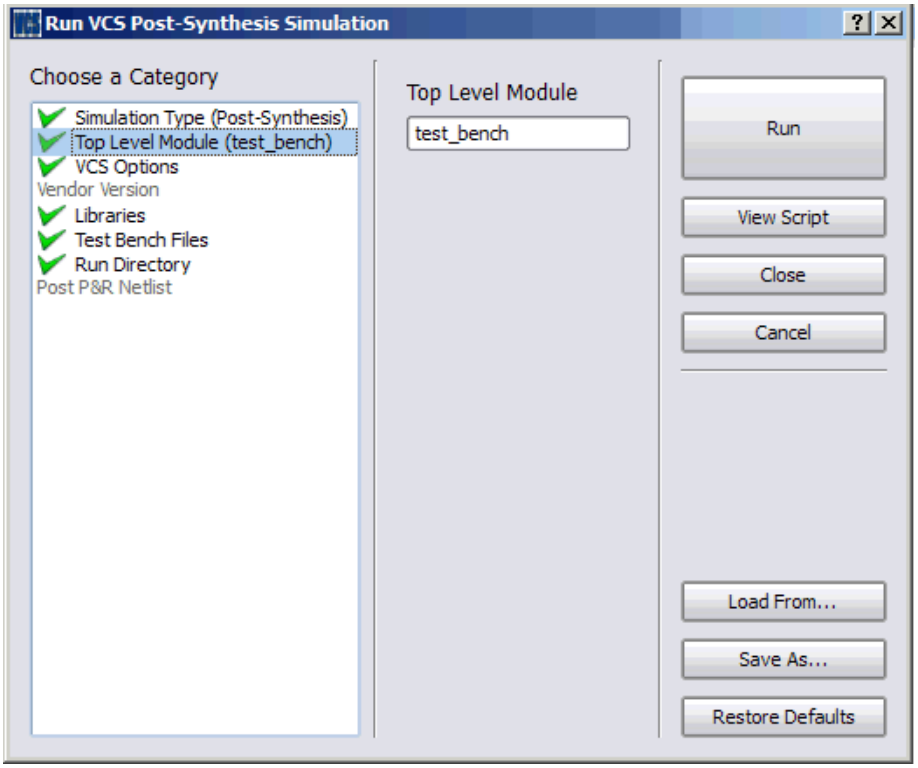
© 2020 Synopsys, Inc.
Synopsys Confidential Information

Command	Description
Choose a Category Test Bench Files	Select Test Bench Files and specify the test bench files typically used for Post-synthesis or Post-P&R simulation. These test bench files are automatically populated in the display window. You can choose to: • Add a test bench file • Edit the selected test bench file • Remove the selected test bench file See *Test Bench Files* , on page 395 and *Changing Library and Test Bench Files* , on page 397 for more information.
Choose a Category Run Directory	Select Run Directory and specify the results directory to run the VCS simulation. See *Run Directory* , on page 396 to view the dialog box.
Choose a Category Post P&R Netlist	Select Post P&R Netlist and specify the post place-and-route netlist to run the VCS simulation. See *Post P&R Netlist* , on page 396 to view the dialog box.
Run	Runs VCS simulation.
View Script	View the script file with the specified VCS commands and options before generating it. For an example, see *VCS Script File* , on page 398.
Load From	Use this option to load an existing VCS script.
Save As	Generates the VCS script. The tool generates the XML script in the directory specified.
Restore Defaults	Restores all the default VCS settings.

© 2020 Synopsys, Inc.

Top Level Module

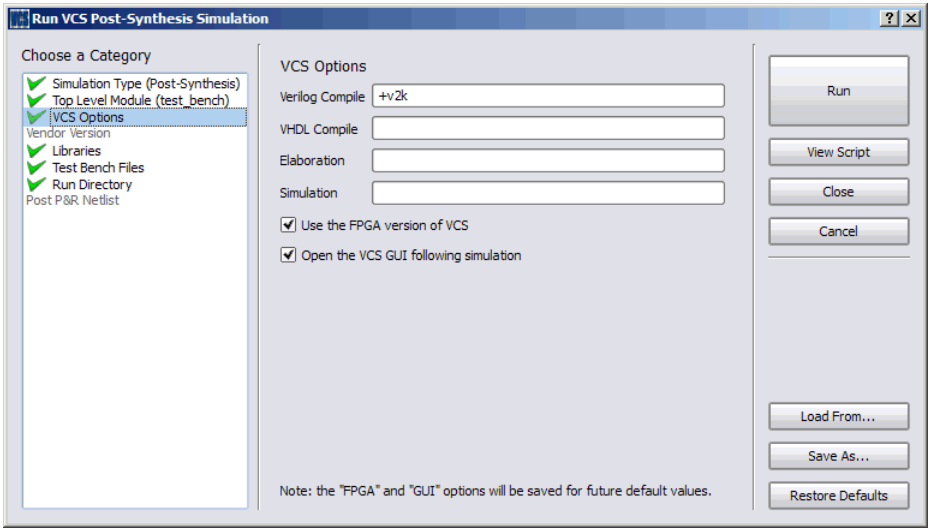
The following dialog box displays the Top Level Module task.



© 2020 Synopsys, Inc.

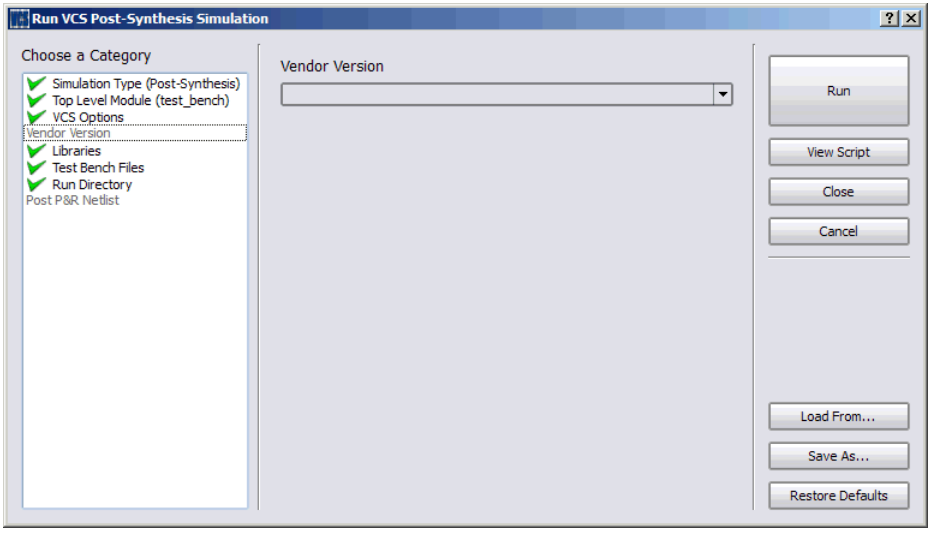
VCS Options

The following dialog box displays the VCS Options task.



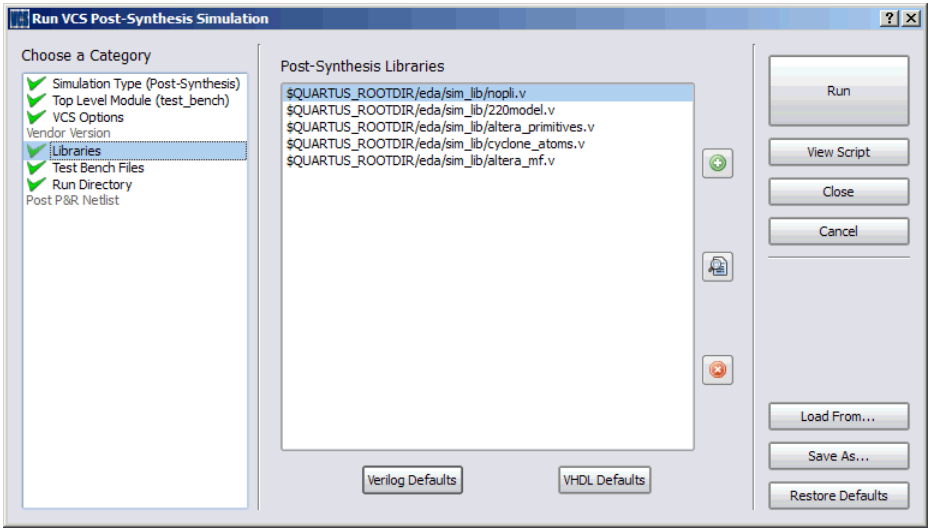
Vendor Version

The following dialog box displays the Vendor Versions task.



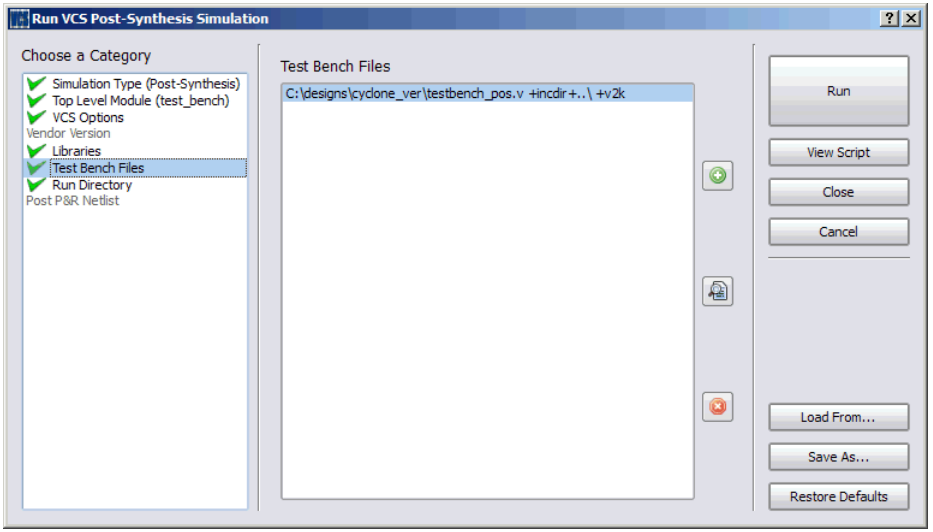
Libraries

The following dialog box displays the Libraries task.



Test Bench Files

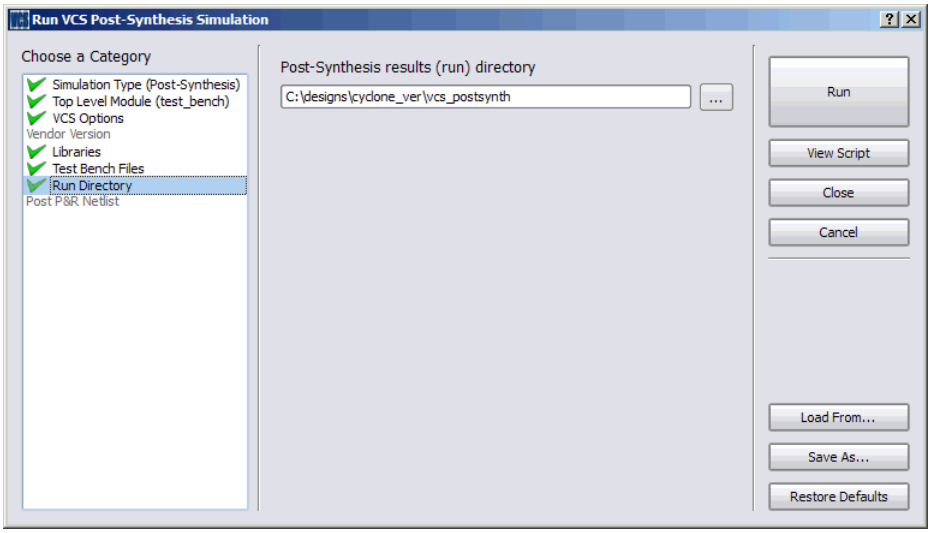
The following dialog box displays the Test Bench Files task.



© 2020 Synopsys, Inc.

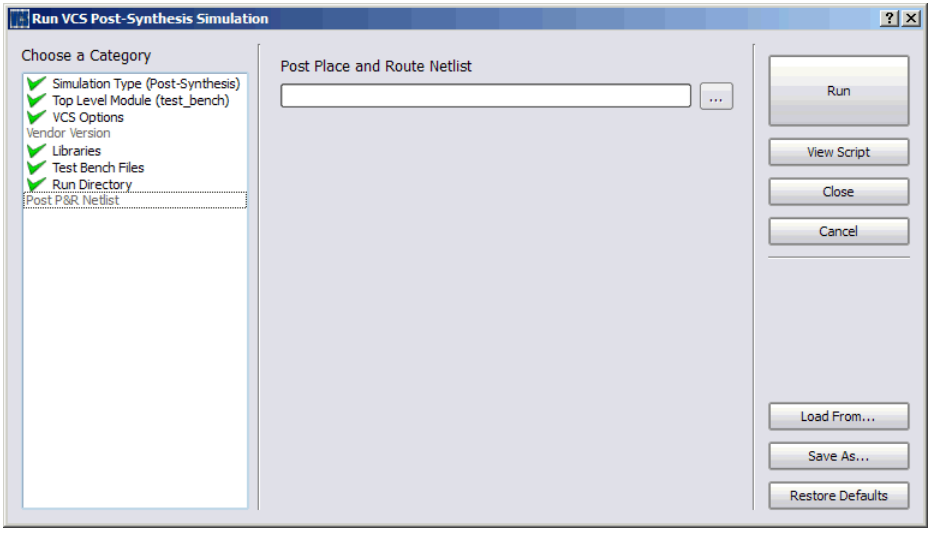
Run Directory

The following dialog box displays the Run Directory task.



Post P&R Netlist

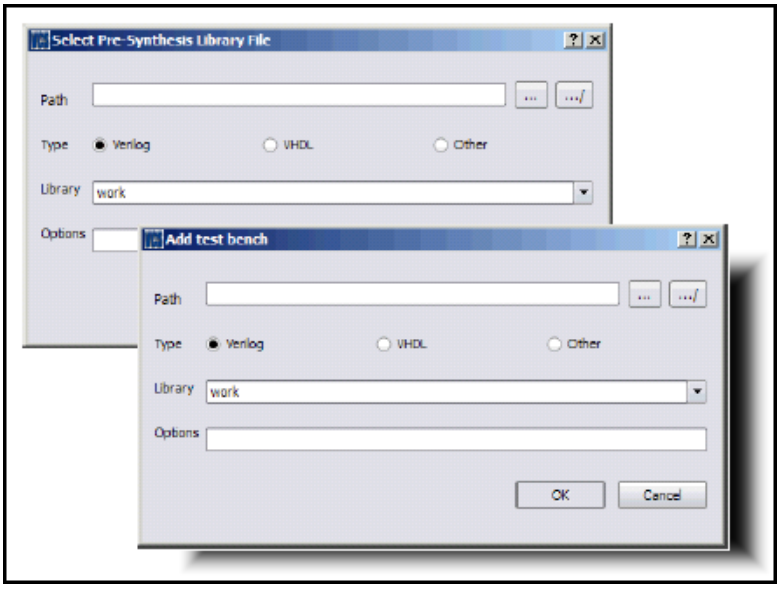
The following dialog box displays the Post P&R Netlist task.



© 2020 Synopsys, Inc.

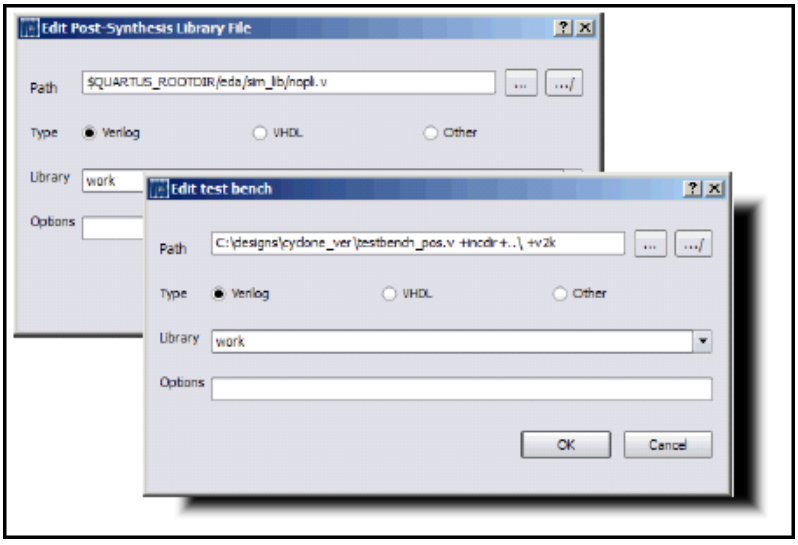
Changing Library and Test Bench Files

You can add Post-synthesis or Post place-and-route library files and test bench files before you launch the VCS simulator. For example, specify options on the following dialog box.



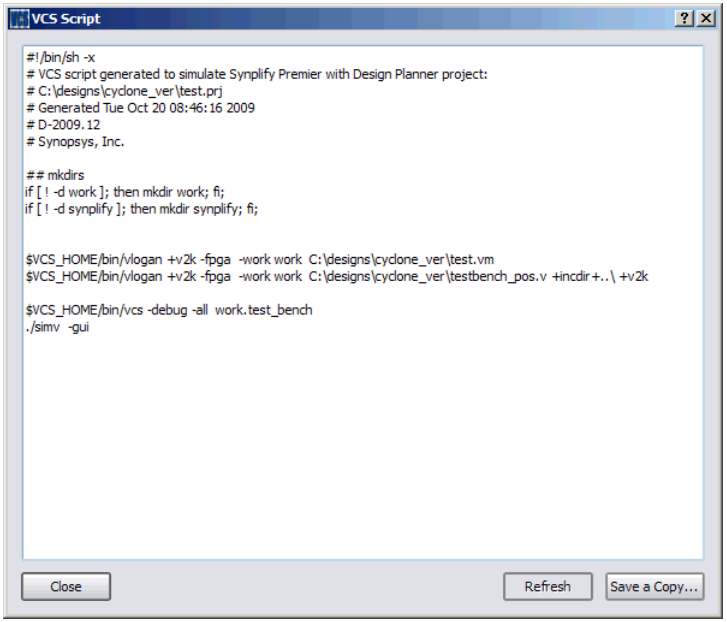
You can also edit library files and test bench files before you launch the VCS simulator. For example: specify options on the following dialog box.

© 2020 Synopsys, Inc.



VCS Script File

When you select the VCS Script button on the Run VCS Simulator dialog box, you can view the VCS script generated by the synthesis software for this VCS run. You can also save this VCS script to a file by clicking on Save a Copy.



© 2020 Synopsys, Inc.

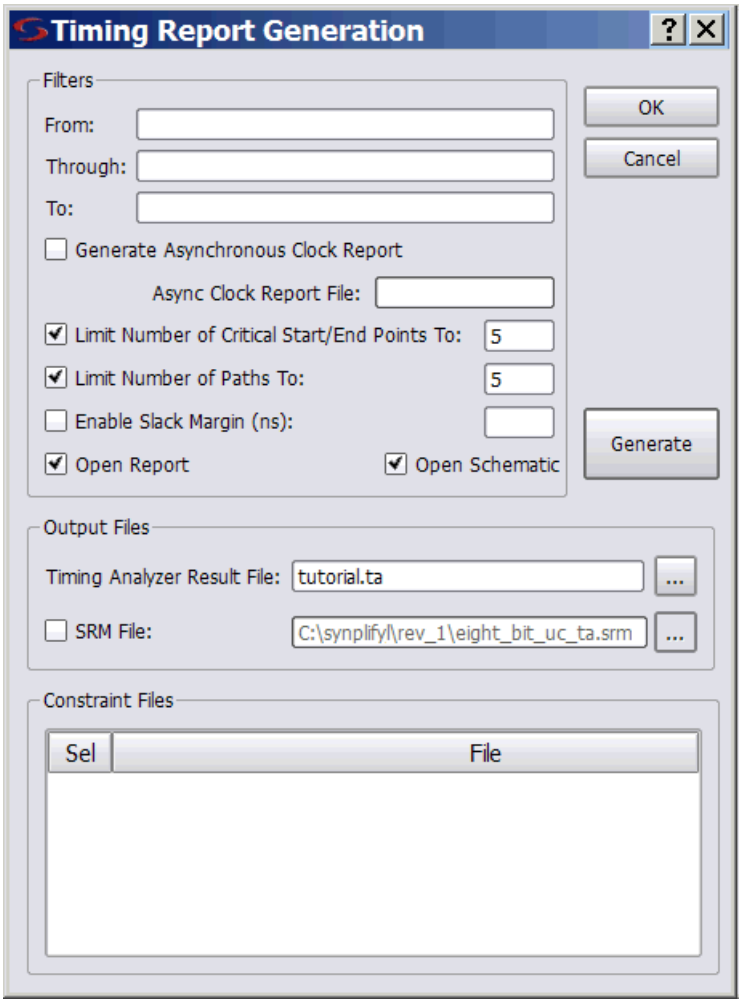
Analysis Menu

When you synthesize a design, a default timing report is automatically written to the log file (*projectName*.srr), located in the results directory. This report provides a clock summary, I/O timing summary, and detailed critical path information for the design. However, you can also generate a custom timing report that provides more information than the default report (specific paths or more than five paths) or one that provides timing based on additional analysis constraint files without rerunning synthesis.

Command	Description
Timing Analyst	Displays the Timing Report Generation dialog box to specify parameters for a stand-alone customized report. See *Timing Report Generation Parameters* , on page 401 for information on setting these options, and *Analyzing Timing in Schematic Views* , on page 358 in the *User Guide* for more information. If you click OK in the dialog box, the specified parameters are saved to a file. To run the report, click Generate. The report is created using your specified parameters.
Generate Timing	Generates and displays a report using the timing option parameters specified above. See the following: • Generating Custom Timing Reports with STA, on page 366 for specifics on how to run this report. • *Timing Report Generation Parameters* , on page 401 for information on setting parameters for the report. This includes information on filtering and options for running backannotation data and power consumption reports.

Timing Report Generation Parameters

You can use the Analysis->Timing Analyst command to specify parameters for a stand-alone timing report. See *Timing Reports*, on page 162 for information on the file contents.



© 2020 Synopsys, Inc.

The following table provides brief descriptions of the parameters for running a stand-alone timing report.

Timing Report Option	Description
From or To	Specifies the starting (From) or ending (To) point of the path for one or more objects. It must be a timing start point (From) or end (To) point for each object. Use this option in combination with the others in the Filters section of the dialog box. See *Combining Path Filters for the Timing Analyzer* , on page 405 for examples of using filters. Tcl equivalent: **set_option -reporting_filter "-from {*object1*} -to {*object2*}"**
Through	Reports all paths through the specified point or list of objects. See for more information on using this filter. Use this option in combination with the others in the Filters section of the dialog box. See the following for additional information: • Timing Analyzer Through Points, on page 404 • Combining Path Filters for the Timing Analyzer, on page 405. Tcl equivalent: **set_option -reporting_filter "-from {*object1*} -to {*object2*} -through {*object3*}"**
Generate Asynchronous Clock Report	Generates a report for paths that cross between clock groups. Generally paths in different clock groups are automatically handled as false paths. This option provides a file that contains information on each of the paths and can be viewed in a spreadsheet. This file is in the results directory (*projectName*_async_clk.rpt.csv). For details on the report, see *Asynchronous Clock Report* , on page 170. Tcl equivalent: **set_option -reporting_async_clock 0|1**
Limit Number of Critical Start/End Points to	Specifies the maximum number of start/end paths to display for critical paths in the design. The default is 5. Use this option in combination with the others in the Filters section of the dialog box. Tcl equivalent: **set_option -num_startend_points** *numberOfPaths*
Limit Number of Paths to	Specifies the maximum number of paths to report. The default is 5. Use this option in combination with the others in the Filters section of the dialog box. Tcl equivalent: **set_option -reporting_number_paths** *numberOfPaths*

Timing Report Option	Description
Enable Slack Margin (ns)	Limits the report to paths within the specified distance of the critical path. Use this option in combination with the others in the Filters section of the dialog box. Tcl equivalent: **set_option -reporting_margin** *slackValue*
Open Report	When enabled, clicking the Generate button opens the Text Editor on the generated custom timing report specified in the timing report file (.ta).
Open Schematic	When enabled, clicking the Generate button opens a Technology view showing the netlist specified in the timing report netlist file (.srm). Tcl equivalent: **set_option -reporting_output_srm 0|1**
Output Files	Displays the name of the generated report: • Async Clock Report File contains the spreadsheet data for the asynchronous clock report. This file is not automatically opened when report generation is complete. You can locate this file in the results directory. Default name is *projectName*_async_clk.rpt.csv (name cannot be changed). Tcl equivalent: **set_option -reporting_async_clock 0|1** • Timing Analyst Results File is the standard timing report file, located in the Implementation Results directory. The file is also listed in the Project view. Default filename is *projectName*.ta. Tcl equivalent: **set_option -reporting_filename** *filename*.**ta** • SRM File updates the Technology view so that you can display the results of the timing updates in the HDL and Physical Analyst tools. The file is also listed in the Project view. Tcl equivalent: **set_option -reporting_netlist** *filename* For more details on any of these reports, see *Timing Reports* , on page 162.
Constraint Files	Enables analysis design constraint files (.adc) to be used for stand-alone timing analysis only. See *Input Files* , on page 144 for information on this file.
Generate	Clicking this button generates the specified timing report file and timing view netlist file (.srm) if requested, saves the current dialog box entries for subsequent use, then closes the dialog box.

Synopsys Confidential Information

Timing Analyzer Through Points

You can specify through points for nets (n:), hierarchical ports (t:), or instantiated cell pins (t:). You can specify the through points in two ways:

OR list — Enter the points as a space-separated list. The points are treated as an OR list and paths are reported if they crosses any of the points in the list. For example, when you type the following, the tool reports paths that pass through points b or c:

```
{n:b n:c}
```

See *Filtering Points: OR List of Through Points* , on page 404.

AND list — Enter the points in a product of sums (POS) format. The tool treats them as an AND list, and only reports the path if it passes through all the points in the list. The POS format for the timing report is the same as for timing constraints. The POS format is as follows:

```
{n:b n:c},{n:d n:e}
```

This constraint translates as follows:

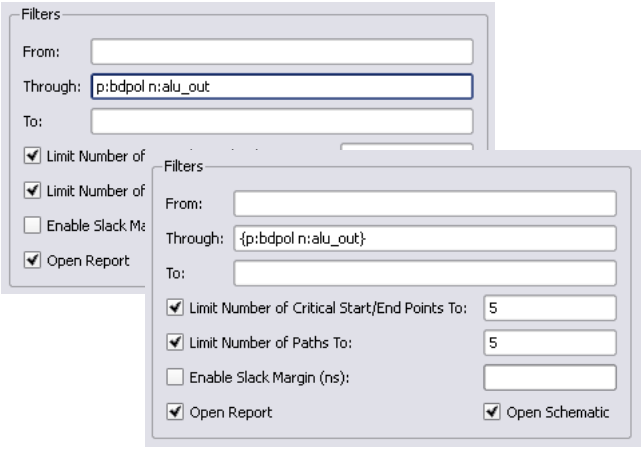
```
b AND d
OR b AND e
OR c AND d
OR c AND e
```

See *Filtering Points: AND List of Through Points* , on page 405.

See *Defining From/To/Through Points for Timing Exceptions*, on page 139 in the *User Guide* for more information about specifying through points.

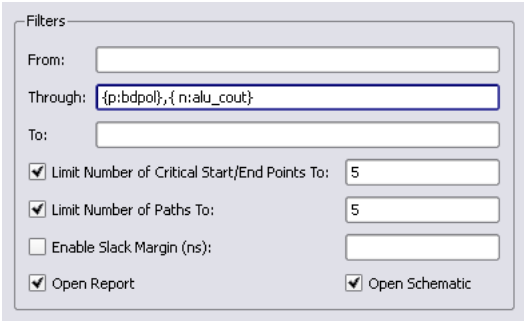
Filtering Points: OR List of Through Points

This example reports the five worst paths through port bdpol or net aluout. You can enter the through points as a space-separated list (enclosing the list in braces is optional.)



Filtering Points: AND List of Through Points

This example reports the five worst paths passing through port bdpol and net aluout. Enclose each list in braces {} and separate the lists with a comma.



Combining Path Filters for the Timing Analyzer

This section describes how to use a combination of path filters to specify what you need and how to specify start and end points for path filtering.

Number and Slack Path Filters

The Limit Number of Paths To option specifies the maximum number of paths to report and the Enable Slack Margin option limits the report to output only paths that have a slack value that is within the specified value. When you use these

© 2020 Synopsys, Inc.
Synopsys Confidential Information

two options together, the tighter constraint applies, so that the actual number of paths reported is the minimum of the option with the smallest value. For example, if you set the number of paths to report to 10 and the slack margin for 1 ns, if the design has only five paths within 1 ns of critical, then only five paths are reported (not the 10 worst paths). But if, for example, the design has 15 paths within a 1 ns of critical, only the first 10 are reported.

From/To/Through Filters

You can specify the from/to points for a path. You can also specify just a from point or just a to point. The from and to points are one or more hierarchical names that specify a port, register, pin on a register, or clock as object (clock alias). Ports and instances can have the same names, so prefix the name with p: for top-level port, i: for instance, or t: for hierarchical port or instance pin. However, the c: prefix for clocks is required for paths to be reported.

The timing analyst searches for the from/to objects in the following order: clock, port, bit port, cell (instance), net, and pin. Always use the prefix qualifier to ensure that all expected paths are reported. Remember that the timing analyst stops at the first occurrence of an object match. For buses, all possible paths from the specified start to end points are considered.

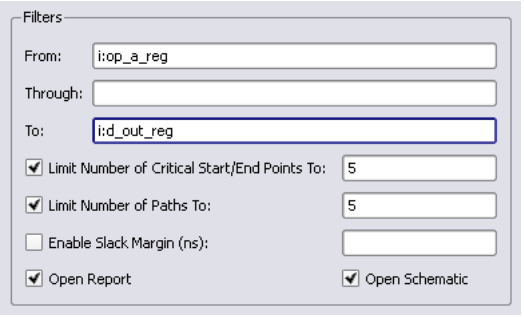
You can specify through points for nets, cell pins, or hierarchical ports.

You can simply type in from/to or through points. You can also cut-and-paste or drag-and-drop valid objects from the RTL or Technology views into the appropriate fields on the Timing Report Generation dialog box. Timing analysis requires that constraints use the Tech View name space. Therefore, it is recommended that you cut-and-paste or drag-and-drop objects from the Technology view rather than the RTL view.

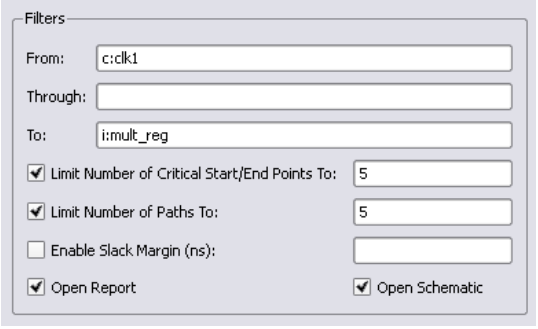
© 2020 Synopsys, Inc.

This following examples show how to specify start, end, or through point combinations for path filtering.

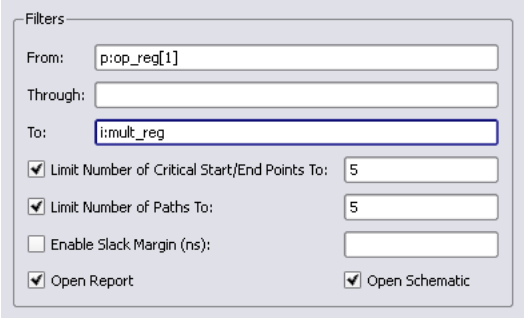
Filtering Points: Single Register to Single Register



Filtering Points: Clock Object to Single Register

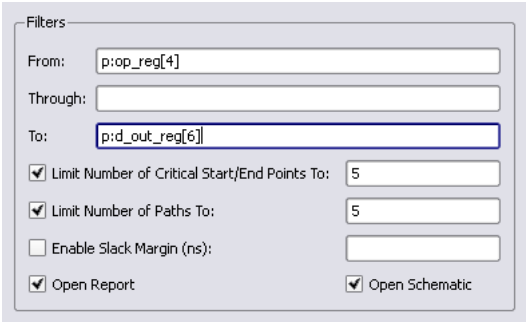


Filtering Points: Single Bit of a Bus to Single Register

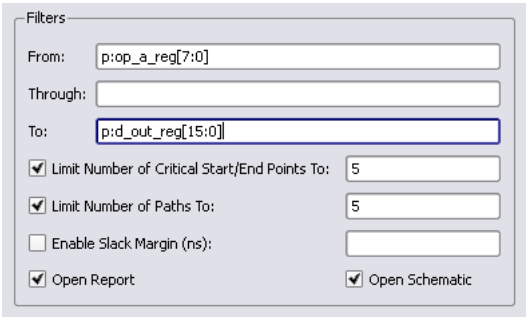


Synopsys Confidential Information

Filtering Points: Single Bit of a Bus to Single Bit of a Bus

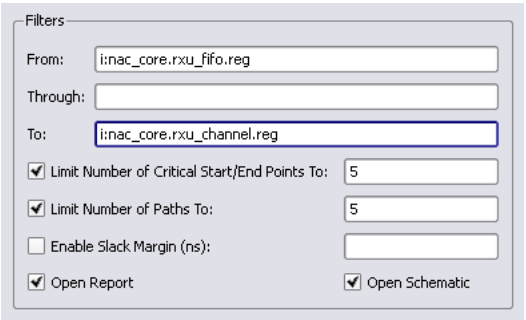


Filtering Points: Multiple Bits of a Bus to Multiple Bits of a Bus

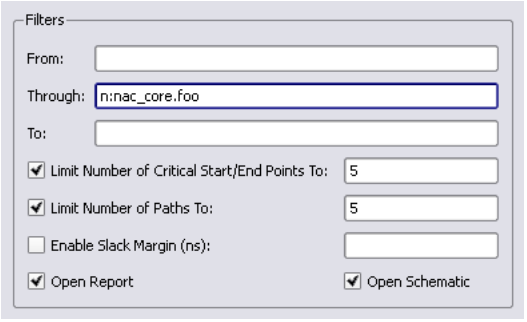


Filtering Points: With Hierarchy

This example reports the five worst paths for the net foo:

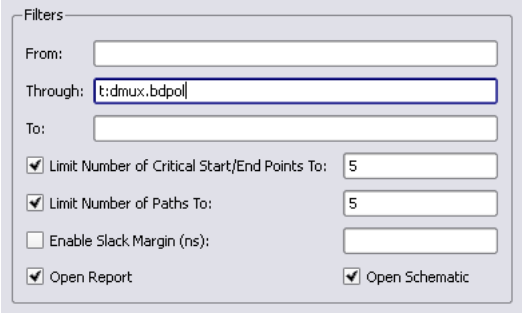


Filtering Points: Through Point for a Net



Filtering Points: Through Point for a Hierarchical Port

This example reports the five worst paths for the hierarchical port bdpol:



Examples Using Wildcards

You can use the question mark (?) or asterisk (*) wildcard characters for object searching and name substitution. These characters work the same way in the synthesis tool environment as in the Linux environment.

© 2020 Synopsys, Inc.

The ? Wildcard

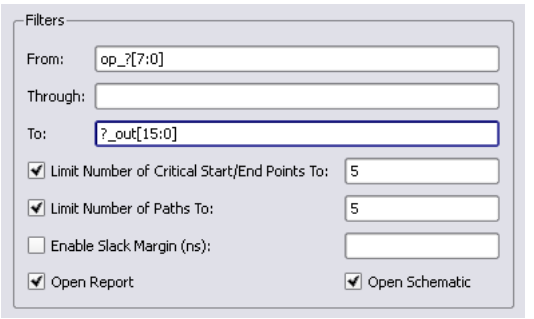
The ? matches single characters. If a design has buses op_a[7:0], op_b[7:0], and op_c[7:0], and you want to filter the paths starting at each of these buses, specify the start points as op_?[7:0]. See *Example: ? Wildcard in the Name*, on page 410 for another example.

The * Wildcard

The * matches a string of characters. In a design with buses op_a2[7:0], op_b2[7:0], and op_c2[7:0], where you want to filter the paths starting at each of these objects, specify the start points as op_*[*]. The report shows all paths beginning at each of these buses and for all of the bits of each bus. See *Example: * Wildcard in the Name (With Hierarchy)*, on page 411 and *Example: * Wildcard in the Bus Index*, on page 411 for more examples.

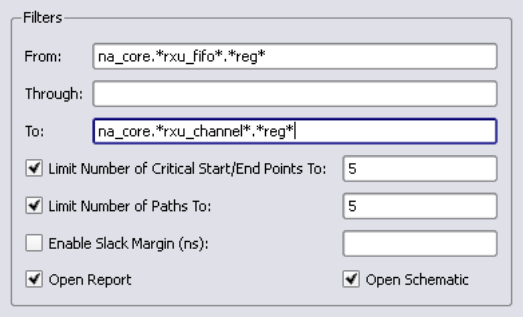
Example: ? Wildcard in the Name

The ? is not supported in bus indices.



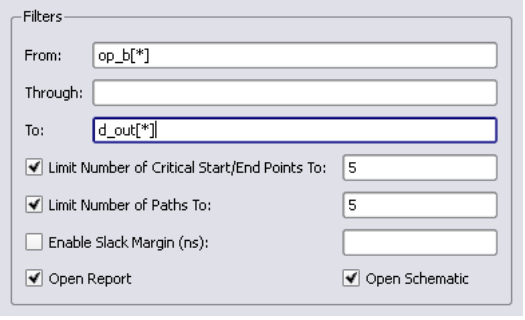
Example: * Wildcard in the Name (With Hierarchy)

This example reports the five worst paths, starting at block rxu_fifo and ending at block rxu_channel within module nac_core. Each register in the design has the characters reg in the name.



Example: * Wildcard in the Bus Index

This example reports the five worst paths, starting at op_b, and ending at d_out, taking into account all bits on these buses.



© 2020 Synopsys, Inc.
Synopsys Confidential Information

HDL Analyst Menu

In the Project View, the HDL Analyst menu contains commands that provide project analysis in the following views:

- RTL View
- Technology View

This section describes the HDL Analyst menu commands for the RTL and Technology views. Commands may be disabled, depending on the current context. Generally, the commands enabled in any context reflect those available in the corresponding popup menus. The descriptions in the table indicate when commands are context-dependent. For explanations about the terms used in the table, such as filtered and unfiltered, transparent and opaque, see *Filtered and Unfiltered Schematic Views*, on page 85 and *Transparent and Opaque Display of Hierarchical Instances*, on page 91. For procedures on using the HDL Analyst tool, see *Analyzing With the Standard HDL Analyst Tool*, on page 336 of the *User Guide*.

For ease of use, the commands have been divided into sections that correspond to the divisions in the HDL Analyst menu.

- HDL Analyst Menu->RTL and Technology View Submenus, on page 412
- HDL Analyst Menu: Hierarchical and Current Level Submenus, on page 413
- HDL Analyst Menu: Filtering and Flattening Commands, on page 415
- HDL Analyst Menu: Timing Commands, on page 419
- HDL Analyst Menu: Analysis Commands, on page 419
- HDL Analyst Menu: Selection Commands, on page 423
- HDL Analyst Menu: FSM Commands, on page 423

HDL Analyst Menu->RTL and Technology View Submenus

This table describes the commands that appear on the HDL Analyst->RTL and HDL Analyst->Technology submenus when the RTL or Technology View is active. For procedures on using these commands, see *Analyzing With the Standard HDL Analyst Tool*, on page 336 of the *User Guide*.

HDL Analyst Command	Description
 RTL->Hierarchical View	Opens a new, hierarchical RTL view. The schematic is unfiltered.
RTL->Flattened View	Opens a new RTL view of your entire design, with a flattened, unfiltered schematic at the level of generic logic cells. See *Usage Notes for Flattening* , on page 417 for some usage tips.
 Technology->Hierarchical View	Opens a new, hierarchical Technology view. The schematic is unfiltered.
Technology->Flattened View	Creates a new Technology view of your entire design, with a flattened, unfiltered schematic at the level of technology cells. See *Usage Notes for Flattening* , on page 417 for tips about flattening.
Technology->Flattened to Gates View	Creates a new Technology view of your entire design, with a flattened, unfiltered schematic at the level of Boolean logic gates. See *Usage Notes for Flattening* , on page 417 for tips about flattening.
Technology->Hierarchical Critical Path	Creates a new Technology view of your design, with a hierarchical, *filtered* schematic showing only the instances and paths whose slack times are within the slack margin you specified in the Slack Margin dialog. This command automatically enables HDL Analyst->Show Timing Information.
Technology->Flattened Critical Path	Creates a new Technology view of your design, with a flattened, *filtered* schematic showing only the instances and paths whose slack times are within the slack margin you specified in the Slack Margin dialog. This command automatically enables HDL Analyst->Show Timing Information. See *Usage Notes for Flattening* , on page 417 for tips about flattening.

HDL Analyst Menu: Hierarchical and Current Level Submenus

This table describes the commands on the HDL Analyst->Hierarchical and HDL Analyst->Current Level submenus. For procedures on using these commands, see *Analyzing With the Standard HDL Analyst Tool*, on page 336 of the *User Guide.*

HDL Analyst Command	Description
Hierarchical->Expand	Expands paths from selected pins and/or ports up to the nearest objects on any hierarchical level, according to pin/port directions. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level. Successive Expand commands expand the paths further, based on the new current selection.
Hierarchical->Expand to Register/Port	Expands paths from selected pins and/or ports, in the port/pin direction, up to the next register, port, or black box. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.
Hierarchical->Expand Paths	Shows all logic, on any hierarchical level, between two or more selected instances, pins, or ports. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.
Hierarchical->Expand Inwards	Expands within the hierarchy of an instance, from the lower-level ports that correspond to the selected pins, to the nearest objects and no further. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.
Hierarchical->Goto Net Driver	Displays the unfiltered schematic sheet that contains the net driver for the selected net. Operates hierarchically, on lower schematic levels as well as the current level.
Hierarchical->Select Net Driver	Selects the driver for the selected net. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.
Hierarchical->Select Net Instances	Selects instances connected to the selected net. The result is a *filtered* schematic. Operates hierarchically, on lower schematic levels as well as the current level.
Current Level->Expand	Expands paths from selected pins and/or ports up to the nearest objects on the current level, according to pin/port directions. The result is a *filtered* schematic. Limited to all sheets on the current schematic level. This command is only available if a HDL Analyst view is open. Successive Expand commands expand the paths further, based on the new current selection.

HDL Analyst Command	Description
Current Level->Expand to Register/Port	Expands paths from selected pins and/or ports, according to the pin/port direction, up to the next register, ports, or black box on the current level. The result is a *filtered* schematic. Limited to all sheets on the current schematic level.
Current Level->Expand Paths	Shows all logic on the current level between two or more selected instances, pins, or ports. The result is a *filtered* schematic. Limited to the current schematic level (all sheets).
Current Level->Goto Net Driver	Displays the unfiltered schematic sheet that contains the net driver for the selected net. Limited to all sheets on the current schematic level.
Current Level->Select Net Driver	Selects the driver for the selected net. The result is a *filtered* schematic. Limited to all sheets on the current schematic level.
Current Level->Select Net Instances	Selects instances on the current level that are connected to the selected net. The result is a *filtered* schematic. Limited to all sheets on the current schematic level.

HDL Analyst Menu: Filtering and Flattening Commands

This table describes the filtering and flattening commands on the HDL Analyst menu. For procedures on filtering and flattening, see *Analyzing With the Standard HDL Analyst Tool*, on page 336 of the *User Guide*.

HDL Analyst Command	Description
 Filter Schematic	Filters your entire design to show only the selected objects. The result is a *filtered* schematic. For more information about using this command, see *Filtering Schematics* , on page 340 of the *User Guide*. This command is only available with an open HDL Analyst view.

© 2020 Synopsys, Inc.

HDL Analyst Command	Description
Flatten Current Schematic (Unfiltered Schematic)	In an unfiltered schematic, the command flattens the current schematic, at the current level and all levels below. In an RTL view, the result is at the generic logic level. In a Technology view, the result is at the technology-cell level. See the next table entry for information about flattening a filtered schematic. This command does not do the following: • Flatten your entire design (unless the current level is the top level) • Open a new view window • Take into account the number of Dissolve Levels defined in the Schematic Options dialog box See *Usage Notes for Flattening* , on page 417 for tips.

HDL Analyst Command	Description
Flatten Current Schematic (Filtered Schematic)	In a filtered schematic, flattening is a two-step process: • Only unhidden transparent instances (including nested ones) are flattened in place, in the context of the entire design. Opaque and hidden hierarchical instances remain hierarchical. The effect of this command is that all hollow boxes with pale yellow borders are removed from the schematic, leaving only what was displayed inside them. • The original filtering is restored. In an RTL view, the result is at the generic logic level. In a Technology view, the result is at the technology-cell level. This command does not do the following: • Flatten everything inside a transparent instance. It only flattens transparent instances and any nested transparent instances they contain. • Open a new view window. • Take into account the number of Dissolve Levels defined in the Schematic Options dialog box. See *Usage Notes for Flattening* , on page 417 for usage tips.
Unflatten Current Schematic	Undoes any flattening operations and returns you to the original schematic, as it was before flattening and any filtering. This command is available only if you have explicitly flattened a hierarchical schematic using HDL Analyst->Flatten Current Schematic, for example. It is not available for flattened schematics created directly with the RTL and Technology submenus of the HDL Analyst menu.

Usage Notes for Flattening

It is usually more memory-efficient to flatten only parts of your design, as needed. The following are a few tips for flattening designs with different commands. For detailed procedures, see *Flattening Schematic Hierarchy*, on page 347 of the *User Guide*.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

RTL/Technology->Flattened View Commands

- Use Flatten Current Schematic to flatten only the current hierarchical level and below.
- Flatten selected hierarchical instances with Dissolve Instances (followed by Flatten Current Schematic, if the schematic is filtered).
- To make hierarchical instances transparent without flattening them, use Dissolve Instances in a filtered schematic. This shows their details nested inside the instances.

Flatten Current Schematic Command (Unfiltered View)

- Flatten selected hierarchical instances with Dissolve Instances.
- To see the lower-level logic inside a hierarchical instance, push into it instead of flattening.
- Selectively flatten your design by hiding the instances you do not need, flattening, and then unhiding the instances.
- Flattening erases the history of displayed sheets for the current view. You can no longer use View->Back. You can, however, use UnFlatten Schematic to get an unflattened view of the design.

Flatten Current Schematic Command (Filtered View)

- Flatten selected hierarchical instances with Dissolve Instances, followed by Flatten Current Schematic.
- Selectively flatten your design by hiding the instances you do not need, flattening, and then unhiding the instances.
- Flattening erases the history of displayed sheets for the current view. You can no longer use View->Back. You can do the following:
- Use View->Back for a view of the transparent instance flattened in the context of the entire design. This is the view generated after step 1 of the two-step flattening process described above. Use UnFlatten Schematic to get an unflattened view of the design.

HDL Analyst Menu: Timing Commands

This table describes the timing commands on the HDL Analyst menu. For procedures on using the timing commands, see *Analyzing With the Standard HDL Analyst Tool*, on page 336 of the *User Guide*.

HDL Analyst Command	Description
Set Slack Margin	Displays the Slack Margin dialog box, where you set the slack margin. HDL Analyst->Show Critical Path displays only those instances whose slack times are worse than the limit set here. Available only in a Technology view.
 Show Critical Path	Filters your entire design to show only the instances and paths whose slack times exceed the slack margin set with Set Slack Margin, above. The result is flat if the entire design was already flat. This command also enables Show Timing Information (see below). Available only in a Technology view.
Show Timing Information	When enabled, Technology view schematics are annotated with timing numbers above each instance. The first number is the cumulative path delay; the second is the slack time of the worst path through the instance. Negative slack indicates that timing has not met requirements. Available only in a Technology view. For more information, see *Viewing Timing Information* , on page 358 on the *User Guide*.

HDL Analyst Menu: Analysis Commands

This table describes the analysis commands on the HDL Analyst menu. For procedures on using the analysis commands, see *Analyzing With the Standard HDL Analyst Tool*, on page 336 of the *User Guide*.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

HDL Analyst Command	Description
Isolate Paths	Filters the current schematic to display only paths associated with all the pins of the selected instances. The paths follow the pin direction (from output to input pins), up to the next register, black box, port, or hierarchical instance. If the selected objects include ports and/or pins on unselected instances, the result also includes paths associated with those selected objects. The range of the operation is all sheets of a filtered schematic or just the current sheet of an unfiltered schematic. The result is always a filtered schematic. In contrast to the Expand operations, which add to what you see, Isolate Paths can only remove objects from the display. While Isolate Paths is similar to Expand to Register/Port, Isolate Paths reduces the display while Expand to Register/Port augments it.
Show Context	Shows the original, unfiltered schematic sheet that contains the selected instance. Available only in a filtered schematic.
Hide Instances	Hides the logic inside the selected hierarchical (non-primitive) instances. This affects only the active HDL Analyst view; the instances are not hidden in other HDL Analyst views. The logic inside hidden instances is not loaded (saving dynamic memory), and it is unrecognized by searching, dissolving, flattening, expansion, and push/pop operations. (Crossprobing does recognize logic inside hidden instances, however.) See *Usage Notes for Hiding Instances , on page 422* for tips.
Unhide Instances	Undoes the effect of Hide Instances: the selected hidden hierarchical instances become visible (susceptible to loading, searching, dissolving, flattening, expansion, and push/pop operations). This affects only the current HDL Analyst view; the instances are not hidden in other HDL Analyst views.

HDL Analyst Command	Description
Show All Hier Pins	Shows all pins on the selected transparent, non-primitive instances. Available only in a filtered schematic. Normally, transparent instance pins that are connected to logic that has been filtered out are not displayed. This command lets you display these pins that connected to logic that has been filtered out. Pins on primitives are always shown.
Dissolve Instances	Shows the lower-level details of the selected non-hidden hierarchical instances. The number of levels dissolved is determined by the Dissolve Levels value in the HDL Analyst Options dialog box (*Standard HDL Analyst Options Command* , on page 444). For usage tips, see *Usage Notes for Dissolving Instances* , on page 422.
Dissolve to Gates	Dissolves the selected instances by flattening them to the gate level. This command displays the lower-level hierarchy of selected instances, but it dissolves technology primitives as well as hierarchical instances. Technology primitives are dissolved to generic synthesis symbols. The command is only available in the Technology view. The number of levels dissolved is determined by the Dissolve Levels value in the HDL Analyst Options dialog box (*Standard HDL Analyst Options Command* , on page 444). Dissolving an instance one level redraws the current sheet, replacing the hierarchical dissolved instance with the logic you would see if you pushed into it using Push/pop mode. Unselected objects or selected hidden instances are not dissolved. The effect of the command varies: • In an unfiltered schematic, this command *flattens* the selected instances. This means the history of displayed sheets is removed. The resulting schematic is unfiltered. • In a filtered schematic, this command makes the selected instances *transparent*, displaying their internal, lower-level logic inside hollow boxes. History is retained. You can use Flatten Schematic to flatten the transparent instances, if necessary. The resulting schematic if filtered.

Synopsys Confidential Information

Usage Notes for Hiding Instances

The following are a few tips for hiding instances. For detailed procedures, see *Flattening Schematic Hierarchy*, on page 347of the *User Guide*.

- Hiding hierarchical instances soon after startup can often save memory. After the interior of an instance has been examined (by searching or displaying), it is too late for this savings.
- You can save memory by creating small, temporary working files: File->Save As .srs or .srm files does not save the hidden logic (hidden instances are saved as black boxes). Restarting the synthesis tool and loading such a saved file can often result in significant memory savings.
- You can selectively flatten instances by temporarily hiding all the others, flattening, then unhiding.
- You can limit the range of Edit->Find (see *Find Command (HDL Analyst)*, on page 317) to prevent it looking inside given instances, by temporarily hiding them.

Usage Notes for Dissolving Instances

Dissolving an instance one level redraws the current sheet, replacing the hierarchical dissolved instance with the logic you would see if you pushed into it using Push/pop mode. Unselected objects or selected hidden instances are not dissolved. For additional information about dissolving instances, see *Flattening Schematic Hierarchy*, on page 347 of the *User Guide*.

The type (filtered or unfiltered) of the resulting schematic is unchanged from that of the current schematic. However, the effect of the command is different in filtered and unfiltered schematics:

- In an unfiltered schematic, this command flattens the selected instances. This means the history of displayed sheets is removed.
- In a filtered schematic, this command makes the selected instances transparent, displaying their internal, lower-level logic inside hollow boxes. History is retained. You can use Flatten Schematic to flatten the transparent instances, if necessary. This command is only available if an HDL Analyst view is open.

© 2020 Synopsys, Inc.

HDL Analyst Menu: Selection Commands

This table describes the selection commands on the HDL Analyst menu.

HDL Analyst Command	Description
Select All Schematic ->Instances ->Ports	Selects all Instances or Ports, respectively, on all sheets of the current schematic. All other objects are unselected. This does not select objects on other schematics.
Select All Sheet ->Instances ->Ports	Selects all Instances or Ports, respectively, on the current schematic sheet. All other objects are unselected.
Unselect All	Unselects all objects in all HDL Analyst views.

HDL Analyst Menu: FSM Commands

This table describes the FSM commands on the HDL Analyst menu.

HDL Analyst Command	Description
View FSM	Displays the selected finite state machine in the FSM Viewer. Available only in an RTL view.
View FSM Info File	Displays information about the selected finite state machine module, including the number of states, the number of inputs, and a table of the states and transitions. Available only in an RTL view.

Options Menu

Use the Options menu to configure the VHDL and Verilog compilers, customize toolbars, and set options for the Project view, Text Editor, and HDL Analyst schematics. When using certain technologies, additional menu commands let you run technology-vendor software from this menu.

The following table describes the Options menu commands.

Command	Description
Basic Options Menu Commands for all Views	
Configure VHDL Compiler	Opens the Implementation Options dialog box where you can set the top-level entity and the encoding method for enumerated types. State-machine encoding is automatically determined by the FSM compiler or FSM explorer, or you can specify it explicitly using the syn_encoding attribute. See *Implementation Options Command* , on page 346.
Configure Verilog Compiler	Opens the Implementation Options dialog box where you can specify the top-level module and the include search path. See *Implementation Options Command* , on page 346.
Configure Parallel or Compile Point Process	Lets you specify the maximum number of parallel synthesis jobs that can be run and how errors are treated, for example with compile points. See *Configure Parallel or Compile Point Process Command* , on page 425.
Toolbars	Lets you customize the toolbars.
Project View Options	Sets options for organizing files in the Project view. See *Project View Options Command* , on page 429.
Editor Options	Sets your Text Editor syntax coloring, font, and tabs. See *Editor Options Command* , on page 436.
P&R Environment Options	Displays the environmental variable options set for the place-and-route tool. See *Place and Route Environment Options Command* , on page 439.

© 2020 Synopsys, Inc.

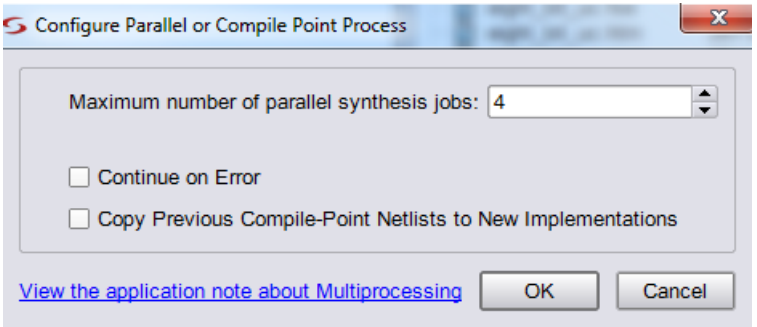
Command	Description
Configure 3rd Party Tool Options	Lets you invoke third-party tools, for example to modify the files generated or debug problems from the EDK tool within the FPGA synthesis products. See *Configure 3rd Party Tools Options Command* , on page 440.
Project Status Page Location	Saves the current project status to a location of your choice. See *Project Status Page Location* , on page 441.
HDL Analyst Options	Sets display preferences for New HDL Analyst schematics (RTL and Technology views). See *HDL Analyst Options Command* , on page 443.
Standard HDL Analyst Options	Sets display preferences for HDL Analyst schematics (RTL and Technology views). See *Standard HDL Analyst Options Command* , on page 444.
Configure External Programs	Lets you set browser and Acrobat Reader options on Linux platforms. See *Configure External Programs Command* , on page 453.
Options Menu Commands Specifically for the Project View	
Configure Identify Launch	If Identify software is not properly installed, you might run into problems when you try to launch it from the synthesis tools. Use the Configure Identify Launch dialog box to help you resolve these issues. For guidelines to follow, see *Handling Problems with Launching Identify* , on page 560 in the *User Guide*.

Configure Parallel or Compile Point Process Command

Use the Configure Parallel or Compile Point Process command to run parallel jobs for multiprocessing with compile points or distributed processing. For example, this option allows the software to run multiple, independent compile point jobs simultaneously, providing additional runtime improvements for the compile point synthesis flow.

This feature is supported on Windows and Linux for certain technologies only. The command is disabled for technologies that are not supported.

© 2020 Synopsys, Inc.


Configure Parallel or Compile Point Process
Maximum number of parallel synthesis jobs: 4
Continue on Error
Copy Previous Compile-Point Netlists to New Implementations
View the application note about Multiprocessing
OK
Cancel

Field/Option	Description
Maximum Number of Parallel Synthesis Jobs	Sets the maximum number of parallel synthesis jobs to run multiprocessing with compile points or distributed processing. It displays the current value from the .ini file, and allows you to reset it. Use this option for multiprocessing or distributed processing to run jobs in parallel. Set a value based on the number of available licenses. With each license, you can run four jobs in parallel by default. See *Multiprocessing With Compile Points* , on page 536. When you set this option, it resets the MaxParallelJobs value in the .ini file. See *Setting Number of Parallel Jobs* , on page 513 in the *User Guide* and *Maximum Parallel Jobs* , on page 428 for other ways to specify this value.
Continue on Error	Allows the software to continue on an error in a compile point and synthesize the rest of the design, even when there might be problems with a portion of the design. The Continue on Error mode automatically enables the MultiProcessing option to run with compile points using a single license; this is the default. For additional runtime improvements, you can specify multiple synthesis jobs that run in parallel. See Chapter 14, *Improving Runtime* for details. For more information about Continue on Error mode during compile-point synthesis, see *Combining Compile Points with Multiprocessing* , on page 470 in the *User Guide*. Tcl equivalent: **set_option -continue_on_error 0|1**
Copy Previous Compile-Point Netlists to New Implementation	Allows you to copy compile point netlist (.srd) files generated from the previous implementation into a new implementation that has been created.

Synopsys Confidential Information

Maximum Parallel Jobs

There are three ways to specify the maximum number of parallel jobs:

.ini File	Set this variable in the MaxParallelJobs variable in the product `ini` file:
	`[JobSetting]`
	`MaxParallelJobs=<n>`
	This value is used by the UI as well as in batch mode, and is effective until you specify a new value. You can change it with the Options->Configure Parallel or Compile Point Process command.

Tcl Variable	Set the following variable in a Tcl file, the project files, or from the Tcl window: `set_option -max_parallel_jobs=<n>` This is a global option that is applied to all project files and their implementations. This value takes effect immediately. If you set it in the Tcl file or project file, it remains in effect until you specify a new value. If you set it from the Tcl window, the max_parallel_jobs value is only effective for the session and will be lost when you exit the application.
Configure Compile Point Process Command	The Maximum Number of Parallel Synthesis Jobs option displays the current ini file value and allows you to reset it.

License Utilization for Multiprocessing

When you decide to run parallel synthesis jobs, additional licenses may be required for the compile point jobs. By default, four parallel jobs use one license. For example, if you set the Maximum number of parallel synthesis jobs to 12, the synthesis tool consumes one license to run 4 compile point jobs and can utilize the two additional licenses to run 8 more parallel jobs if they are available for your computing environment. Licenses are released as jobs complete, and then consumed by new jobs which need to run.

The actual number of licenses utilized depends on the following:

1. Synthesis software scheme for the compile point requirements used to determine the maximum number of parallel jobs or licenses a particular design tries to use.
2. Value set on the Configure Parallel or Compile Point Process dialog box.

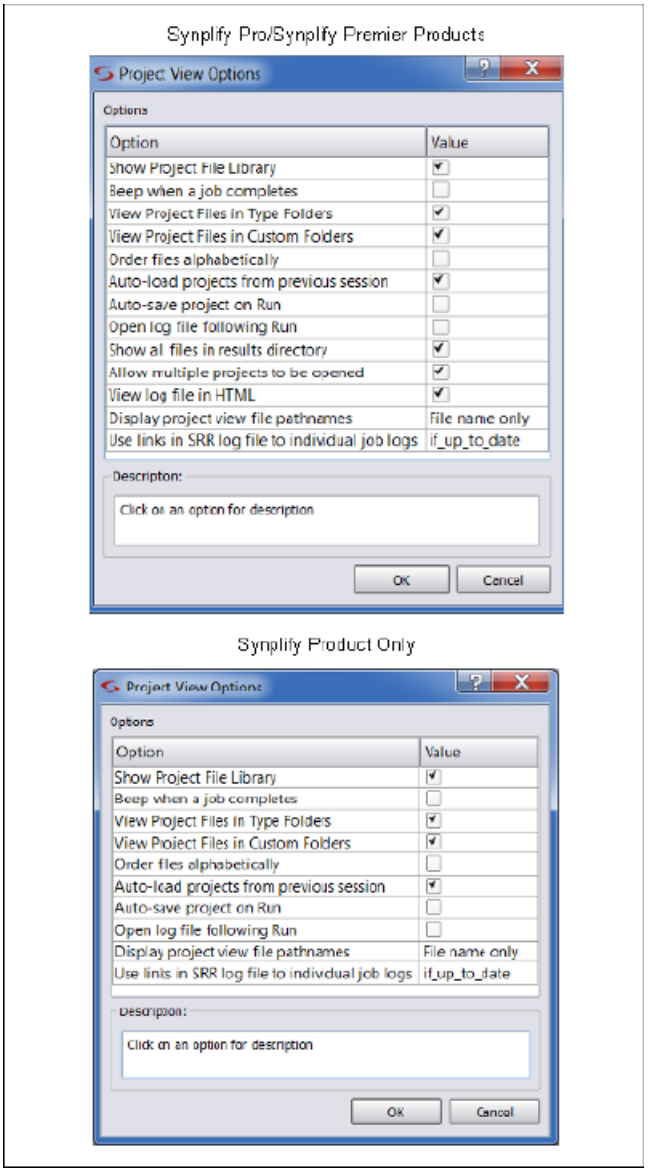
3. Number of licenses actually available. You can use Help->Preferred License Selection to check the number of available license. If you need to increase the number of available licenses, you can specify multiple license types. For more information, see *Using Different License Types for Multiprocessing*, on page 538.

Note that factors 1 and 3 above can change during a single synthesis run. The number of jobs equals the number of licenses, which then equates the lowest value of these three factors.

Project View Options Command

Select Options->Project View Options to display the Project View Options dialog box, where you define how projects appear and are organized in the Project view.

© 2020 Synopsys, Inc.


Synplify Pro/Synplify Premier Products
Project View Options
Options
Option
Value
Show Project File Library
Beep when a job completes
View Project Files in Type Folders
View Project Files in Custom Folders
Order files alphabetically
Auto-load projects from previous session
Auto-save project on Run
Open log file following Run
Show all files in results directory
Allow multiple projects to be opened
View log file in HTML
Display project view file pathnames
File name only
Use links in SRR log file to individual job logs
if_up_to_date
Description:
Click on an option for description
OK
Cancel
Synplify Product Only
Project View Options
Options
Option
Value
Show Project File Library
Beep when a job completes
View Project Files in Type Folders
View Project Files in Custom Folders
Order files alphabetically
Auto-load projects from previous session
Auto-save project on Run
Open log file following Run
Display project view file pathnames
File name only
Use links in SRR log file to individual job logs
if_up_to_date
Description:
Click on an option for description
OK
Cancel

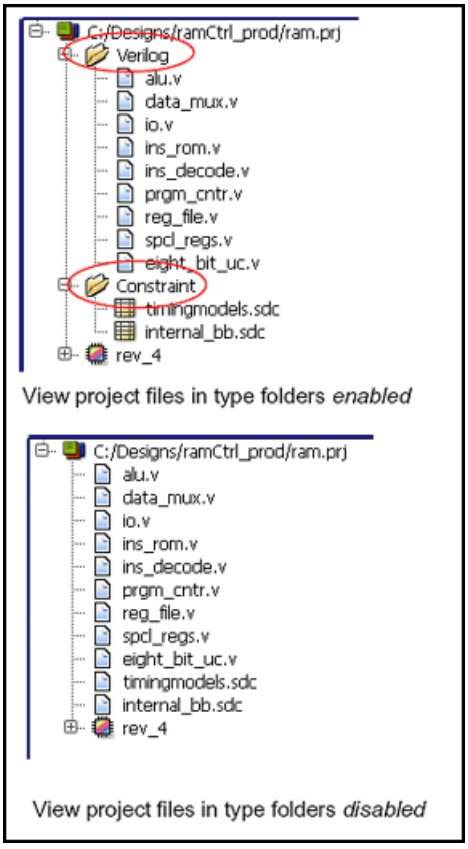
The following table describes the Project View Options dialog box features.

Field/Option	Description
Show Project File Library	When enabled, displays the corresponding VHDL library next to each source VHDL filename, in the Project Tree view of the Project view. For example, with library dune, file pc.vhd is listed as [dune] pc.vhd if this option is enabled, and as pc.vhd if it is disabled. See also *Set VHDL Library Command* , on page 337, for how to change the library of a file.
Beep when a job completes	When enabled, sounds an audible signal whenever a project finishes running.
View Project Files in Type Folders	When enabled, organizes project files into separate folders by type. See *View Project Files in Type Folders Option* , on page 433 and add_file, on page 23.
View Project Files in Custom Folders	When enabled, allows you to view files contained within the custom folders created for the project. See *View Project Files in Custom Folders Option* , on page 433.
Order files alphabetically	When enabled, the software orders the files within folders alphabetically instead of in project order. You can also use the Sort Files option in the Project view.
Autoload projects from previous session	Enable/Disable automatically loading projects from the previous session. Otherwise, projects will not be loaded automatically. This option is enabled by default. See *Loading Projects With the Run Command* , on page 434.
Auto-save project on Run	Enable/Disable automatically saving projects when the Run button is selected. See *Automatically Save Project on Run* , on page 435.
Open Log file following Run	Enable/Disable automatically opening and displaying log file after a synthesis run.
Show all files in results directory	When enabled, shows all files in the Implementation Results view. When disabled, the results directory shows only files generated by the synthesis tool itself.

© 2020 Synopsys, Inc.

Field/Option	Description
Allow multiple projects to be opened	When enabled, multiple projects are displayed at the same time. See *Allow Multiple Projects to be Opened Option* , on page 434.
View log file in HTML	Enable/Disable viewing of log file report in HTML format versus text format. See *Log File* , on page 157.
Project file name display	From the drop-down menu, select one the following ways to display project files: • File name only • Relative file path • Full file path
Use links in SRR log file to individual job logs	Determines if individual job logs use links in the `srr` log file. You can select: • off—appends individual job logs to the `srr` log file. • on—always link to individual job logs. • if_up_to_date—only links to individual job logs if the module is up-to-date.

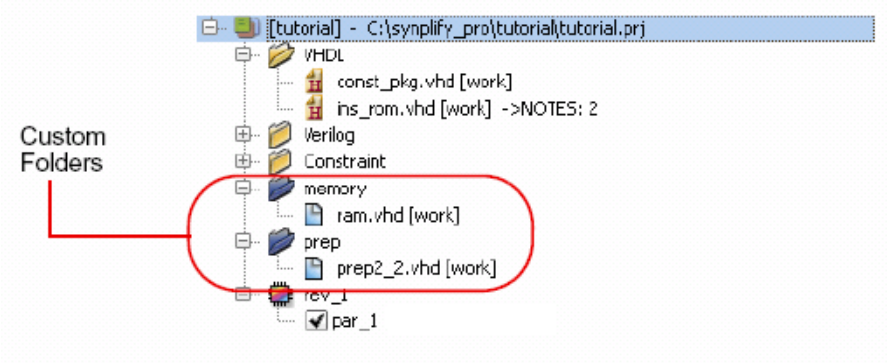
View Project Files in Type Folders Option



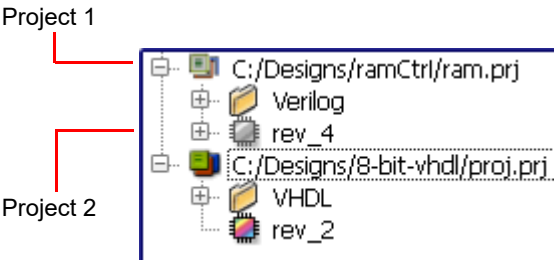
View Project Files in Custom Folders Option

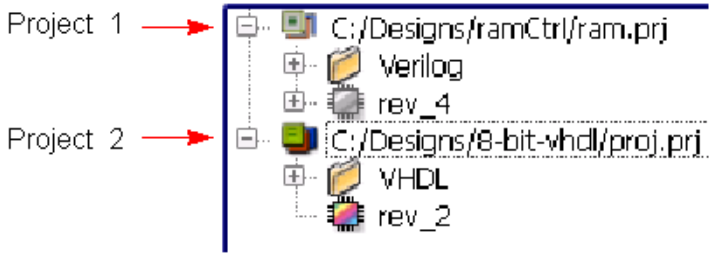
Selecting this option enables you to view user-defined custom folders that contain a predefined subset of project files in various hierarchy groupings or organizational structures. Custom folders are distinguished by their blue color. For information on creating custom folders, see *Creating Custom Folders, on page 70* in the *User Guide*.

© 2020 Synopsys, Inc.



Allow Multiple Projects to be Opened Option





Loading Projects With the Run Command

When you load a project that includes the project -run command, a dialog box appears in the Project view with the following message:

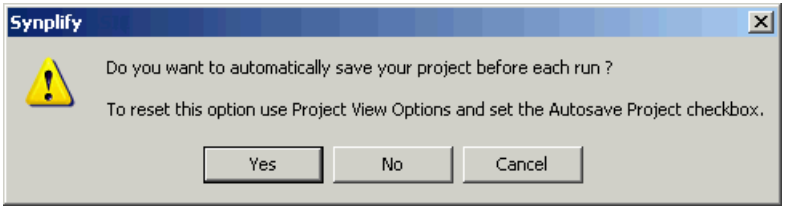
```
Project run command encountered during project load. Are you sure
you want to run?
```

You can reply with either yes or no.

Automatically Save Project on Run

If you have modified your project on the disk directory since being loaded into the Project view and you run your design, a message is generated that infers the UI is out-of-date.

The following dialog box appears with a message to which you must reply.



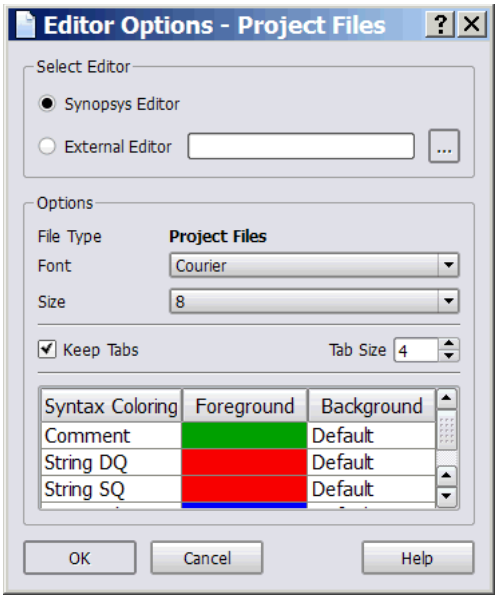
You can specify one of the following:

- Yes — The Auto-save project on Run switch on the Project View Options dialog box is automatically enabled, and then your design is run.
- No — The Auto-save project on Run switch on the Project View Options dialog box is not enabled, but your design is run.
- Cancel — Closes this message dialog box and does not run your design.

© 2020 Synopsys, Inc.

Editor Options Command

Select Options->Editor Options to display the Editor Options dialog box, where you select either the internal text editor or an external text editor.



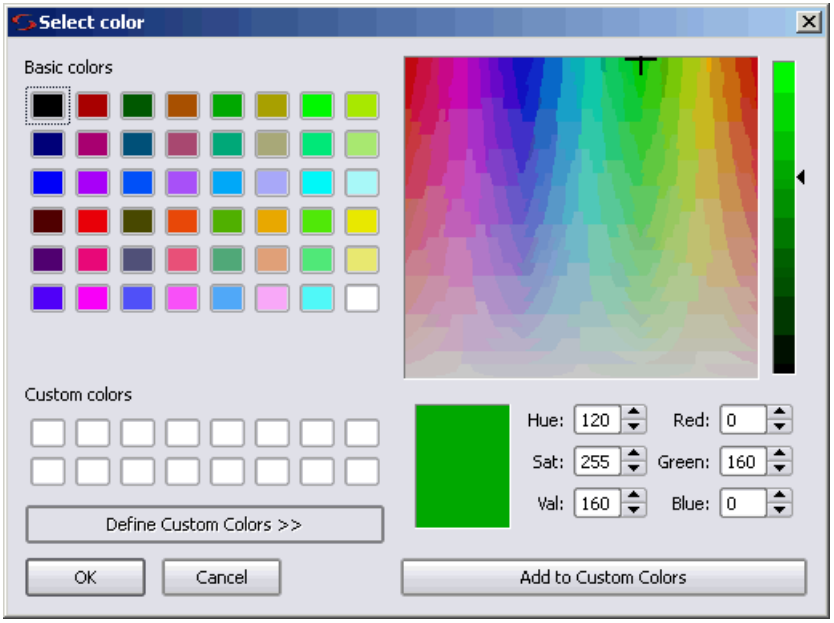
The following table describes the Editor Options dialog box features.

Feature	Description
Select Editor	Select an internal or external editor.
• Synopsys Editor	Sets the Synopsys text editor as the default text editor.
• External Editor	Uses the specified external text editor program to view text files from within the Synopsys tool. The executable specified must open its own window for text editing. See *Using an External Text Editor* , on page 44 of the *User Guide* for a procedure. *Note:* Files opened with an external editor cannot be crossprobed.
Options	Set text editing preferences.
• File Type	You can define text editor preferences for the following file types: project files, HDL files, log files, constraint files, and default files.

Feature	Description
• Font	Lets you define fonts to use with the text editor.
• Font Size	Lets you define font size to use with the text editor.
• Keep Tabs • Tab Size	Lets you define whether to use tab settings with the text editor.
• Syntax Coloring	Lets you define foreground or background syntax coloring to use with the text editor. See *Color Options* , on page 437.

Color Options

Click in the Foreground or Background field for the corresponding object in the Syntax Coloring field to display the color palette.



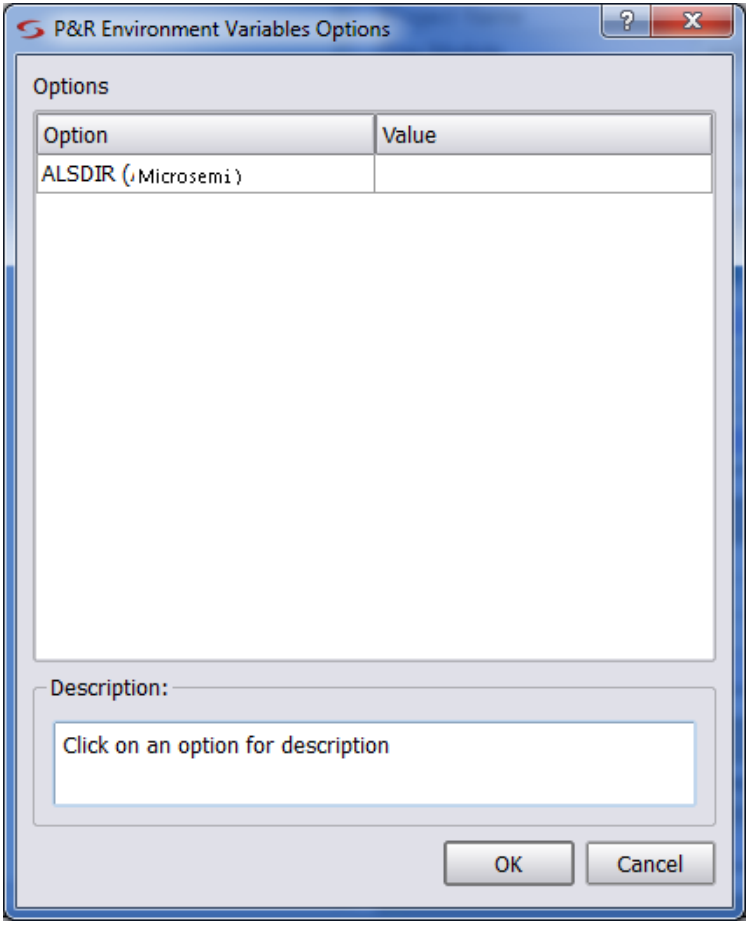
You can set syntax colors for some common syntax options listed in the following table.

Syntax	Description
Comment	Comment strings contained in all file types.
Error	Error messages contained in the log file.
Gates	Gates contained in HDL source files.
Info	Informational messages contained in the log file.
Keywords	Generic keywords contained in the project, HDL source, constraint, and log files.
Line Comment	Line comments contained in the HDL source, C, C++, and log files.
Note	Notes contained in the log file.
SDCKeyword	Constraint-specific keywords contained in the .sdc file.
Strength	Strength values contained in HDL source files.
String DQ	String values within double quotes contained in the project, HDL source, constraint, C, C++, and log files.
String SQ	String values within single quotes contained in the project, HDL source, constraint, C, C++, and log files.
SVKeyword	SystemVerilog keywords contained in the Verilog file.
Types	Type values contained in HDL source files.
Warning	Warning messages contained in the log file.

© 2020 Synopsys, Inc.

Place and Route Environment Options Command

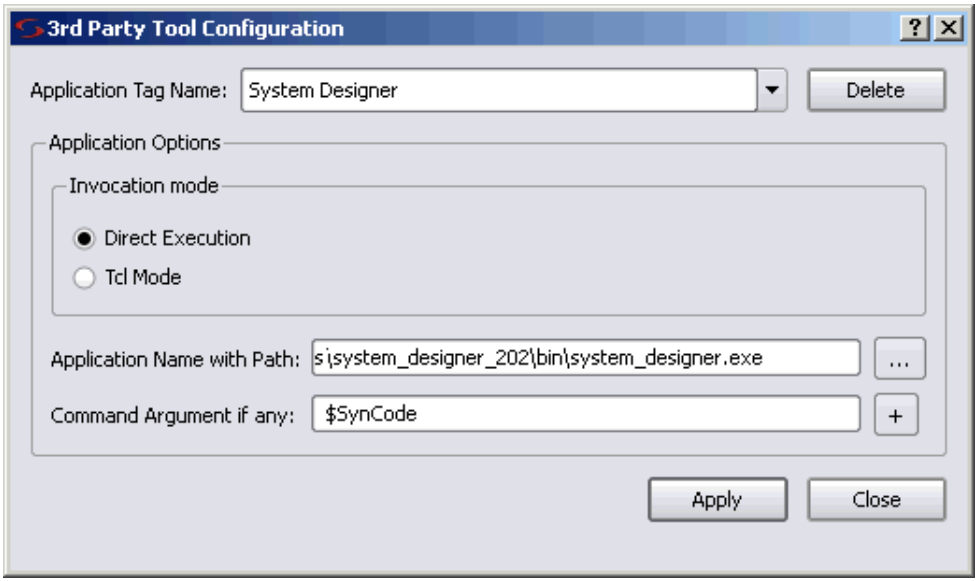
Select Options->P&R Environment Options to display the environment variable options set for the place-and-route tool. This option allows you to change the specified location of the selected place-and-route tool set on your system; the software locates and runs this updated version of the P&R tool for the current session of the synthesis tool.



Configure 3rd Party Tools Options Command

Use the Configure 3rd Party Tools Option command to invoke third-party tools, such as the Embedded Development Kit (EDK) from within the Synopsys FPGA products. This allows you to modify source files or libraries added to your synthesis projects from within the third-party tool directly. Use the following dialog box to configure the location and common arguments for the tools.

For more information, see *Invoking Third-Party Vendor Tools*, on page 530 in the *User Guide*.



The 3rd Party Tool Configuration dialog box includes the following options:

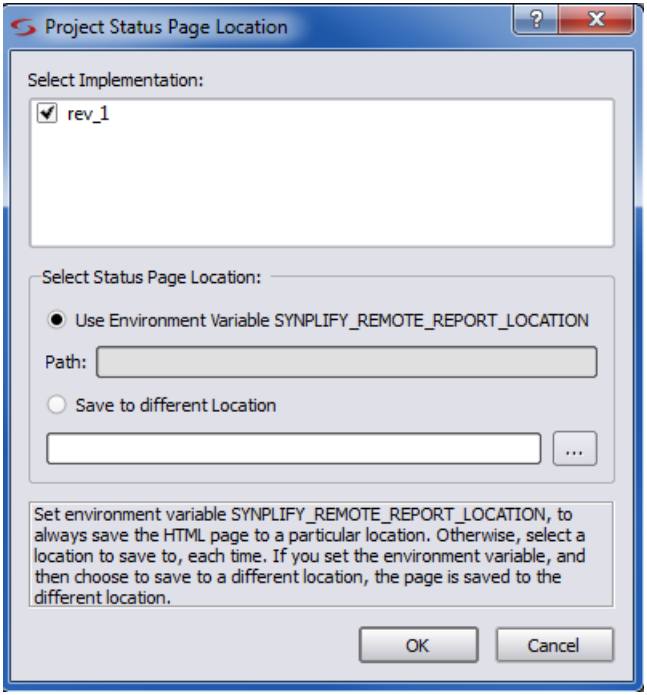
Feature	Description
Application Tag Name	Specifies an application or Tcl procedure name. Type in the name or select a preconfigured application from the list.
Direct Execution	Sets up the direct invocation of a third-party tool from within the FPGA synthesis tool, using the path defined for the executable in Application Name with Path.

Feature	Description
Tcl Mode	Sets up the tool to execute the Tcl procedure from within the FPGA synthesis tool, using the path defined for the procedure in Tcl Procedure Name. You must execute this Tcl script from the Tcl window to register the invocation procedure for the third-party tool.
Application Name with Path	When using direct execution, specifies the path to the executable for the application.
Tcl Procedure Name	When using Tcl mode, specifies the Tcl procedure name.
Command Argument if any	Defines any additional arguments for the third-party application. You can select arguments from the drop-down list or type them. Note: For internal Synopsys tools, you must include the $Syncode parameter.
Procedure Arguments if any	Defines additional arguments for the Tcl procedure. You can select arguments from the drop-down list or type them.

Project Status Page Location

Lets you save the current project status to a location of your choice. You can then view the project status offline with any browser on a mobile device.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

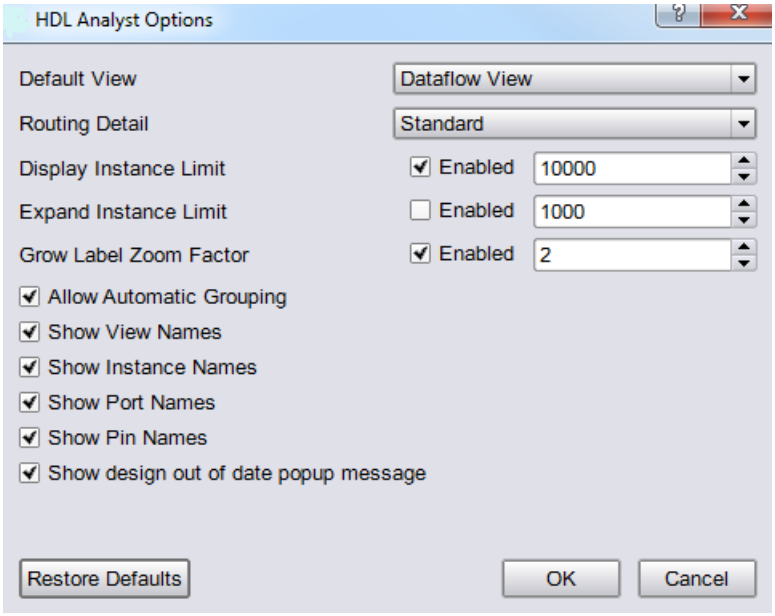


The following table describes the Project Status Page Location dialog box options.

Option	Description
Select Implementation	Select the implementation for the design for which you want synthesis results. You can select multiple implementations.
Select Status Page Location • Use Environment Variable SYNPLIFY_REMOTE_REPORT_LOCATION • Save to Different Location	Select the location on your computer where you want to save the project status reports: • Use the environment variable to specify a standard location for the project status reports. Choose this option if you always want to save the reports to the same location. • Choose a location for the project status reports for the current implementation. You can change this as often as you like. For more information, see *Accessing Results Remotely* , on page 195 in the *User Guide*.

HDL Analyst Options Command

Select Options->Schematic Options to display a dialog box where you define preferences for the HDL Analyst schematic. For details see *Setting Schematic Preferences, on page 242* in the *User Guide*.



The following options are on the HDL Analyst Options panel.

Field/Option	Description
Default View	Specify how you want the schematic views to display: • Clocks View - Displays all sequential elements connected to clock nets so that clocks in the design can be debugged. • Dataflow View - Displays objects from a left to right datapath flow. This is the default.
Routing Detail	Specify how the tool determines the detailed routing for the design: • Standard - This is the default. • Quick - Direct net connections
Display Instance Limit	When enabled, uses the specified limit to display instances. The default is 10000.

© 2020 Synopsys, Inc.

Field/Option	Description
Expand Instance Limit	When enabled, uses the specified limit to expand instances. The default is 1000.
Grow Label Zoom Factor	Specify a zoom factor for labels displayed in the schematic view. Select a value between 1 and 10, where labels are shown increasing in size respectively. Changes will appear in the next opened schematic view. The default is 2.
Allow Automatic Grouping	When enabled, automatic grouping is performed.
Show View Names	When enabled, module names are displayed.
Show Instance Names	When enabled, instance names are displayed.
Show Port Names	When enabled, port names are displayed.
Show Pin Names	When enabled, pin names are displayed.
Show design out of date popup message	When enabled, shows the design out of date popup message.
Restore Defaults	Click this button to reset all options to their defaults.

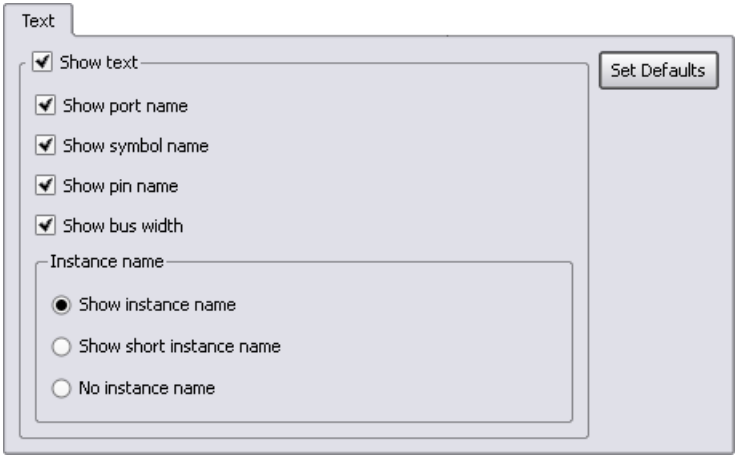
Standard HDL Analyst Options Command

Select Options->HDL Analyst Options to display the HDL Analyst Options dialog box, where you define preferences for the HDL Analyst schematic views (RTL and Technology views). Some preferences take effect immediately, others only take effect in the next view that you open. For details, see *Setting Schematic Preferences, on page 305* in the *User Guide*.

For information about the options, see the following, which correspond to the tabs on the dialog box:

- Text Panel, on page 445
- General Panel, on page 446
- Sheet Size Panel, on page 450
- Visual Properties Panel, on page 452

Text Panel

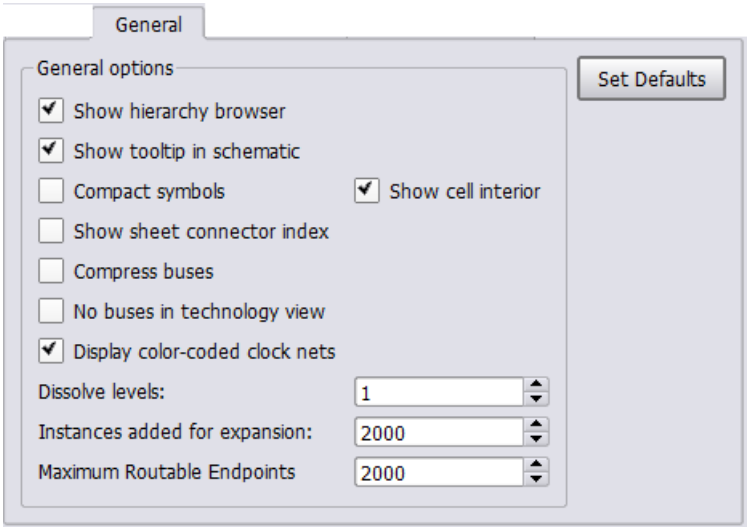


The following options are on the Text panel.

Field/Option	Description
Show text	Enables the selective display of schematic labels. Which labels are displayed is governed by the other Show * features and Instance name, described below.
Show port name	When enabled, port names are displayed.
Show symbol name	When enabled, symbol names are displayed.
Show pin name	When enabled, pin names are displayed.
Show bus width	When enabled, connectivity bit ranges are displayed near pins (in square brackets: [ ]), indicating the bits used for each bus connection.
Instance name	Determines how to display instance names: • Show instance name • Show short instance name • No instance name
Set Defaults	Set the dialog box to display the default values.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

General Panel



The following options are on the General panel.

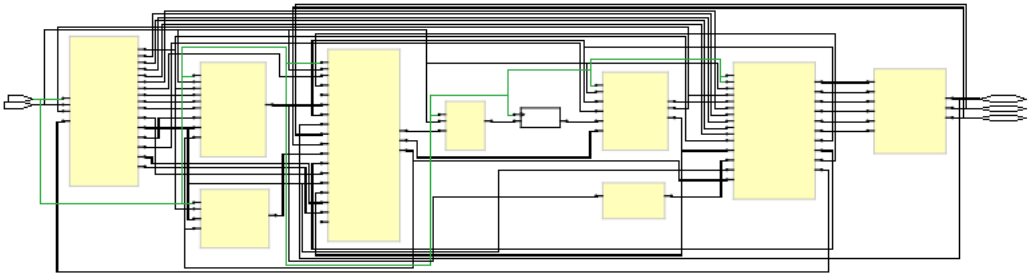
Field/Option	Description
Show hierarchy browser	When enabled, a hierarchy browser is present at the left pane of RTL and Technology views.
Show tooltip in schematic	When enabled, displays tooltips that hover objects as you move over them in the RTL and Technology schematic views.
Compact symbols	When enabled, symbols are displayed in a slightly more compact manner, to save space in schematics. When this is enabled, Show cell interior is disabled.
Show cell interior	When enabled, the internal logic of cells that are technology-specific primitives (such as LUTs) is shown in Technology views. This is not available if Compact symbols is enabled.
Show sheet connector index	When enabled, sheet connectors show connecting sheet numbers - see *Sheet Connectors* , on page 89.

Field/Option	Description
Compress buses	When enabled, buses having the same source and destination instances are displayed as bundles, to reduce clutter. A single bundle can connect to more than one pin on a given instance. The display of a bundle of buses is similar to that of a single bus.
No buses in technology view	When enabled, buses are not displayed, they are only indicated as bits in a Technology View. This applies only to flattened views created by HDL Analyst->Technology->Flattened View (or Flattened to Gates View), not to hierarchical views that you have flattened (using, for example, HDL Analyst->Flatten Current Schematic).
Display color-coded clock nets	Displays clock nets in the HDL Analyst View with the color green. See *Color-coded Clock Nets* , on page 447.
Dissolve levels	The number of levels to dissolve, during HDL Analyst->Dissolve Instances. See *Dissolve Instances* , on page 421.
Instances added for expansion	The maximum number of instances to add during any operation (such as HDL Analyst->Hierarchical->Expand) that results in a *filtered* schematic. When this limit is reached, you are prompted to continue adding more instances.
Maximum Routable Endpoints	Specifies the maximum number of endpoints for nets, which the synthesis tool routes to their explicit connection endpoints in the design to improve HDL Analyst performance. The default value is set to 2000. You can use this option to change this value. For more information, see *Results of Maximum Routable Endpoints in the HDL Analyst View* , on page 448.

Color-coded Clock Nets

Clock nets are displayed with the color green in the RTL and Technology views.

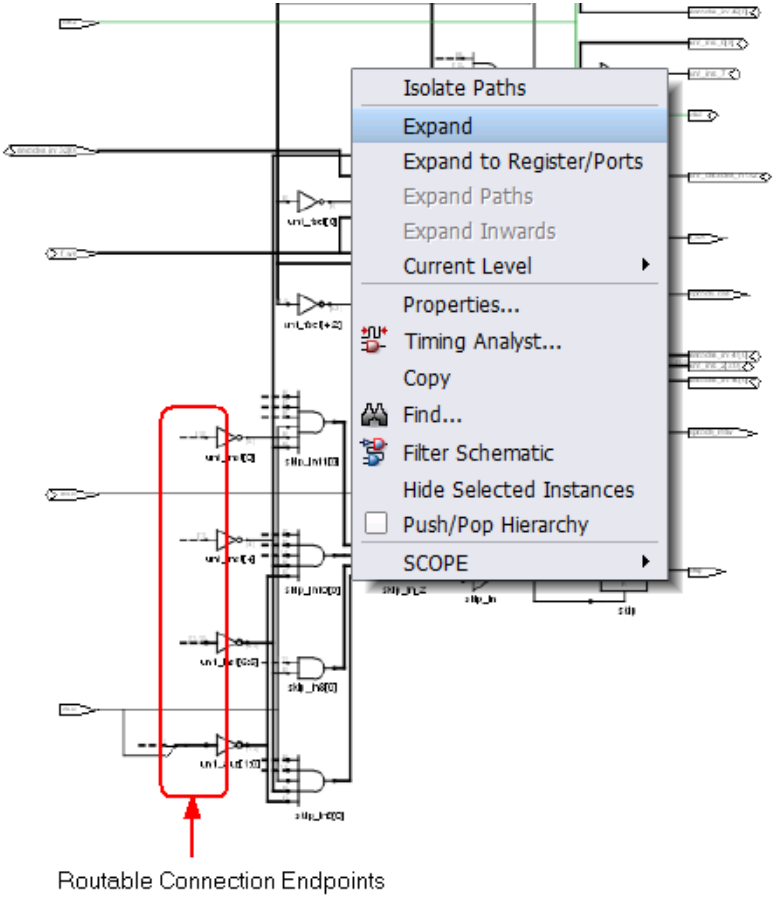
© 2020 Synopsys, Inc.



Results of Maximum Routable Endpoints in the HDL Analyst View

Use the Maximum Routable Endpoints option to specify the maximum number of endpoints for nets in the design to be explicitly routed to their connection endpoints. When you adjust the default value of 2000 sufficiently, improvements in performance can be seen in the HDL Analyst tool.

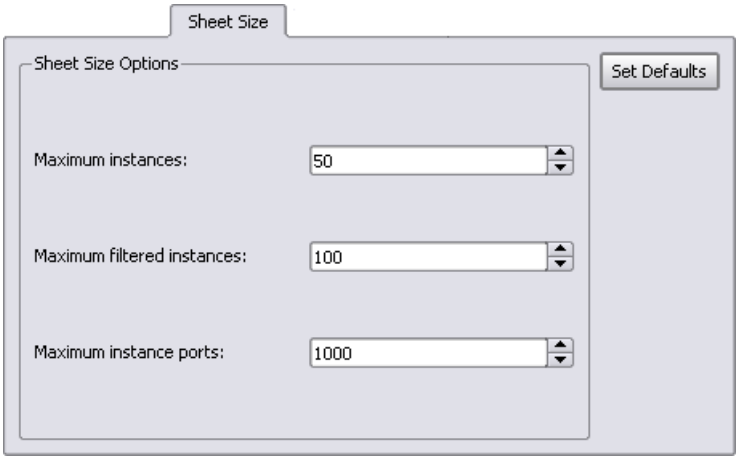
If the number of connection endpoints routed for the design has been reduced, you will see dashes (---) for these endpoints in the HDL Analyst (RTL or Technology) view. Note that you can still select these endpoints and perform any viable operation for these nets as shown in the RTL view below.



Note: Occasionally, the software does not route nets for some reason. You will see dashes (---) for these endpoints in the HDL Analyst (RTL or Technology) view. Note that you can still select these nets and perform any viable operation for them.

© 2020 Synopsys, Inc.

Sheet Size Panel



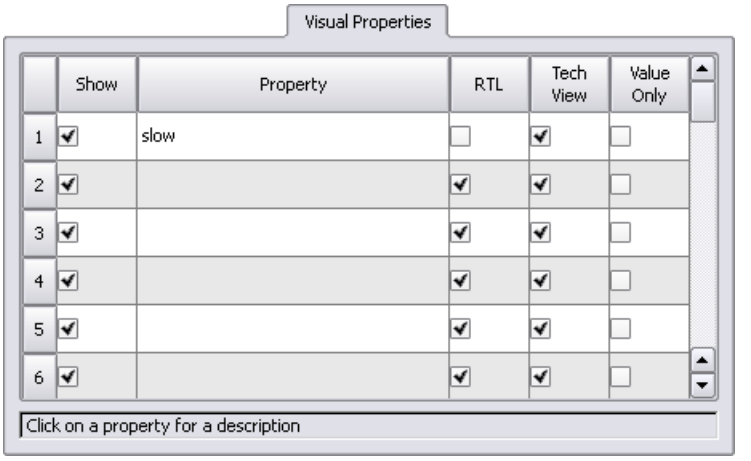
The following options are on the Sheet Size panel.

Maximum instances	Defines the maximum number of instances to display on a single sheet of an unfiltered schematic. If a given hierarchical level has more than this number of instances, then it will be partitioned into multiple sheets. See *Multiple-sheet Schematics* , on page 102.
Maximum filtered instances	Defines the maximum number of instances to display on a filtered schematic sheet, at any visible hierarchical level. This limit is applied recursively, at each visible *level*, when • the sheet itself is a level, and • each transparent instance is a level (even if inside another transparent instance). Whenever a given level has more child instances inside it than the value of Filtered Instances, it is divided into multiple sheets. (Only children are counted, not grandchildren or below. Instance A is a *child* of instance B if it is inside no other instance that is inside B.) In fact, at each level except the sheet itself, an additional margin of allowable child instances is added to the Maximum filtered instances value, increasing its effective value. This means that you can see more child instances than Maximum filtered instances itself implies. The Maximum filtered instances value must be at least the Maximum instances value. See *Multiple-sheet Schematics* , on page 102.
Maximum Instance Ports	Defines the maximum number of instance pins to display on a schematic sheet.

© 2020 Synopsys, Inc.

Visual Properties Panel

Controls the display of the selected property in open HDL Analyst views. The properties are displayed as colored boxes on the relevant objects. To display these properties, the View->Visual Properties command must also be enabled. For more information about properties, see *Viewing Object Properties*, on page 297 in the *User Guide*.

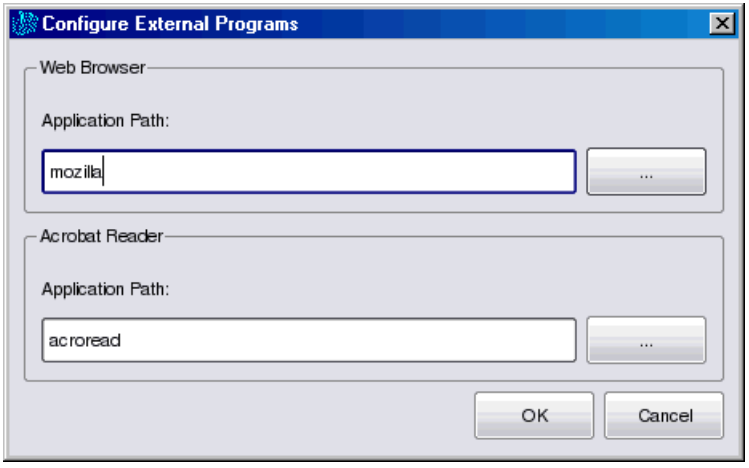


The following options are on the Visual Properties panel.

Show	Toggles the property name and value is displayed in a color-coded box on the object.
Property	Sets the properties to display.
RTL	Enables or disables the display of visual properties in the RTL view.
Tech View	Enables or disables the display of visual properties of in the Technology view.
Value Only	Displays only the value of an item and not its property name.

Configure External Programs Command

This command is for Linux platforms only. It lets you specify the web browser and PDF reader for accessing Synopsys support (see *Web Menu*, on page 454 for details) and online documentation.



Field/Option	Description
Web Browser	Specify your web browser as an absolute path. You can use the Browse button to locate the browser you need. The default is netscape. If your browser requires additional environment settings, you must do so outside the synthesis tool.
Acrobat Reader	Specify your PDF reader as an absolute path. You can use the Browse button to locate the reader you need. The default is acroread.

Synopsys Confidential Information

Web Menu

This menu contains commands that access up-to-date information from Synopsys Support.

Command	Description
Synopsys Home	Opens the Synopsys home web page for Synopsys products.
FPGA Implementation Tools	Opens the Synopsys FPGA design solution web page for Synopsys FPGA products. You can find information about the full line of Synopsys FPGA Implementation products here.

Help Menu

There are four help systems accessible from the Help menu:

- Help on the Synopsys FPGA synthesis tool (Help->Help Topics)
- Help on standard Tcl commands (Help->TCL)
- Help on using messages (Help->Error Messages)
- Help on using online help (Help->How to Use Help)

The following table describes the Help menu commands.

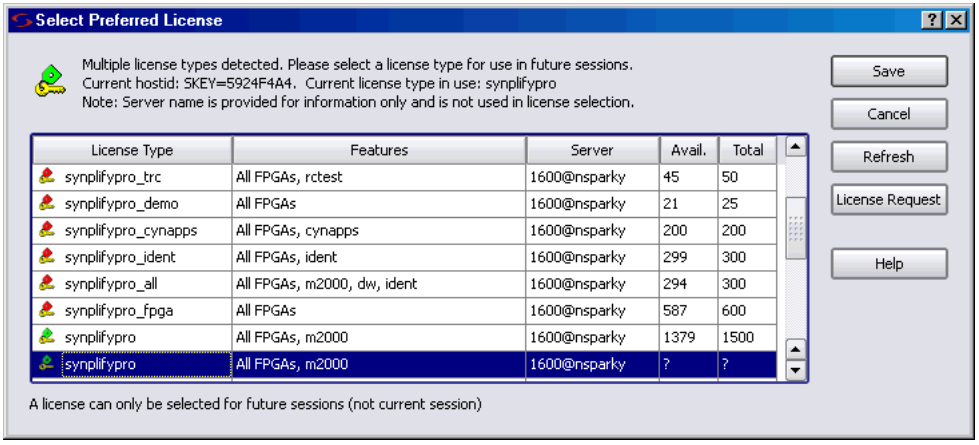
Command	Description
Help Topics	Displays hyperlinked online help for the product.
Additional Products	Displays the Synopsys FPGA family of products with a brief description.
How to Use Help	Displays help on how to use Synopsys FPGA online help.
PDF Documents	Displays an Open dialog box with hyperlinked PDF documentation for the product including user guide and reference manuals. You need Adobe Acrobat Reader® to view the PDF files.
Error Messages	Displays help on the message viewer.
TCL	Displays help for Tcl commands.
Mouse Stroke Tutor	Displays the Mouse Stroke Tutor dialog box which provides information on the available mouse strokes - see *Using Mouse Strokes* , on page 53 for details.
License Agreement	Displays the Synopsys software license agreement.
Floating License Usage	Specifies the number of floating licenses currently being used and their users.

© 2020 Synopsys, Inc.

Command	Description
Preferred License Selection	Displays the floating licenses that are available for your selection. See *Preferred License Selection Command* , on page 456.
Tip of the Day	Displays a daily tip on how to use the Synopsys tools better. See *Tip of the Day Command* , on page 457.
About this program	Displays the About dialog box, showing the tool product name, license expiration date, customer identification number, version number, and copyright. Clicking the Versions button in the About dialog box displays the Version Information dialog box, listing the installation directory and the versions of all the compiler and mapper programs for the tool.

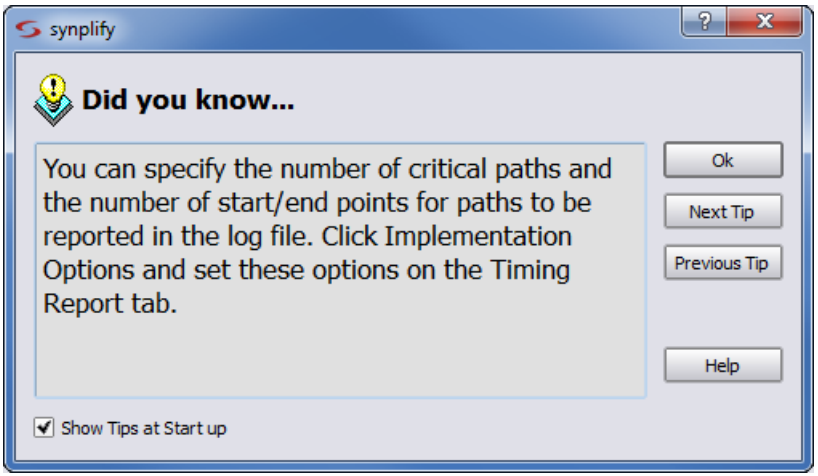
Preferred License Selection Command

Select Help->Preferred License to display the Select Preferred License dialog box, listing the available licenses for you to choose from. Select a license from the License Type column and click Save. Close and restart the Synopsys tool, then the new session uses the preferred license you selected.



Tip of the Day Command

Select Help->Tip of the Day to display a dialog box with a daily tip on how to best use the Synopsys tool. This dialog box also displays automatically when you first start the tool. To prevent it from redisplaying at product startup, deselect Show Tips at Startup.



© 2020 Synopsys, Inc.

SYNOPSYS®
Silicon to Software™

CHAPTER 6

GUI Popup Menu Commands

In addition to the GUI menu commands described in Chapter 5, *User Interface Commands*, the FPGA synthesis tools also have context-sensitive commands that are accessed from popup or right-click menus in different parts of the interface. Most of these commands have an equivalent menu command. This chapter only describes the unique commands that are not documented in the previous chapter.

See the following sections for details:

- Popup Menus, on page 460
- Project View Popup Menus, on page 466
- RTL and Technology Views Popup Menus, on page 483

© 2020 Synopsys, Inc.

Popup Menus

Popup menus, available by clicking the right mouse button, offer quick ways to access commonly used menu commands that are specific to the view where you click. Commands shown greyed out (dimmed) are currently inaccessible. Popup menu commands generally duplicate commands available from the regular menus, but sometimes have commands that are only available from the popup menu. The following table lists the popup menus:

Popup Menu	Description
Project view	See *Project View Popup Menus* , on page 466 for details.
SCOPE window	Contains commonly used commands from the Edit menu.
Watch Window	See *Watch Window Popup Menu* , on page 460 for details.
Tcl window	Contains commands from the Edit menu. For details, see *Edit Menu Commands for the Text Editor* , on page 312.
Text Editor window	See *Text Editor Popup Menu* , on page 461 for more information.
RTL and Technology views	See *RTL and Technology Views Popup Menus* , on page 483.
FSM viewer	See *FSM Viewer Popup Menu* , on page 463.

Watch Window Popup Menu

The Watch window popup menu contains the following commands:

Command	Description
Configure Watch	Displays the Log Watch Configuration dialog box, where you choose the implementations to watch.
Refresh	Refreshes (updates) the window display.
Clear Parameters	Empties the Watch window.

For more information on the Watch window and the Configure Watch dialog box, see *Watch Window*, on page 37.

Tcl Window Popup Menu

The Tcl window popup menu contains the Copy, Paste, and Find commands from the Edit menu, as well as the Clear command, which empties the Tcl window. For information on the Edit menu commands available in the Tcl window, see *Edit Menu Commands for the Text Editor*, on page 312.

Text Editor Popup Menu

The popup menu in the Text Editor window contains the following commonly used text-editing commands from the Edit menu: Undo, Redo, Cut, Copy, Paste, and Toggle Bookmark. In addition, HDL Analyst specific commands appear when both an HDL Analyst view and it's corresponding HDL source file is open. For details of these commands, see *Edit Menu Commands for the Text Editor*, on page 312 and *HDL Analyst Menu*, on page 412.

The following table lists the commands that are unique to the popup menu:

Command	Description
Filter Analyst	Filters your design to show only the currently selected objects in the HDL text file. This is the same as HDL Analyst->Filter Schematic.
Select in Analyst	Crossprobes from the Text Editor and selects the objects in the HDL Analyst view. To use this command, the Enhanced Text Crossprobing (option must be engaged.

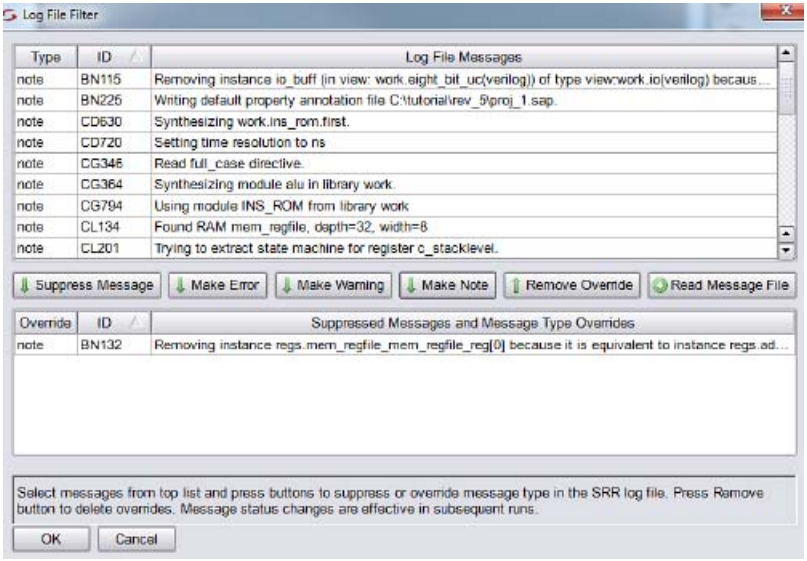
Log File Popup Menu

The popup menu in the log file contains commands that control operations in the log file. The popup menu differs when the log file is opened in the HTML mode or in the ASCII text mode.

Log File Filter Dialog Box

The Log File Filter dialog box is available by selecting Log File Message Filter from the log file popup menu when the log file is opened in the HDML mode. The dialog box allows messages in the current session to be promoted or demoted in severity or suppressed from the log files for subsequent sessions. For additional information on using this dialog box, see *Log File Message Controls*, on page 213 of the *User Guide*.

© 2020 Synopsys, Inc.
Synopsys Confidential Information



The following table describes the dialog box functions.

Function	Description
Log File Messages window	Displays the message ID and text and the default message type of messages generated during the current session.
Suppress Message button	Suppresses the selected note, warning, or advisory message. The selected message is removed from the upper Log File Messages window and displayed in the lower window with the Override column indicating suppress status. Note that error messages cannot be suppressed.
Make Error button	Promotes the status of the selected warning (or note) to an error. The selected message is removed from the upper Log File Messages window and displayed in the lower window with the Override column indicating error status.
Make Warning button	Promotes the status of the selected note to a warning. The selected message is removed from the upper Log File Messages window and displayed in the lower window with the Override column indicating warning status.

Function	Description
Make Note button	Demotes the status of the selected warning to a note. The selected message is removed from the upper Log File Messages window and displayed in the lower window with the Override column indicating note status.
Remove Override button	Removes the override status on the selected message in the lower window and returns the message to the upper Log File Messages window.
Read Message File	Select the message filter file (*project*.pfl) to be read for the project.
lower window	Lists the status of all messages that have been promoted, demoted, or suppressed.
OK button	Updates the status of any changed messages in the .pfl file. Note that you must recompile/resynthesize the design before any message status changes become effective.

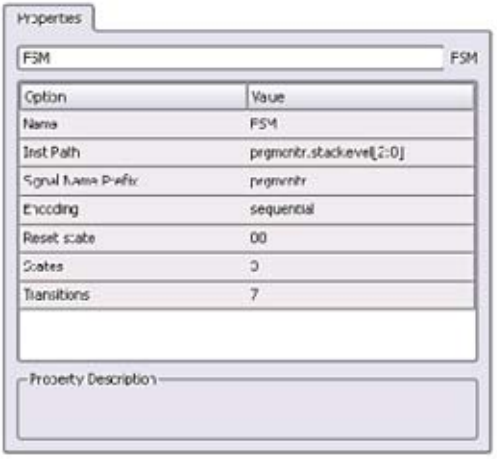
FSM Viewer Popup Menu

The popup menu in the FSM Viewer contains commands that determine what is shown in the FSM Viewer. The following table lists the popup commands in the FSM Viewer.

Command	Description
Properties	Displays the Object Properties dialog box and view properties of a selected state or transition. Information about a selected transition includes the conditions enabling the transition and the identities of its origin and destination states. Information about a selected state includes its name, RTL encoding, and mapped encoding.
Filter	See *View Menu: FSM Viewer Commands* , on page 327.
Unfilter	See *View Menu: FSM Viewer Commands* , on page 327.
FSM Properties	Displays the Object Properties dialog box indicating the FSM identity and location, encoding style, reset state, and the number of states and transitions.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

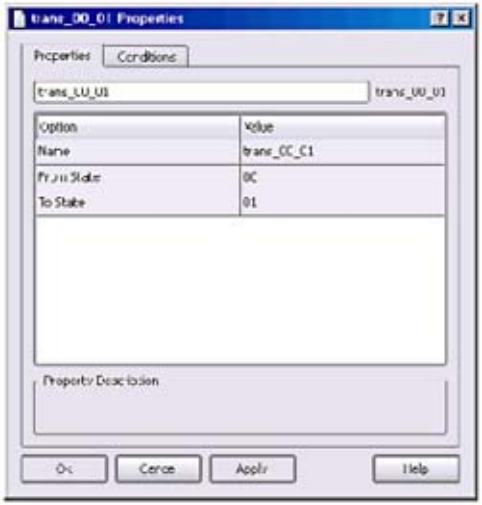
FSM Properties



State properties
(state selected)



Transition properties
(transition selected)



Field/Option	**Description**
	Icons indicating the object type: FSM, state, or transition.
Name	The name of the selected state or transition, or FSM if nothing is selected.
Inst Path	The full name and position of the state machine in the hierarchy.
Signal Name Prefix	The position of the state machine in the hierarchy.
Encoding	The style of encoding used for the state machine. This can be onehot, sequential, gray, or safe. See *syn_encoding* , on page 77, for information on changing the encoding type.
Reset State	The initial state of the FSM: the active state after resetting.
States	The number of states in the state machine.
Transitions	The number of transitions in the state machine.
RTL Encoding	The name (address) of the selected state, as referred to in the RTL (HDL) file.
Mapped Encoding	The encoding of the selected state.
From	The origin state of the selected transition.
To	The destination state of the selected transition.
Conditions (min-terms)	The conditions enabling the selected transition, as defined in the RTL (HDL) file.

© 2020 Synopsys, Inc.
Synopsys Confidential Information

Project View Popup Menus

The popup menu commands available in the Project view are context-sensitive, depending on what is currently selected and where in the view you click to open the popup menu. Most commands duplicate commands from the File, Project, Run, and Options menus.

Project Management View Popup Commands

The following table describes the Project Management view commands that are not duplicated on other menus in the tool:

Command	Description
Project Management View, No Selections	
Open Project	Displays the Open Project Dialog. See *Open Project Command* , on page 311.
New Project	Creates a new empty project in the Project Window.
Refresh	Refreshes the display.
Project View Options	Displays the Project View Options dialog. See *Project View Options Command* , on page 429.
Project Selected	
Open as Text	Opens the selected file in the Text Editor.
Add File	Displays the Add Files to Project dialog. See *Add Source File Command* , on page 334.
Synthesize	Compiles and maps the selected design.
Compile Only	Compiles the selected design.
Write Output Netlist Only	Writes the mapped output netlist to structural Verilog (.vm) or VHDL (.vhm) format. Same as enabling: • Write Mapped Verilog Netlist • Write Mapped VDHL Netlist on the Implementation Results tab of the Implementation Options dialog box.
Arrange VHDL Files	Reorders the VHDL source files.

© 2020 Synopsys, Inc.

Command	Description
Save Project	Displays the Save Project As dialog box.
Close Project	Closes your project.
Project Folder or File Selected	
Add Folder	Creates a folder with the new name you specified and adds it to the Project view. See *Add Folder Command* , on page 470.
Rename Folder	Renames an existing folder with the new name you specified in the Project view. See *Rename Folder Command* , on page 470.
Delete Folder	Deletes the specified folder and all its contents as necessary. See *Delete Folder Command* , on page 471.
Remove from Folder	Removes the selected file from its corresponding folder.
Place in Folder	Places the selected file into the folder you specify.
Launch Tools->Run Vendor Tool	Launches the vendor application or Tcl procedure tool from the Project view for the selected file of folder. See *Vendor Tool Invocation Popup Menu Command* , on page 472.
Constraint File Selected	
File Options	Displays the File Options dialog box. See *File Options Popup Menu Command* , on page 473.
Open	Opens the SCOPE window.
Open as Text	Opens the selected file in the Text Editor.
Copy File	Displays the Copy File dialog box, where you copy the selected file and add it to the current project. You specify a new name for the file. See *Copy File Popup Menu Command* , on page 475.
Change File	Opens the Source File dialog box where you choose a new file to replace the selected file. See *Change File Command* , on page 337.
Remove File From Project	Removes the file from the project.
HDL File Selected	
File Options	Displays the File Options dialog box. See *File Options Popup Menu Command* , on page 473.

Synopsys Confidential Information

Command	Description
Open	Opens the file in the Text Editor.
Syntax Check	Runs a syntax check on your design code. Reports errors, warnings, or notes in the Tcl Window.
Synthesis Check	Runs a synthesis check on your design code. This includes a syntax check and a check to see if the synthesis tool could map the design to the hardware. No optimizations are performed. Reports errors, warnings, or notes in the Tcl Window.
Copy File	Displays the Copy File dialog box, where you copy the selected file and add it to the current project. You specify a new name for the file. See *Copy File Popup Menu Command* , on page 475.
Change File	Opens the Source File dialog box where you choose a new file to replace the selected file. See *Change File Command* , on page 337.
Remove File From Project	Removes the file from the project.
Implementation Selected	
Implementation Options	Displays the Implementation Options dialog box. See *Implementation Options Command* , on page 346.
Change Implementation Name	Displays the Implementation Name dialog box, where you rename the selected implementation. See *Change Implementation Popup Menu Commands* , on page 476.
Copy Implementation	Copies the selected implementation and adds it to the current project with the name you specify in the dialog box. See *Change Implementation Popup Menu Commands* , on page 476.
Remove Implementation	Removes the selected implementation from the project.
RTL View	Creates an RTL View based on the properties of the selected implementation.
Tech View	Creates a Technology View based on the properties of the selected implementation.
Add P&R Implementation	Displays the Add New Place & Route Task dialog box where you set options to run place & route after synthesis. See *Add P&R Implementation Popup Menu Command* , on page 478

Command	Description
Run	Starts a synthesis run on your design.
Identify Implementation Selected	
Identify Instrumentor	Displays the Identify Instrumentor tool, so that you can instrument signals based on the RTL and SRS netlist (after compile). See *Working with the Identify Tools* , on page 556.
Launch Identify Debugger	Launches the Identify Debugger tool to debug the instrumented design. See *Working with the Identify Tools* , on page 556.
Place & Route Implementation Selected	
Add Place & Route Job	Displays the Add New Place & Route Task dialog box, so you can set options and run placement and routing. See *Add P&R Implementation Popup Menu Command* , on page 478.
Remove Place & Route Job	Deletes the place-and-route implementation from the project.
Run Place & Route Job	Runs the place-and-route job for the design.

Project Management Commands

The following table lists the popup commands in the Project Management views that are not available on the tool command menus. The Project Management view consists of two tabs, and the table lists the popup commands available in both tabs.

For the Design Hierarchy tab, the tools support all the project management commands listed in *Project Management View Popup Commands*, on page 466, as well as the unique commands listed here.

Command	Description
Project Management View -> Project FilesTab Popup Commands	
All Synplify project management commands	Refer to the table in *Project Management View Popup Commands* , on page 466 for descriptions of these commands.

Synopsys Confidential Information

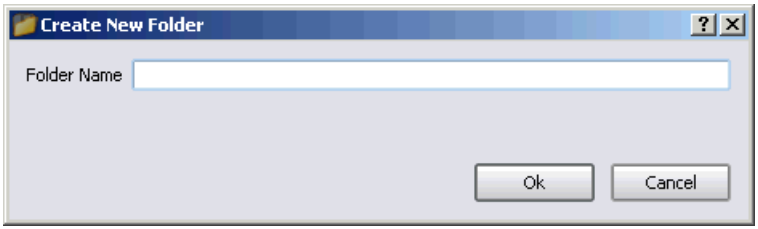
Command	Description
Project Options	With the project selected, displays project properties such as name and location. See *Project Options Popup Menu Command* , on page 477.
Show Compile Points	Displays the compile points of the selected implementation and lets you edit them. See *Show Compile Points Popup Menu Command* , on page 476.
P & R Options	With a place-and-route implementation selected, displays the Options for Place & Route on Implementation dialog box, so you can change options and rerun placement and routing. See *Options for Place & Route Jobs Popup Menu Command* , on page 480 for a description of the features.

Project Management View Popup Folder Commands

The Project view popup menu includes commands for manipulating folders.

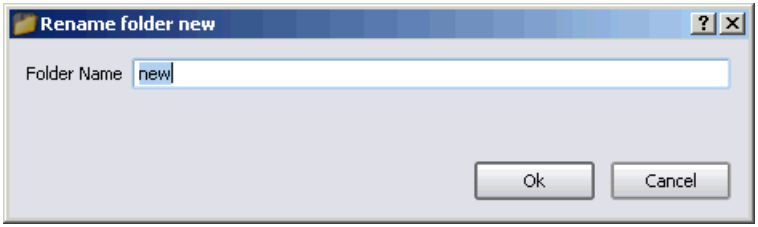
Add Folder Command

Use this option to add a folder to the Project view.



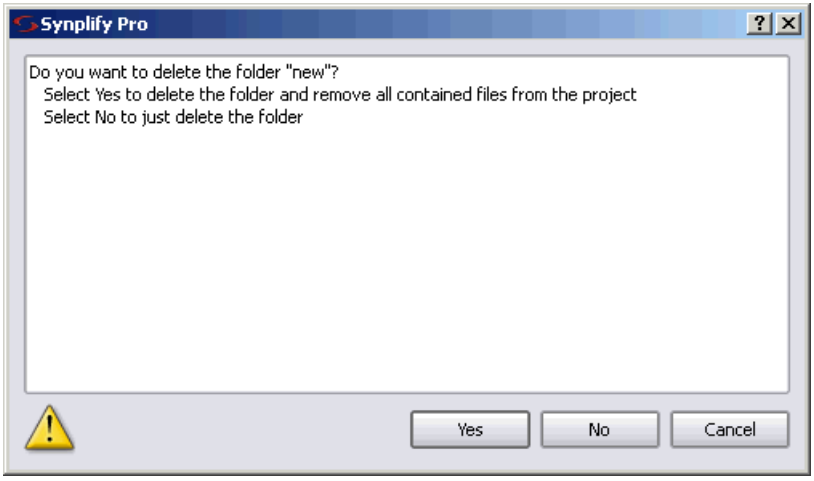
Rename Folder Command

Use this option to rename an existing folder in the Project view.



Delete Folder Command

Use this option to delete a folder from the Project view.



This dialog box includes the following options:

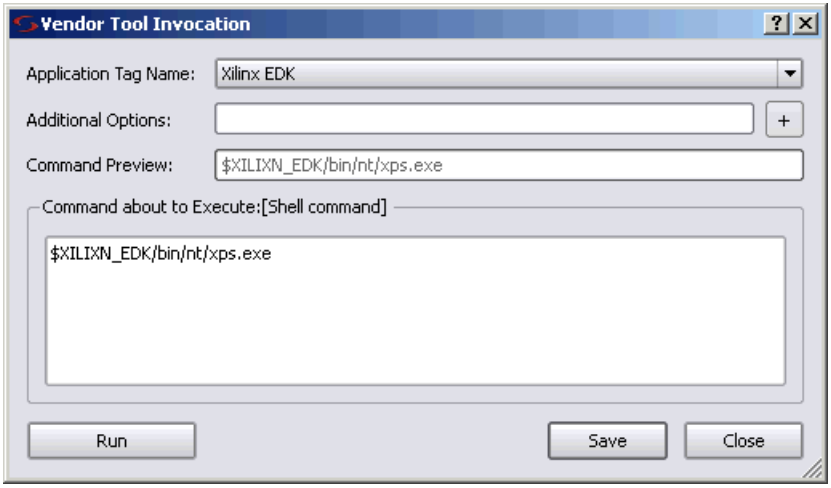
Feature	Description
Yes	Select Yes to delete the folder and all files contained in the folder from the Project view.
No	Select No to delete just the folder from the Project view.
Cancel	Select Cancel, to discontinue the operation.

Synopsys Confidential Information

Vendor Tool Invocation Popup Menu Command

Use the Vendor Tool Invocation command to invoke third-party tools, such as the Embedded Development Kit (EDK) from within the Synopsys FPGA products. This allows you to modify source files or libraries added to your synthesis projects from within the third-party tool directly. Use the following dialog box to run the vendor tools.

For more information, see *Invoking Third-Party Vendor Tools*, on page 530 in the *User Guide*.



The Vendor Tool Invocation dialog box includes the following options:

Feature	Description
Application Tag Name	Specifies an application or Tcl procedure name. Type in the name or select a preconfigured application from the list.
Additional Options	Defines any additional arguments for the Tcl procedure or third-party application. You can select arguments from the drop-down list or type them. Note: For internal Synopsys tools, you must include the $Syncode parameter.

Feature	Description
Command Preview	Sets up the direct invocation of a third-party tool from within the FPGA synthesis tool, using the path defined for the executable in Application Name with Path or to execute the Tcl procedure from within the FPGA synthesis tool, using the path defined for the procedure in Tcl Procedure Name.
Run	The synthesis tool launches the third-party tool or runs the Tcl procedure with the arguments you specified.

File Options Popup Menu Command

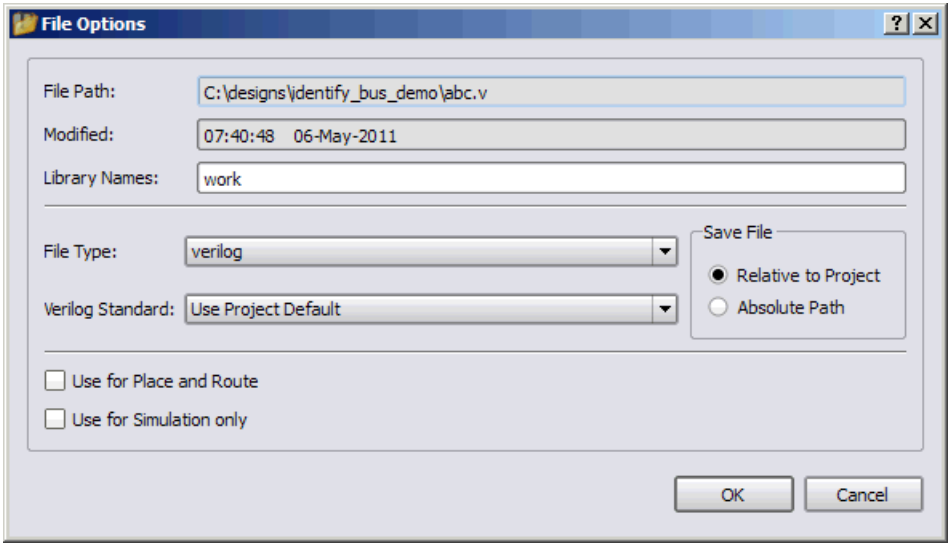
To display the File Options dialog box, right-click on a project file and select File Options from the popup menu. Specify the path as relative or absolute when listing the file in the project (`prj`) file and if the file is to be passed to the place-and-route tool or used only for simulation.

Field/Option	Description
File Path	Path to the selected file.
File Type	The folder type for the selected file. You can select the file folder type from a large list of file types. Changing the folder file type does *not* change the file contents or its extension; it simply places the file in the specified Project view folder. For example, if you change the file type of a VHDL file to Verilog, the file retains its Verilog extension, but is moved from the VHDL folder to the Verilog folder.
Library Names	Name of the library which must be compatible with the HDL simulator. For VHDL files, the dialog box is the same as that accessed by Project->Set VHDL Library - see *Set VHDL Library Command* , on page 337.
Last modified	Date the file was last modified.
Save file	The format for the path type: choose either Relative to Project (the default) or with an Absolute Path.

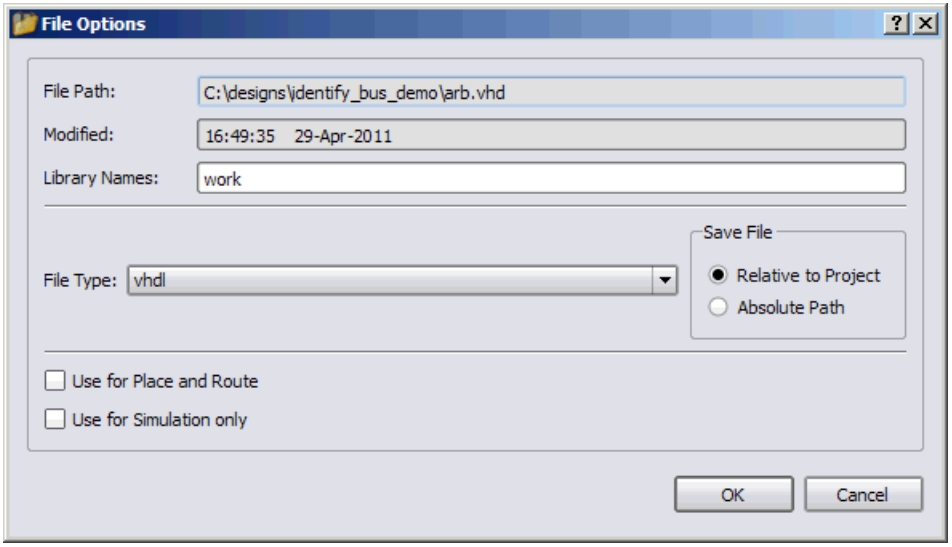
© 2020 Synopsys, Inc.

Field/Option	Description
Verilog Standard (Verilog only)	Select the Verilog file type from the menu: Use Project Default, Verilog 95, Verilog 2001, or SystemVerilog. Use Project Default sets the type of the selected file to the default for the project (new projects default to SystemVerilog).
Use for Place and Route	Determines if files are automatically passed to the backend place-and-route tool. The files are copied to the place-and-route implementation directory and then invoked when the place-and-route tool is run.
Use for Simulation Only	Determines if files are only to be used for simulation. For example, files such as test benches containing HDL constructs used only for simulation can be specified using this option.

The following is the Verilog dialog box:

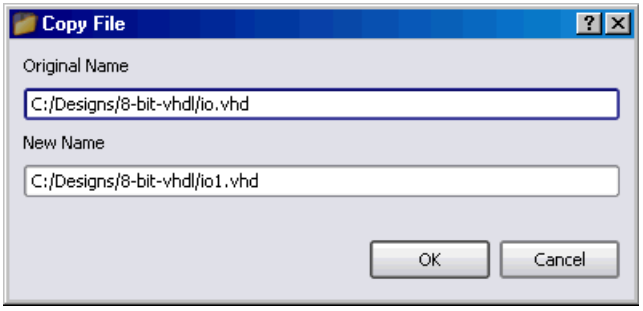


The following is the VHDL dialog box:



Copy File Popup Menu Command

With a file selected, select the Copy File popup menu command to copy the selected file and add it to the current project. This displays the Copy File dialog box where you specify the name of the new file.

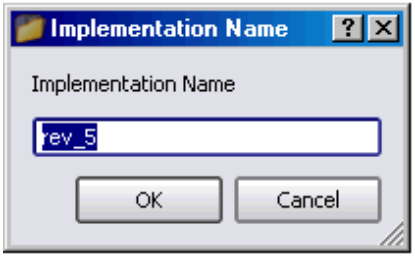


Synopsys Confidential Information

Change Implementation Popup Menu Commands

With an implementation selected, right-click and select the Change Implementation Name or Copy Implementation popup menu commands to display a dialog box where you specify the new name.

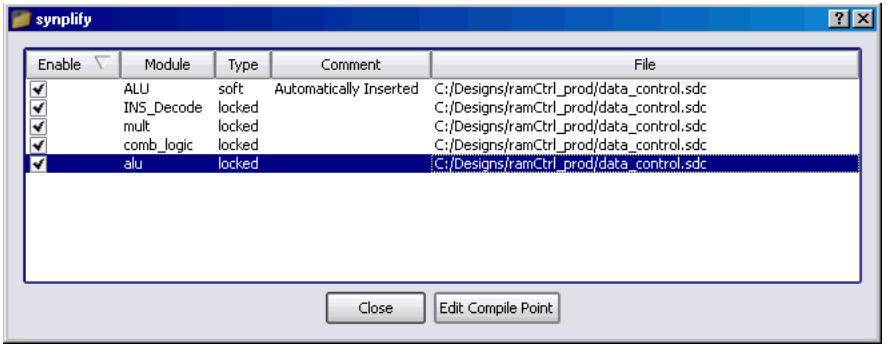
Command	Description
Change Implementation Name	The implementation name you specify is the new name for the implementation.
Copy Implementation	The currently selected implementation is copied and saved to the project with the new implementation name you specify.



Show Compile Points Popup Menu Command

With an implementation selected, select the Show Compile Points popup menu command to display the Compile Points dialog box and view or edit the compile points of the selected implementation.

Compile points are only available for certain technologies. For more information on compile points and the compile-point synthesis flow, see *Compile Point Types*, on page 440 and *Synthesizing Compile Points*, on page 455 of the *User Guide*.



The columns Enable, Module, Type, and Comment in the dialog box correspond to the columns Enabled, Module, Type, and Comment in the SCOPE spreadsheet for the compile point. The File column lists the top-level constraint file where the compile point is defined.

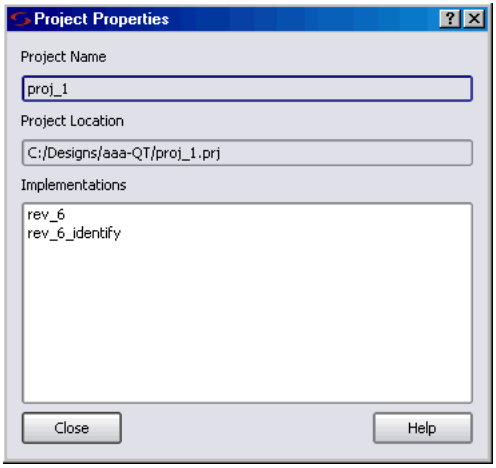
To open and edit the SCOPE spreadsheet for a compile point, either double-click the row of the compile point or select it and click the Edit Compile Point button.

Project Options Popup Menu Command

With a project selected, select the Project Options popup menu command to display the Project Properties dialog box and change the implementation of a project.

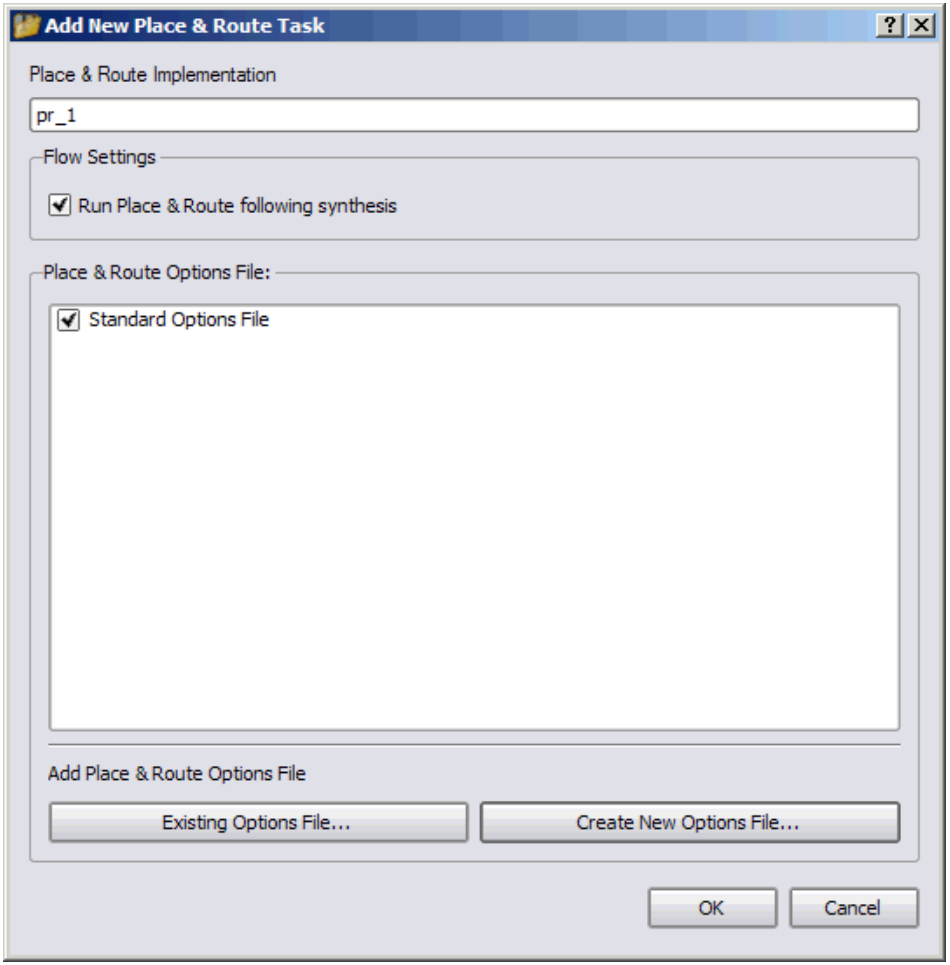
In the dialog box, select an implementation in the Implementations list, then click OK or Apply to make it the active implementation of the project.

© 2020 Synopsys, Inc.
Synopsys Confidential Information



Add P&R Implementation Popup Menu Command

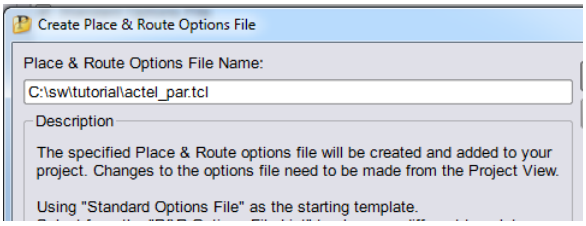
Displays the Add New Place & Route Task dialog box. For information about using this command for place-and-route encapsulation, see *Running P&R Automatically after Synthesis*, on page 554 in the *User Guide*.



Command	Description
Place & Route Implementation Name	Enter a name for the place & route implementation. Do not use spaces for the implementation name.
Flow Settings	
Run Place & Route following synthesis	Enable/disable the running of the place & route tool from the synthesis tool immediately following synthesis.

© 2020 Synopsys, Inc.

Command	Description
Add Place & Route Options File Existing Options File	This option opens the Select Place & Route option file dialog box where you browse for an existing place & route options file. See *Running P&R Automatically after Synthesis* , on page 554 for information about using this feature.
Add Place & Route Options File Create New Options File	This option opens the Create Place & Route Options File dialog box where you specify a new place & route options file. See *Running P&R Automatically after Synthesis* , on page 554 for information about creating a new options file.

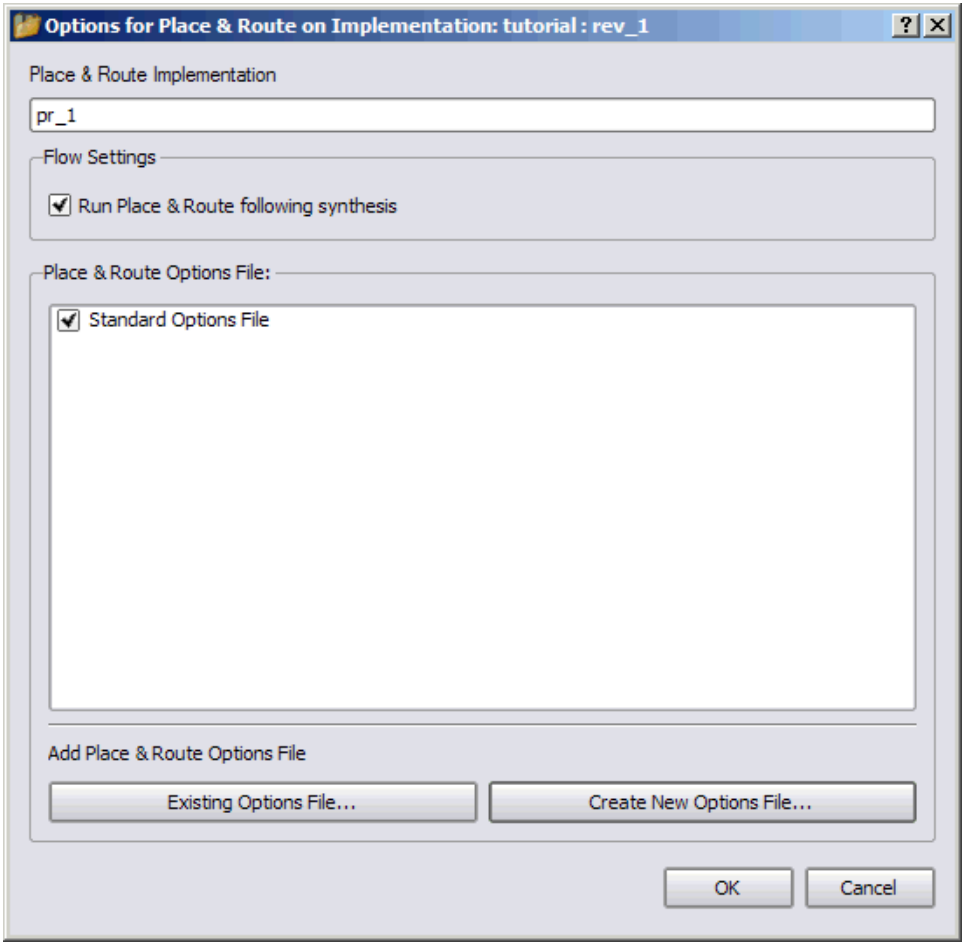


Once the par implementation is created, then you can right-click and perform any of the following options:

- P&R Options—See Options for Place & Route Jobs Popup Menu Command, on page 480.
- Add Place & Route Job—See Add P&R Implementation Popup Menu Command, on page 478.
- Run Place & Route Job—Runs the place-and-route job.

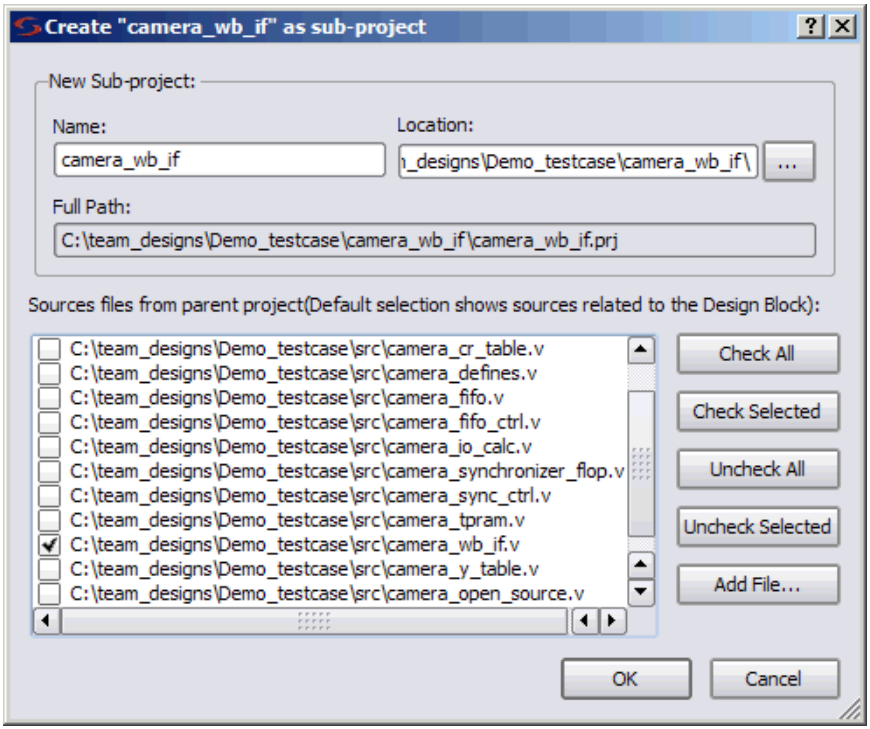
Options for Place & Route Jobs Popup Menu Command

You can select a place-and-route job for a particular implementation, easily change options and then rerun the job. These options are the same found on the Options for Place & Route on Implementation dialog box. For a description of these options, see *Add P&R Implementation Popup Menu Command*, on page 478.



Synopsys Confidential Information

in the *User Guide*



RTL and Technology Views Popup Menus

Some commands are only available from the popup menus in the RTL and Technology views, but most of the commands are duplicates of commands from the HDL Analyst, Edit, and View menus. The popup menus in the RTL and Technology views are nearly identical. See the following:

- Hierarchy Browser Popup Menu Commands, on page 483
- RTL View and Technology View Popup Menu Commands, on page 483

Hierarchy Browser Popup Menu Commands

The following commands become available when you right-click in the Hierarchy Browser of an RTL or Technology view. The Filter, Hide Instances, and Unhide Instances commands are the same as the corresponding commands in the HDL Analyst menu. The following commands are unique to this popup menu.

Command	Description
Collapse All	Collapses all trees in the Hierarchy Browser.
Filter	Highlights and filters objects such as ports, instances, and primitives in the HDL analyst window.
Reload	Refreshes the Hierarchy Browser. Use this if the Hierarchy Browser and schematic view do not match.
Hide/Unhide Instances	Hides or unhides selected instances in the HDL analyst window. For more information on hidden instances, see *Hidden Hierarchical Instances* , on page 93.

RTL View and Technology View Popup Menu Commands

The commands on the popup menu are context-sensitive, and vary depending on the object selected, the kind of view, and where you click. In general, if you have a selected object and you right-click in the background, the menu includes global commands as well as selection-specific commands for the objects.

Most of the commands duplicate commands available on the HDL Analyst menu (see *HDL Analyst Menu*, on page 412). The following table lists the unique commands.

Synopsys Confidential Information

Common Commands

Command	See ...
Show Critical Path	HDL Analyst Menu: Timing Commands, on page 419.
Timing Analyst	HDL Analyst Menu: Timing Commands, on page 419.
Find	Find Command (HDL Analyst), on page 317.
Filter Schematic	HDL Analyst Menu: Filtering and Flattening Commands, on page 415.
Push/Pop Hierarchy	HDL Analyst Menu->RTL and Technology View Submenus, on page 412.
Select All Schematic	HDL Analyst Menu: Selection Commands, on page 423.
Select All Sheet	HDL Analyst Menu: Selection Commands, on page 423.
Unselect All	HDL Analyst Menu: Selection Commands, on page 423.
Flatten Schematic	HDL Analyst Menu: Filtering and Flattening Commands, on page 415.
Unflatten Current Schematic	HDL Analyst Menu: Filtering and Flattening Commands, on page 415.
HDL Analyst Options	HDL Analyst Options Command, on page 443.
SCOPE->Edit Attributes (object *name*)	Opens a SCOPE window where you can enter attributes for the selected object. It displays the Select Constraint File dialog box (Edit Attributes Popup Menu Command, on page 487), where you select the constraint file to edit. If no constraint file exists, you are prompted to create one.

SCOPE->Edit Compile Point Constraints (module *moduleName*)	For technologies that support compile points, it opens a SCOPE window where you can enter constraints for the selected compile point. It displays the Select Compile Point Definition File dialog box and lets you create or edit a compile-point constraint file for the selected region or instance. See Edit Attributes Popup Menu Command, on page 487.
SCOPE->Edit Module Constraints (module *moduleName*)	Opens a SCOPE window so you can define module constraints for the selected module). If you do not have a constraint file, it prompts you to create one. The file created is a separate, module-level constraint file.

Instance Selected

Command	**See ...**
Isolate Paths	Isolate Paths , on page 420.
Expand Paths	Hierarchical->Expand Paths , on page 414.
Current Level Expand Paths	Current Level->Expand Paths , on page 415.
Show Context	Show Context , on page 420.
Hide Instance	Hide Instances , on page 420.
Unhide Instance	Unhide Instances , on page 420.
Show All Hier Pins	Show All Hier Pins , on page 421.
Dissolve Instance	Dissolve Instances , on page 421.
Dissolve to Gates	Dissolve to Gates , on page 421.

Port Selected

Command	**See ...**
Expand to Register/Port	Hierarchical->Expand to Register/Port , on page 414.
Expand Inwards	Hierarchical->Expand Inwards , on page 414.
Current Level->Expand	Current Level->Expand , on page 414.
Current Level->Expand to Register/Port	Current Level->Expand to Register/Port , on page 415.
Current Level->Expand Paths	Current Level->Expand Paths , on page 415.

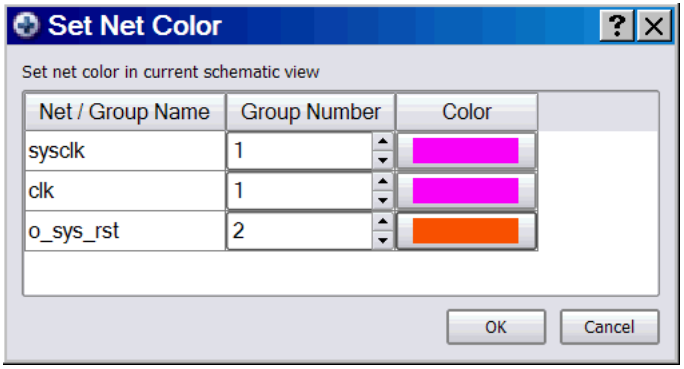
Properties	Properties Popup Menu Command, on page 487.

Net Selected

Command	See ...
Goto Net Driver	Hierarchical->Goto Net Driver , on page 414.
Select Net Driver	Hierarchical->Select Net Driver , on page 414.
Select Net Instances	Hierarchical->Select Net Instances , on page 414.
Current Level->Goto Net Driver	Current Level->Goto Net Driver , on page 415.
Current Level->Select Net Driver	Current Level->Select Net Driver , on page 415.
Current Level->Select Net Instances	Current Level->Select Net Instances , on page 415.
Set Net Color	Sets the color of the selected net from a color pallet. For details, see Set Net Color Popup Menu Command, on page 486.

Set Net Color Popup Menu Command

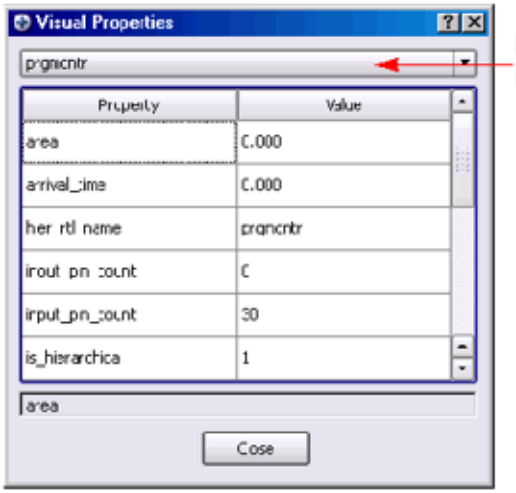
The set net color command sets the color of the selected net in the HDL Analyst for the current session. To use the command, select the desired net or nets in the RTL view and select set net color from the popup menu to display the dialog box.



Double click the corresponding color in the Color column to display the color pallet and then double click the desired color and click OK. Nets can be grouped and assigned to the same color by selecting the same group number in the Group Number column.

Properties Popup Menu Command

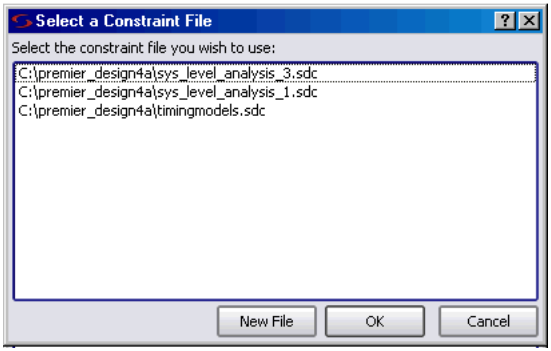
The software displays property information about the selected object when you right-click on a net, instance, pin, or port in an HDL Analyst view. See *Visual Properties Panel*, on page 452 or Viewing Object Properties, on page 297 in the *User Guide* for more information about viewing object properties.

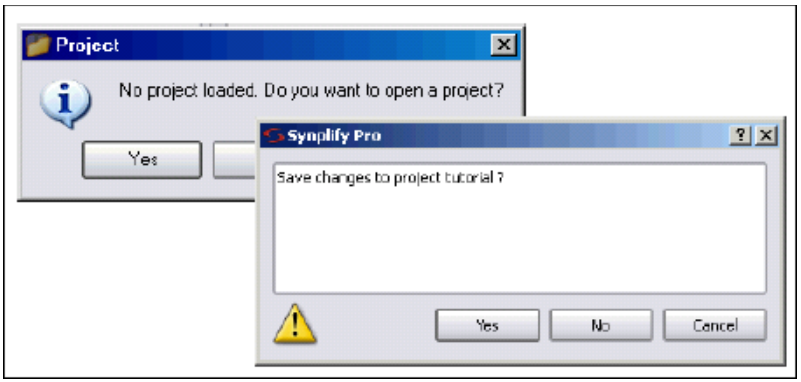


Edit Attributes Popup Menu Command

You use the Select a Constraint File dialog box to choose or create a constraint file. You can open the constraint file and edit it. For technologies that support the compile points, it lets you create or edit a compile-point constraint file for the selected region or instance.

Synopsys Confidential Information


Select a Constraint File
Select the constraint file you wish to use:
C:\premier_design4a\sys_level_analysis_3.sdc
C:\premier_design4a\sys_level_analysis_1.sdc
C:\premier_design4a\timingmodels.sdc
New File
OK
Cancel


Project
No project loaded. Do you want to open a project?
Yes
Synplify Pro
Save changes to project tutorial ?
Yes
No
Cancel

Index

Symbols

__ALLOWNESTEDBLOCKCOMMENTSTART__ directive 375
SEARCHFILENAMEONLY directive 373
! character, find command 159
? wildcard
  Timing Analyzer 410
.srr file
  See log file
.srs file
  See srs file

Numerics

64-bit mapping 350

A

aborting a synthesis run 387
About this program command 456
add files
  -_include tcl argument 24
Add Implementation command 333
Add P&R Implementation command 478
Add Place & Route Options File command 480
Add Place and Route Job command 478
Add Source File command 333
add_file Tcl command 23
add_folder Tcl command 27
add_to_collection command 204
Additional Products command 455
annotated properties for analyst
  object properties for filtering 155
append_to_collection command 206
archive utility
  SEARCHFILENAMEONLY directive 373
  copy tcl command 85
  unarchive tcl command 85
Arrange VHDL files command 382
asynchronous clock report generation option 402
Attributes panel, SCOPE 235
auto constraints
  Maximize option (Constraints tab) 352

B

Back command 327
batch mode 141
Build Project command 306
bus bundling 447
bus_dimension_separator_style command 300
bus_naming_style command 300
buses
  compressed display 447
  enabling bit range display 445
  hiding in flattened Technology views 447
By any transition command 328
By input transitions command 327
By output transitions command 327

C

c_diff command (collections) 227
c_intersect command (collections) 227
c_print command (collections) 227
c_sub command (collections) 227
c_symdiff command (collections) 227
c_symdiff command, examples 169
c_union command (collections) 227

Synopsys Confidential Information
© 2020 Synopsys, Inc.

camera mouse pointer 306
case sensitivity, Tcl find command 150
cell interior display, enabling/disabling 446
Change File command 333
Change Implementation Name command 468
check_fdc_query command 32
check_fdc_query Tcl command 32
Clear Parameters command 460
clock alias 406
clock as object 406
clock groups, SCOPE 219
clock paths, ignoring 248
Clocks panel, SCOPE 218
Close command 306
Close Project command 307
Collapse All command 483
collection commands
  c_diff 165
  c_intersect 166
  c_list 167
  c_print 168
  c_symdiff 169
  c_union 170
  SCOPE 226
collections
  Synopsys standard commands 204
Collections panel, SCOPE 225
commands
  Add Place & Route Job 478
  Hierarchy Browser 483
  menu
    See individual command entries
  set_modules (Tcl) 172
  Tcl
    See Tcl commands
  Tcl collection 164
  Tcl command equivalents 14
  Tcl expand 161
  Tcl find 147
Comment Code command 313
Compile Only command 381
compile point constraints
  editing 485
Compile Points panel, SCOPE 237
compiler directives
  __ALLOWNESTEDBLOCKCOMMENTS TART__ 375
  __SYN_COMPATIBLE_INCLUDEPATH_ _ 376
  _BETA_FEATURES_ON_ 372
  SEARCHFILENAMEONLY 373
  IGNORE_VERILOG_BLACKBOX_GUTS 371
  UI option 364
  Verilog 369
Configure External Programs command 453
Configure Mapper Parallel Job command 424
Configure Verilog Compiler command 424
Configure VHDL Compiler command 424
Configure Watch command 460
connectivity, enabling bit range display 445
constraint checker
  check_fdc_query command 32
constraint file
  define_compile_point 302
  define_current_design 303
constraint files
  editing compile point files 485
  SCOPE spreadsheet 216
constraint_file Tcl command 37
constraints
  automatic. *See* auto constraints
  check constraints 382
  FPGA timing 262
Constraints panel
  Implementation Options dialog box 351
context-sensitive popup menus
  See popup menus
Continue on Error
  Configure Compile Point Process 427
Continue on Error compile point option 427
Copy command 312

Copy File command 467
Copy Implementation command 468
copy_collection command 207
copying image
  Create Image command 306
Create Image command 306
Create Place & Route Options file dialog box 480
create_clock timing constraint 263
create_generated_clock timing constraint 265
critical paths
  creating new schematics 413
  custom timing reports 401
  finding 419
  Timing Report panel, Implementation Options dialog box 356
custom folders
  project_folder Tcl command 96
Customize command 424
customizing
  project files 431
Cut command 312

D

define_compile_point
  Tcl 302
define_current_design
  Tcl 303
defining I/O standards 236
delay paths
  POS 254
Delay Paths panel, SCOPE 232
Delete all bookmarks command 312
design parameters (Verilog)
  extracting 369
Device panel
  Conservative Register Optimization 122
  Implementation Options dialog box 347
dialog boxes
  Implementation Options 346
directives
  SEARCHFILENAMEONLY 373
  beta features 372
  ignore syntax check 371
  IGNORE_VERILOG_BLACKBOX_GUTS 371
  specifying for the compiler (Verilog) 369
disabling sequential optimizations 121
display settings
  Project view 431
Dissolve Instances command 421
Dissolve to Gates command 421
dissolving instances 421
duplicate modules (Verilog)
  Tcl option 117

E

Edit Attributes command 484
Edit Compile Point Constraints command 485
Edit menu 311
  Advanced submenu 313
Edit Module Constraints command 485
Editor Options command 424
Enable Slack Margin 405
encoding
  enumeration, default (VHDL) 359
  state machine
    displaying 465
encryptIP script
  command-line arguments 53
  syntax 53
encryptP1735 script 56
  command-line arguments 56
  public keys repository file 57
  syntax 56
  use models 59
enumeration encoding, default (VHDL) 359
environment variables
  accessing, get_env Tcl command 66
examples
  Tcl find command syntax 151
Exit command 307
Expand command
  current level 414

hierarchical 414
Expand Inwards command 414
Expand Paths command
current level 415
hierarchical 414
Expand to Register/Port command
current level 415
hierarchical 414
expanding
paths between schematic objects 414
export project Tcl command 66
Extract Parameters 369

F

false paths
architectural 248
clocks as from/to points 257
code-introduced 248
defined 248
POS 254
FDC
create_clock constraint 263
create_generated_clock 265
reset_path 269
set_clock_groups 271
set_clock_latency 277
set_clock_uncertainty 279
set_false_path 281
set_input_delay 284
set_max_delay 287
set_multicycle_path 290
set_output_delay 294
set_reg_input_delay 297
set_reg_output_delay 298
standard collection commands 204
File menu
Recent Projects submenu 307
File Options command 473
files
.ta *See also* timing report file 403
adding to project 23, 334
constraint 37
copying 467, 468
include 24
log. *See* log file
opening recent project 307
organization into folders 431
project 94
removing from project 333
replacing in project 337
srs *See* srs file
stand-alone timing report (.ta) 400
temporary 422
timing report. *See also* timing report file 403
Filter Schematic command 415
popup menu 461
filtering
critical paths 419
FSM states and transitions 327
paths from pins or ports 420
selected objects 415
timing reports 402
Find again command 312
Find command
HDL Analyst 317
Text Editor 312
find command
batch mode 80
filter properties 155
finding
critical paths 419
Flatten Current Schematic command
filtered schematic 417
unfiltered schematic 416
Flattened Critical Path command 413
flattened schematic, creating 413
Flattened to Gates View command 413
Flattened View command 413
flattening
instances 421
schematics 416
Floating License Usage command 455
folders
adding to project 27
folders for project files 431
foreach_in_collection command 208
Forward command 327
FPGA Implementation Tools command 454
FPGA timing constraints 262
from points

clocks 256
multiple 251
object search order (Timing Analyzer) 406
objects 250
timing analyzer 406
FSM Explorer command 382
FSM Table command 328
FSM Viewer
popup menu 463
popup menu commands 463
FSMs
optimizing with FSM Compiler 125
Full View command 326

G

Generated Clocks panel, SCOPE 223
get_env Tcl command 66
get_object_name command 210
get_option Tcl command 66
Goto command 312
Goto Net Driver command
current level 415
hierarchical 414

H

HDL Analyst
Find command 317
Visual Properties 327
HDL Analyst menu 412
Current Level submenu 414
Hierarchical submenu 414
RTL submenu 413
Select All Schematic submenu 423
Select All Sheet submenu 423
Technology submenu 413
HDL Analyst Options command 425
HDL Analyst tool
displaying timing information 419
HDL parameter overrides 68
hdl_define Tcl command 67
hdl_param Tcl command 68
Help command 455
Help menu 455
Hide Instances command 420
hiding instances 420
Hierarchical Critical Path command 413
Hierarchical View command 413
hierarchy
flattening 416
Hierarchy Browser
commands 483
popup menu 483
refreshing 483
hierarchy browser
enabling/disabling display 446
hierarchy separator 299
High Reliability panel
Implementation Options dialog box 357
How to Use Help command 455

I

I/O constraints
multiple on same port 230
I/O Standards panel, SCOPE 236
impl Tcl command 72
implementation options
Options Panel 349
Implementation Options command 333, 346
Implementation Options dialog box 338, 346
Constraints panel 351
Device panel 347
High Reliability panel 357
Options panel 349
Place and Route panel 379
Timing Report panel 355
Verilog panel 362
VHDL panel 358
implementation options, device
partdata tcl command 82
Implementation Results panel
Options for implementation dialog box 353
implementations
creating 333
naming 468

include command
verilog library directories 366
include files 24
index_collection command 210
Inputs/Outputs panel, SCOPE 228
instances
dissolving 421
expanding paths between 414
expansion maximum limit 447
expansion maximum limit (per filtered sheet) 451
expansion maximum limit (per unfiltered sheet) 451
finding by name 312
hiding and unhiding 420
isolating paths through 420
making transparent 421
name display 445
selecting all in schematic 423
Instances command
schematic selection 423
sheet selection 423
IP
license queuing syntax 142
IP core wizard 383
IP cores (SYNCore)
building ram models 389
Isolate Paths command 420

J

Job Status command 383, 387
job tcl command 74

L

labels, displaying 445
Launch Identify Instrumentor command 387, 389
levels
See hierarchy
license
floating 455
saving 456
specifying in batch mode 141
License Agreement command 455
license queuing 143
Limit Number of Paths 405
Linux, 64-bit mapping 350
Log File
HTML 330
text 330
log file
displaying 326
Tcl commands for filtering 144
Log File command
View menu 330
Log Watch window
popup menu 460
Log Watch Window command 326
log_filter Tcl command
syntax 75
log_report Tcl command 77
Lowercase command 313

M

maximum parallel jobs 428
memory compiler 389
memory, saving 422
menubar 16
menus
context-sensitive
See popup menus
Edit 311
HDL Analyst 412
Help 455
Options 424
popup
See popup menus
Project 333
Run 381
View 325
Messages
Tcl Window command 325
Mouse Stroke Tutor command 455
multicycle paths
clocks as from/to points 256
examples 246
POS 254
using different start/end clocks 245

multiple drivers
resolving 129
Multiple File Compilation Unit
Verilog panel 365
multiple projects
displaying project files 432
MultiProcessing
Continue on Error mode 427
multiprocessing
maximum parallel jobs 428

N

naming rules 298
net drivers
displaying and selecting 414
netlist formats
Implementation Options dialog box, Implementation Results panel 355
nets
expanding hierarchically from pins and ports 414
finding by name 312
selecting instances on 414
New command 306
New Implementation command 338
New Project command 306
Next Bookmark command 312
Next Error command 384
Next Sheet command 327
Normal View command 326

O

object prefixes
Tcl find command 149
object properties
annotated properties for analyst 155
object search order (Timing Analyzer) 406
object types
Tcl find command 149
objects
displaying compactly 446
expanding paths between 414
filtering 415
unselecting
all in schematic 423
Open command
File menu 306
Open Project command 306
open_design command 80
open_file command 81
opening
project 306
operators
Tcl collection 164
option settings
reporting 66
options
setting 113
Options for implementation dialog box
Implementation Results panel 353
Options menu 424
Options panel
Implementation Options dialog box 349
output files
log. *See* log file
srs
See srs file
overriding FSM Compiler 121

P

Pan command 326
parameters
overriding HDL 68
partdata tcl command 82
Paste command 312
path delays
clocks as from/to points 257
path filtering 405
paths
expanding hierarchically from pins and ports 414
pins
displaying names 445
displaying on transparent instances 421
expanding hierarchically from 414

expanding paths between 414
isolating paths from 420
maximum on schematic sheet 451
place & route
run from the synthesis tool 479
Place and Route panel
Implementation Options dialog box 379
place and route tcl commands
job 74
pointers, mouse
zoom 326
popup menus
FSM Viewer 463
Hierarchy Browser 483
Log Watch window 460
Project view 466
RTL view 483
Tcl window 461
Technology view 483
ports
displaying names 445
expanding hierarchically from 414
expanding paths between 414
finding by name 312
isolating paths from 420
selecting all in schematic 423
Ports command
schematic 423
sheet 423
POS
interface 253
preferences
project file display 431
Preferred License Selection command 456
prefixes
Timing Analyzer points 406
Previous bookmark command 312
Previous Error/Warning command 384
Previous Sheet command 327
primitives
internal logic, displaying 446
Print command 306
Print Setup command 306
printing
view 306
printing image
Create Image command 306
Product of Sums
See POS
program_terminate command 83
program_version command 84
project files
organization into folders 431
Project menu 333
commands 333
Project Options command 470
project Tcl command 85
Project view
display settings 431
popup menu 466
setting up 429
Project View Options command 424
project_data Tcl command 93
project_file Tcl command 94
project_folder
Tcl command 96
projects
adding files 334
closing 307
creating (Build Project) 306
creating (New) 306
displaying multiple 432
opening 306
properties
find command 155
project 93
Push Tristates
Verilog panel 364
Push/Pop Hierarchy command 327

Q

quitting a synthesis run 387

R

recent projects, opening 307
recording command 104
Redo command 312
Refresh command 460

Registers panel, SCOPE 231
regular expressions
  Tcl find command 150
Reload command 483
Remove Files From Project command 333
Remove Implementation command 468
remove_from_collection command 212
Replace command
  Text Editor 312
replacing
  text 323
report_clocks command 105
report_messages command 106
reports
  timing report (.ta file) 400
reset_path timing constraint 269
Resolve Multiple Drivers option 129
resolving conflicting timing constraints 258
resource sharing
  Resource Sharing option 351
Resynthesize All command 381
RTL view
  displaying 81
  opening hierarchical view 413
  popup menu 483
  popup menu commands 483
Run All Implementations command 383
Run menu 381
Run Tcl Script command 383, 384
running place & route 479

S

sar file
  Archive Project command 339
Save All command 306
Save As command 306
Save command 306
schematic objects
  displaying compactly 446
  expanding paths between 414
  filtering 415
  unselecting all 423
schematics
  displaying labels 445
  flattening 416
  navigating sheets 326
  opening hierarchical RTL 413
  sheet connectors 446
  unselecting objects 423
SCOPE
  Attributes panel 235
  clock groups 219
  Clocks panel 218
  Collections panel 225
  Compile Points panel 237
  Delay Paths panel 232
  Generated Clocks panel 223
  I/O Standards panel 236
  Inputs/Outputs panel 228
  Registers panel 231
  TCL View 240
SCOPE spreadsheet
  popup menu commands 460
  starting 216
SCOPE timing constraints summary 217
sdc
  standard sdc collection commands 204
sdc2fdc utility 111
Select All command 312
Select All States command 328
Select in Analyst command 461
Select Net Driver command
  current level 415
  hierarchical 414
Select Net Instances command
  current level 415
  hierarchical 414
Select Place & Route option file dialog box 480
Selected command 327
sequential elements
  naming 299
sequential optimizations
  disabling 121
Set Library command 333
set modules command (collections) 227

set modules_copy command (collections) 227
Set Slack Margin command 419
Set VHDL Library command 333
set_clock_groups timing constraint 271
set_clock_latency timing constraint 277
set_clock_uncertainty timing constraint 279
set_false_path timing constraint 281
set_hierarchy_separator command 299
set_input_delay timing constraint 284
set_max_delay timing constraint 287
set_multicycle_path timing constraint 290
set_option
  Resolve Multiple Drivers 129
set_option Tcl command 113
set_output_delay timing constraint 294
set_reg_input_delay timing constraint 297
set_reg_output_delay timing constraint 298
set_rtl_ff_names command 299
settings
  reporting option 66
sheet connectors 446
Show All Hier Pins command 421
Show Compile Points command 470
Show Context command 420
Show Critical Path command 419
Show Timing Information command 419
sizeof_collection command 213
slack
  margin
    setting 419
  slack margin 405
srm file
  hidden logic not saved 422
srr file
  See log file
srs file
  hidden logic not saved
start/end points
  Timing Report panel, Implementation Options dialog box 356
state machines
  See also FSM Compiler, FSM viewer, FSMs.
  displaying in FSM viewer 423
  encoding
    displaying 465
  filtering states and transitions 327
Status Bar command 325
status_report Tcl command 136
stopping a synthesis run 387
symbols
  enabling name display 445
  finding by name 312
syn_reference_clock attribute
  effect on multiple I/O constraints 231
syn_tristatetomux attribute
  effect of tristate pushing 368
SYNCore wizard 383, 389
Synopsys FPGA implementation tools
  product information 454
Synopsys FPGA products 454
Synopsys Home Page command 454
synplify command-line command 141
synplify_pro command-line command 141
syntax
  bus dimension separator 300
  bus naming 300
Syntax Check command 382
synthesis
  stopping 387
Synthesis Check command 382
synthesis jobs
  monitoring 387
synthesis_off directive, handling 359
synthesis_on directive, handling 359
Synthesize command 381
SystemVerilog 364

T

Tcl

c_diff collection command 165
c_intersect collection command 166
c_list collection command 167
c_print collection command 168
c_symdiff collection command 169
c_union collection command 170
collection commands 164
set_modules collection command 172
-verilog argument 23
-vhdl argument 23
Tcl (Tool Command Language) 14
tcl argument
-_include 24
Tcl collection commands 164
c_diff 165
c_intersect 166
c_list 167
c_print 168
c_symdiff 169
c_union 170
set_modules 172
Tcl collection operators 164
Tcl commands
add_file 23
add_folder 27
collections 226
constraint_file 37
get_env 66
get_option 66
hdl_param 68
impl 72
log file commands 144
project 85
project_data 93
project_file 94
project_folder 96
set_option 113
Tcl conventions 14
Tcl expand command 161
Tcl find command 147
case sensitivity 150
examples 151
object prefixes 149
object types 149
regular expression syntax 150
special characters 150
wildcards 150
TCL Help command 455
Tcl Script
Tcl Window command 325
Tcl scripts
running 383, 384
Tcl shell command
sdc2fdc 111
TCL View, SCOPE 240
Tcl window
popup menu 461
Tcl Window command 325
Technical Resource Center
specifying PDF reader (UNIX) 453
specifying web browser (UNIX) 453
Technology view
creating 413
popup menu 483
popup menu commands 483
technology view
displaying 81
text
copying, cutting and pasting 311
replacing 323
Text Editor
popup menu commands 461
through constraints
point-to-point delays 233
through points
clocks 257
lists, multiple 253
lists, single 252
multiple 253
product of sums UI 253
single 252
specifying for timing exceptions 252
specifying for timing report 404
timing analyst
generating report 400
timing analyzer
wildcards 409
timing constraints
checking 382
conflict resolution 258
constraint priority 258
create_clock 263
create_generated_clock 265
FPGA 262

reset_path 269
set_clock_groups 271
set_clock_latency 277
set_clock_uncertainty 279
set_false_path 281
set_input_delay 284
set_max_delay 287
set_multicycle_path 290
set_output_delay 294
set_reg_input_delay 297
set_reg_output_delay 298
timing exceptions
False Paths panel 248
multicycle paths 245
priority 258
specifying paths/points 248
timing information, displaying (HDL Analyst tool) 419
timing report
asynchronous clock report 402
defining through points 404
file (.ta) 400
specifying slack margin 405
using path filtering 405
timing report file
generating custom 401
stand-alone 403
Timing Report panel
Implementation Options dialog box 355
Number of Critical Paths 356
Start/End Points 356
timing reports
file. *See* timing report file
filtering 402
parameters 400
stand-alone 400
stand-alone (.ta file) 400
Tip of the Day command 456
to points 406
clocks 256
multiple 251
objects 250
Timing Analyzer 406
Toggle bookmark command 312
Toolbars command 325
tooltips
displaying 329
transparent instances
displaying pins 421
tristates
pushing tristates, description 366
pushing tristates, example 366
pushing tristates, pros and cons 367

U

Uncomment Code 313
Undo command 312
Unfilter command 328
unfiltering 420
FSM diagram 328
schematic 420
Unflatten Current Schematic command 417
Unhide Instances command 420
unhiding hidden instance 420
UNIX
configure external programs 425
Unselect All command 423
View menu (FSM Viewer) 328
Uppercase command 313
utilities
sdc2fdc 111

V

variables
accessing, get_env Tcl command 66
reporting 66
VCS Simulator command 383
Vendor Constraints
Implementation Results panel, Implementation Options dialog box 355
writing 355
Verilog
`‘ifdef` and `‘define` statements 369
allow duplicate modules (Tcl option) 117
beta features 372
compiler, configuring 424
extract design parameters 369
library directories 366
specifying compiler directives 369

Verilog 2001
Verilog panel 364
-verilog argument
Tcl 23
Verilog include files
using _SEARCHFILENAMEONLY_ directive 373
Verilog panel 364
Implementation Options dialog box 362
Multiple File Compilation Unit 365
options 364
Push Tristates 364
SystemVerilog 364
version information 456
VHDL
compiler, configuring 424
enumeration encoding, default 359
ignoring code with synthesis off/on 359
-vhdl argument
Tcl 23
VHDL libraries
setting up 337
VHDL panel
Implementation Options dialog box 358
View FSM command 423
View FSM Info File command 423
View Log File command 326
View menu 325
Filter submenu 327
Log File command 330
RTL and Technology view commands 326
View Result File command 326
View Sheets command 327
Visual Properties command 327

W

web browser
specifying for UNIX 453
wildcards
Tcl find command 150
text Find 314
text replacement 323
timing analyzer 409
Windows, 64-bit mapping 350
Write Output Netlist Only command 382

Z

zoom mouse pointer 326
Zoom Out command 326